THE ROLE OF INTELLIGENT MOBILE AGENTS IN NETWORK MANAGEMENT AND ROUTING

Vinay Gopal Balamuru, M.S.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2000

APPROVED:

Armin Mikler, Major Professor

Paul Tarau, Committee Member

Robert Renka, Committee Member

Tom Jacob, Chair of the Department of
Computer Science and Committee Member

C. Neal Tate, Dean of the Robert B. Toulouse
School of Graduate Studies

Balamuru, Vinay Gopal, The Role of Intelligent Mobile Agents in Network Management and Routing. Master of Science (Computer Science), December 2000, 86 pp., 3 tables, 68 Figures, 39 references, 40 titles.

In this research, the application of intelligent mobile agents to the management of distributed network environments is investigated. Intelligent mobile agents are programs which can move about network systems in a deterministic manner in carrying their execution state. These agents can be considered an application of distributed artificial intelligence where the (usually small) agent code is moved to the data and executed locally. The mobile agent paradigm offers potential advantages over many conventional mechanisms which move (often large) data to the code, thereby wasting available network bandwidth. The performance of agents in network routing and knowledge acquisition has been investigated and simulated. A working mobile agent system has also been designed and implemented in JDK 1.2.

ACKNOWLEDGEMENTS

I, Vinay Gopal Balamuru, would like to acknowledge all those who helped make this project a reality. In particular, I would like to acknowledge my parents, without whom none of this and more would have come to pass the way it did, Dr. Armin Mikler and Dr. Paul Tarau for their invaluable advice, guidance and patience at all stages of this project and ”UB” for permitting the use of a couple of his graphs. Last but not least, I thank Milind Belhe, Jatin Bavishi, Satyam Tyagi, Hemil Chhatbar and Sachin ChandraMohan for being good friends.

In closing, I must acknowledge God, Fate and Chance for everything in the past, present and future, as appropriate.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Evolution and Definition

In this research, the application of intelligent mobile agents to the management of distributed network environments is investigated. Mobile Agents are, in their simplest form, programs capable of remote execution at judiciously selected locations and can be considered to be an application of distributed artificial intelligence where the (usually small) agent code is moved to the data and executed locally as opposed to many conventional mechanisms which move (often large) data to the code. The performance of agents in network routing and knowledge acquisition has been investigated and simulated. In addition, a mobile agent system has been designed and developed using JDK 1.2.

1.1.1 Mobile Agents: A case of Deja Vu?

On November 2, 1988, a Cornell Graduate Student, Robert Tappan Morris, released a worm program into the Internet [1]. This was a piece of code that exploited bugs in the Berkeley UNIX Operating System that made it possible to gain unauthorized access to a very large number of computers all over the Internet. The program consisted of a bootstrap which was compiled and executed on the machine under attack. Once running, it connected to the machine which it came from, uploaded the main worm and then executed it. The execution involved the worm attempting to spread itself to all other computers specified in the host's routing tables. The overall effect on the Internet was catastrophic. Further details of the attack, and the resulting consequences, are beyond the scope of this document.

Regardless of the controversy generated by the worm, the issue represents a prime example showcasing the power of the mobile computing paradigm. The mobile code itself is commonly known as a "Mobile Agent". The capability to transport and execute code or data at remote resources opens the door to exciting new possibilities, some of which will be addressed in the following sections.

1.1.2 What is a Mobile Agent"

JavaWorld [2] describes the concept of agents as follows.

"In a broad sense, the precepts of agent technology exist in many of the applications we use today and take for granted. For example, your e-mail client is a type of agent. At your request, it goes about its business of collecting your unread e-mail from your mail server. Contemporary e-mail clients will even presort your incoming messages into specified folders based on criteria you define. In this manner, the software becomes an extension of the user, performing tasks on the user's behalf. Indeed, the computer itself can be considered an agent, as its primary task is to increase productivity through automation."

More interesting, however, is the concept of an "intelligent agent" which employs a logical approach to solving the task at hand. If the capability to transport themselves between different network nodes is granted to the agents, they become they become "intelligent mobile agents". Intelligent Mobile Agents could therefore be defined as mobile code or programs existing in a network which are capable of making independent or collaborative decisions which influence their behavior and actions.

1.2 Mobile Agents: Their Place in the Sun

Eventually, the worth of a technology is determined by the success of the applications that can be developed using it. This section describes the motivating forces behind the development of mobile agent technology. The mobile agent paradigm and its advantages is also described.

1.2.1 The need for change

In this section, an attempt is made to address various aspects of mobile agent technology. Conventional network communication models are described and compared with the agent model. Much of the following information was found at the General Magic [3] web site.

Technology today is growing in leaps and bounds and places powerful new tools and applications in the hands of the consumer. Much of this technology involves the use of devices or applications that either query remote services in order to get their desired information (centralized), utilize some kind of client-server paradigm or perhaps use some kind of remote procedure call (RPC) [3]. In essence, the RPC protocol enables one computer to call procedures on another remote computer. Each message transported by the network either requests or acknowledges the performance of some procedure (which is internal to the computer that performs it). One limitation of the RPC model is the amount of overhead involved, both computational and network-based. Figure 1.1 describes the operation of the RPC model.

However, the excessive amount of messages and data passed over communication links can generate a large amount of traffic, a fraction of which is either redundant or even altogether unnecessary. It could be hypothesized that an improved solution to the network traffic problem would involve an altogether different computational and communication paradigm.

One such alternative to current approaches is the use of remote programming (RP)[3] (described by Figure 1.2) in which the procedures can *physically* move from one computer to the other carrying the data and state of the computation. The

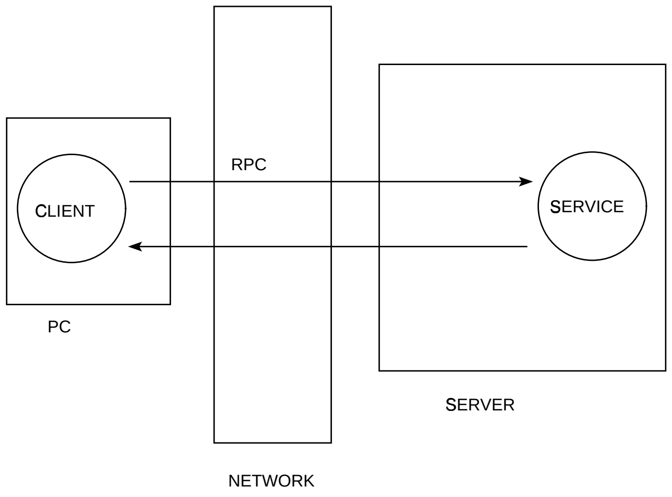

Figure 1.1: The RPC Model

procedures themselves are executed locally. It should be understood that the communicating computers need to agree beforehand (before any computation is executed) about the kind of instructions and syntax (but not the procedures) that are acceptable. In short, they should both understand the same language. The procedure(s) and the state(s) in this case comprise the Remote Program. It should be stressed that remote programming (RP) is not the same as remote procedure calls (RPC) where the procedures are executed remotely by way of some kind of virtual interface available to all entities involved in the computation [3].

Mobile Agents can be considered to be an extension of the RP concept wherein the procedures can move between a number of computers carrying out a variety of tasks. The extent of the capabilities of mobile agents is related to the design of the agent delivery and execution system, as well as the logical reasoning capabilities programmed into the individual agents themselves. Figure 1.3 illustrates a simple mobile agent system where agent code is loaded into a client which launches it to a server. Once on the server, the agent can execute and decide to move to another

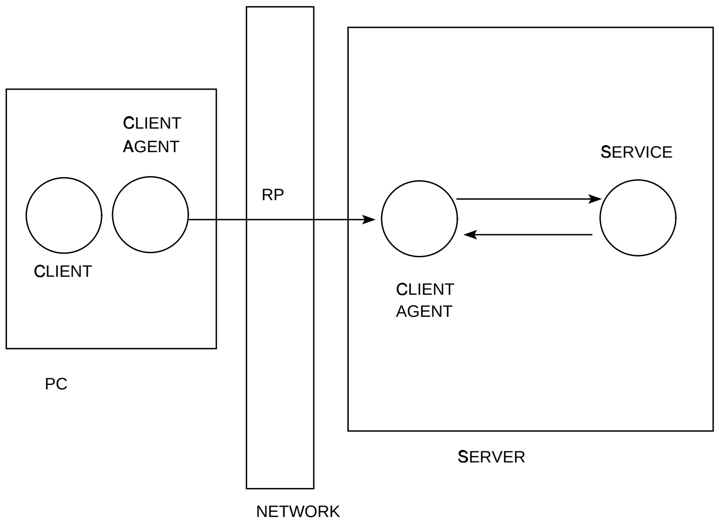

Figure 1.2: The RP Model

remote server, if appropriate. It should be noted that in the mobile agent model, each server acts as a client in order to deliver the agent to its next host. This is an improvement over the RP model where the roles of client and server are easily decoupled.

1.2.2 Advantages of Mobile Computing

While the previous section dwelt on various computing paradigms, this section attempts to describe, in greater detail, the features and advantages specific to mobile-agent based systems.

- Simplicity: Agent systems in general are much simpler to apply to a variety of applications. The concept implicitly acknowledges the possibility of distributed resources and the need to travel across the network in order to access them. Agents are easily adapted to perform a variety of tasks, as opposed to centralized systems.

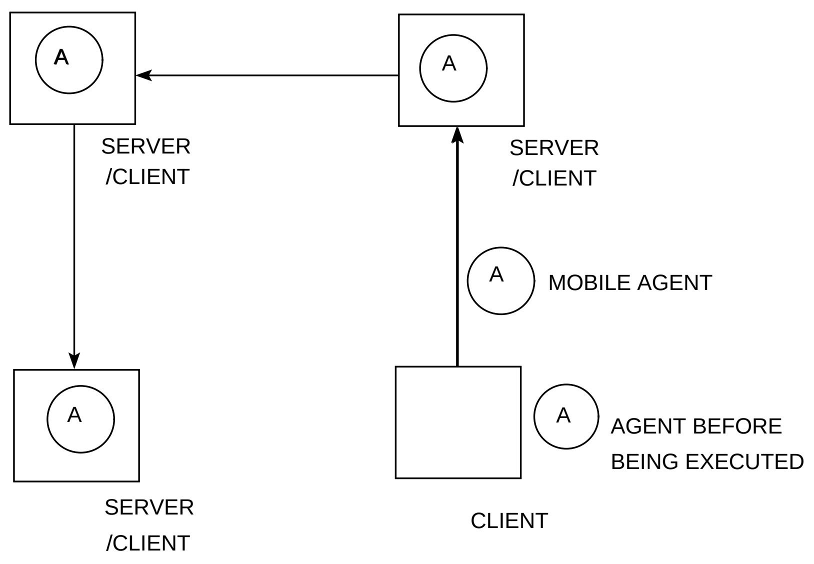

Figure 1.3: A Simple Mobile Agent Model

- Scalability: Mobile agent applications can scale well to varying network sizes and configurations since agents could potentially travel freely over the network and clone or merge (as described later) when necessary.The scalability of agent systems assumes greater importance when dealing with dynamically changing network configurations (topology and connectivity).

- Interaction: Both computers can interact without requiring ongoing network communication once the agent has been transferred between them. Therefore, large tasks can be delegated with a minimum of message passing overhead. This is in contrast to RPC or traditional client - server models where a greater amount of communication is required to synchronize the operation between various clients and servers, especially in the execution of complex tasks. This is because in agent systems, the code moves to the data, in contrast to client server systems or RPC, where the relevant data or control information has to be passed over the network every time a decision has to be made.

- Communication Cost: As a consequence of a lower network traffic overhead resulting from mobile agent systems, the communication cost to the user is reduced. This assumes greater importance to users or subscribers who have to pay for bandwidth or network usage time.
- Customization: Agents can allow software manufacturers to add to the functionality of user software by adding procedures to the agent repertoire without having to modify anything on remote computers. This makes software installation and modification easier.
- Collaboration: Mobile agents can be programmed to communicate with other agents at the same physical location with the intent of sharing information. The communication could be by using a shared resource, such as a database or hash table or by direct interprocess communication or socket calls. The collaborative capabilities of agents is useful in data gathering tasks where agents can use information gathered by other agents.
- Fault Tolerance:In cases where more than one agent carries the the same information in its data segment, the loss of a few agents is unlikely to affect the overall system performance drastically.

Harrison et al [4] summarize the issue by noting that:

"While none of the individual advantages of mobile agents is overwhelmingly strong, we believe that the aggregate advantages of mobile agents is overwhelmingly strong, because:

- *They can provide a pervasive, open, generalized framework for the development and personalization of network services.*

- *While alternatives to mobile agents can be advanced for each of the individual advantages, there is no single alternative to all of the functionality supported by a mobile agent framework. "*

Several working agent systems have been developed (see the next chapter) and have reached a stage of maturity. In the future, the focus of agent research should be the development of agent interaction and operational logic, in order to efficiently utilize the capabilities of the available technology.

1.2.3 The Mobile Agent Terminology

Once it has been established that the mobile agent concept is theoretically sound, it becomes necessary to identify concepts associated with the technology. These are listed and described briefly as follows [3] :

Places: A place refers to any location that an agent can enter and reside. A place acts as a portal for the agent to execute upon. To illustrate the concept of a place, consider a network representing a shopping center. It could contain a ticket place where agents could buy tickets, a flower place where agents could buy flowers and a directory where agents could learn about various accessible places.

Agents: The agent is a program together with its associated execution state that moves from computer to computer. An agent system might comprise several identical or different agents performing their assigned tasks either cooperatively or independently.

Travel: This is the act of moving from one *physical* location (place) to another. All mobile agents should be able to travel, at some point or the other during their lifetime. For instance, in our example, an agent might travel to a ticketing place, purchase a ticket, and then travel to a flower place, to place an order for flowers, and then travel back home. In order to travel, an agent must know its destination (or sequence of destinations) and should be able to handle and recover from the exceptions generated if any destination is, for whatever reason, unreachable.

Meeting: A meeting is said to occur when multiple agents present at a single place interact. The concept of different agents meeting at a single place is primary to the collaborative capabilities of mobile agents.

Connection: This is a mode of communication where agents present at different places can "speak" by message passing. A possible use of connections would be for a remote agent to contact the human user at a specified location with information it has gathered. It could then, say, execute a transaction based on the final decision of the human operator.

Security / Rights: This assumes importance in order to ensure that malicious code is not executed on machines. Therefore, the agent system should have some sort of verification mechanism built-in.

Agent Language: An agent programming language defines the operations and type of information that can be associated with the mobile agents. Several agent languages have been developed, and are still being developed, with varying capabilities. Some examples of agent languages are TACOMA [11], Ara [12] and Agent-Tcl [13]. Agent languages are described in greater detail in another section.

Agent System / Server: The agent system is essentially a program which runs on each computer that an agent is expected to move to. It serves to receive and house the agent (which usually moves in the form of a string) and process it as required. It can usually draw upon the resources of its host computer as required by the agent.

Agent Protocol: Agent protocols enable two agent systems to communicate in order to transport agents over the network. The protocol suite can operate over a variety of transport networks such as TCP/IP, X.25, e-mail etc. The agent protocol operates at two levels. The lower level governs the transport of agents while the higher level is concerned with their encoding, preservation of state information, decoding and cryptography (if implemented).

In conclude the chapter, it can be stated with conviction that mobile agents are a powerful tool to solve complex and diverse tasks in the distributed environment. The agents can easily be customized with the programmer's goals in mind, since they, by nature, operate in a distributed autonomous manner and can interact when necessary. The possible applications can be as diverse as distributed data mining to multi-agent negotiations. The next chapter describes some real-life applications of mobile agents. Various mobile agent systems are described and the issue of security in the mobile agent scenario is addressed in some depth.

CHAPTER 2

MOBILE AGENTS IN NETWORK MANAGEMENT

2.1 Mobile Agents Today

System management today has become a complex and involved task, due in part to the rich diversity of components which comprise the typical distributed computing environment. The very definition of the term "System Management" becomes vague in the case of Multi-Vendor Environments. Intelligent agents offer a promising approach to this issue. The agents are defined in [5] as *"special software that is installed on managed stations on the LAN or WAN, which is used to collect performance information in a standard format and implement preemptive actions based on predefined policies"*. Since the agents send only items connected to the management process (such as CPU utilization, disk space and I/O), the operation becomes more efficient than continuously polling every station, hence minimizing traffic and consuming less bandwidth. The information collected can provide an early warning of potential problems and take appropriate actions. Kramer et al [6] have studied the issue of knowledge acquisition and routing in some detail and have amongst other things, developed co-operative adaptive agent behavioral strategies for network management which are based on social insect models.

Commercial Users of Agents Some organizations which which utilize agent technology [5] include Computer Associates (CA-Unicenter), Hewlett-Packard (Operations-Center) and Saber Software (LAN Management System). Several other companies are also considering the use of agents in their systems [5]. It should be noted that agents themselves have been used for some time now in distributed systems, for instance the Simple Network Management Protocol (SNMP). What is new is the use of Intelligent Agents to manage systems-level activities. The agents could be used to carry out tasks such as:

- Data Visualization
- File and Information Distribution
- Software Updates and Management
- Network Management and Monitoring
- Disk and File Monitoring and Management
- Creation of task schedule.
- Management and monitoring of Active Networks

Some representative applications are described in the following section.

2.2 A Survey of Mobile Agent Frameworks

This section briefly describes some available mobile agent frameworks and their applicability. A deeper investigation of these languages and the underlying associated issues has been undertaken by Fuggeta et al [15].

Agent Tcl: Agent Tcl was developed at the University of Dartmouth [16]. It provides an extension of the Tcl[13] interpreter which supports the transport of agent code across the network. Each agent is implemented by a Unix process running a Tcl interpreter. When transferred in a network, the agents are received by agent servers running on each local machine. Agent Tcl has been used in distributed data searching applications (library and medical records), work-flow applications (e-mail and purchase orders) and distributed database querying applications [27].

Ara: Ara is a mobile agent platform that supports multiple languages (C / C++ / Tcl). The agents are executed using language interpreters which exist upon a run-time core. Ara supports strong mobility (movement of both code and system state). thus relieving the programmer from the implementation details of communication protocols. Ara uses a flexible security model which allows fine-grained admission and execution control of mobile agents. However, Ara lacks in the areas of structured agent inter-operation and supportive services for distributed resource discovery (used in some real world applications). Development work on the Ara system is mainly focussed on system support and security, as opposed to application level features. Applications are typically weak-connection / high-volume environments like wireless or intermittently connected computers which impose restrictions on the bandwidth or connectivity vs. data volume ratio [30].

Facile: Facile [18] was developed as a multi-paradigm programming language combining the elements of both functional and concurrent programming. Agents are implemented as threads which run within computational environments known as nodes. Facile provides safe execution of mobile agents because they only have access to explicitly granted resources . Facile has been used to implement distributed teleconferencing and graphics applications. Facile can be considered a tool which supports the development of agents, rather than an agent language itself. For instance, Knabe [28] has used Facile to implement a Mobile Service Agent (MSA) capable of various tasks in the network domain.

Java: Java [19] was developed by Sun Microsystems as a flexible object oriented language with a wide application sphere. Java programs are compiled to byte code by the Java compiler. This can be executed by a Java virtual machine. Java can support weak mobility by using "Object Serialization" whereby class instances, known as "objects", can be written to files or sockets, transported across the network, and reconstructed at other physical locations.

Java Aglets: Aglets [20] were developed by IBM Tokyo Research Laboratory in Japan. This system extends the capability of Java [19] to transport objects between hosts on the Internet. When Aglets move, they can take their code with them to use on the next machine. The Aglets system uses a built-in security mechanism which protects the system from malicious or untrusted agents. Aglets have been used to implement an agent-based middle-ware called "e-Marketplace" which is geared towards the Internet shopping market [29].

Jinni: JINNI [31], the Java INference engine and Networked Interactor, is a lightweight, multi-threaded, logic programming language, intended to be used as a flexible scripting tool for gluing together knowledge processing components and Java objects in networked client/server applications, as well as through applets over the Web. Mobile threads, implemented by capturing first order continuations in a compact data structure sent over the network, allow Jinni to inter-operate with remote high performance BinProlog servers for CPU-intensive knowledge processing and with other Jinni components over the Internet. Jinni can be used as a framework to develop agent applications for, amongst others, stock market trading, mobile devices and educational environments [31].

Mole: Mole [21] is a Java based mobile agent API that was developed at the University of Stuttgart. The authors claim that it was the first mobile agent system developed in the Java language. Mole executes agents as Java threads. and supports weak mobility. Applications mentioned by Tschudin [21] include the infrastructure for an electronic documents system, a network management simulator and as entities in Multi User Dungeon (MUD) environments.

Obliq: Obliq was developed at DEC [22]. Obliq is a lexically-scoped, untyped, interpreted language that supports distributed object-oriented computation. Obliq objects have states and are local to a site. Obliq computations can roam over the network, while maintaining network connections. Obliq achieves mobility by the use of mobile threads which execute procedures on a remote execution engine. Obliq supports weak mobility using a mechanism for synchronous shipping of stand-alone code. Obliq technology has been used to enable dynamic application migration[24] and also as a distributed application builder (in the form of Visual Obliq) [23].

Sumatra: Sumatra [26], developed at University of Maryland, is a mobile agent system designed with an intent to support resource-aware mobile programs. In other words, they are able to handle asynchronous events and possibly react by moving the code to a different site. Sumatra provides strong mobility. The use of Sumatra in creating network-aware mobile programs which can adapt to variations in network conditions has been documented by Ranganathan et al [25]. In this application, Adaptalk, a Java based Internet chat application takes advantage of the agent support to dynamically place the server, hence reducing the response time.

TACOMA: The TACOMA (Tromso And Cornell Mobile Agents) [11] mobile agent system focuses upon the provision of operating systems support for mobile agent based computing systems. The agents are written in Tcl. The agent servers are implemented by using the services of the Unix operating system. The agent data is stored in a structure known as a "briefcase" and the system data is stored in another structure known as a "cabinet". Briefcase - Cabinet interactions form the basis of agent interactions with the underlying system. An interesting application of the Tacoma system is in an Internet accessible wide-area weather monitoring system.

Comprehensive lists of various agent language systems and their application areas have been compiled at http://www.agentbuilder.com/AgentTools/index.html and http://www.informatik.uni-stuttgart.de/ ipvr/vs/projekte/mole/mal/Agents-survey-Mamadou.ppt

2.3 Security Issues associated with Mobile Agent Implementations

While previous sections have described the mobile agent concept and the possible benefits, an equally important task is the identification of the possible risks and pitfalls the paradigm introduces. The "Internet Worm" [1] serves as a lasting reminder of the damage that can be wrought by imparting malicious intent to agents. Likewise, it would seem reasonable that the agents themselves could be attacked or corrupted by malicious host environments.

In perspective, it should be noted that very few Internet technologies are inherently secure. Security is usually implemented as a separate layer with the aim of providing a safe intermediate authentication environment. In spite of this, flaws are regularly discovered in many supposedly secure environments. Likewise, in the case of mobile agents, the development of a secure layer is best postponed to a later time when the platform is sufficiently mature and promising as to justify the expense of the development and implementation of security features.The rest of this section attempts to identify various types of attack and possible counter-measures [32] .

Types of Attack The different types of attack conceivable in an agent environment are listed below.

- Agent against Platform: In this case, the agent attacks the execution environment which receives it.

- Platform against Agent: In contrast, a malicious execution platform might seek to attack agents which reside on it.
- Agent against Agent: Malicious agents could also attempt to attack other agents.
- Other entities against Both: External elements could disrupt normal operation by seeking to destroy the agent and its execution environment.

Threats posed by Malicious Mobile Agents Agents can attempt to compromise system security by one or more of these methods.

- Disclosure/Eavesdropping: The agent could attempt to copy or relay sensitive data (which they could gain access to, via the execution environment) thus compromising the privacy of other users or systems.
- Alteration/Corruption: Alternatively, the agent could try to corrupt local data, possibly confusing other computer programs.
- Denial of Service/Resource Consumption: Mobile agents can launch denial of service attacks by consuming an excessive amount of the agent platform's computing resources, possibly by running attack scripts, uncontrolled spawning etc.
- Spoofing/Masquerading: An agent can attempt to disguise its identity in order to gain the trust of the agent platform or other agents

Security Mechanisms: Once various types of attack have been identified, it becomes easier to propose counter strategies to reduce or eliminate the risk associated with allowing mobile code to execute on a computer network. A few strategies are presented below:

Using an interpreted language to write the agents: This makes it possible to restrict the commands that the agent can execute, thus limiting the amount of potential damage. An example of such an example is Safe Tcl [33]. Safe Tcl initializes the Tcl interpreter to a safe subset of Tcl commands so that the scripts cannot harm the hosting platform. It uses "command aliases" which transparently invokes safe versions of regular Tcl commands.

Agents can be signed: Another technique for protecting an agent system is signing code or other objects with a digital signature which serves as a means of confirming the authenticity of an object, its origin, and its integrity. Because an agent operates on behalf of an end-user or organization, mobile agent systems [34], [35] [36] commonly use the signature of the user as an indication of the authority under which the agent operates.

Proof-Carrying Code: Abstract Proof-Carrying Code (PCC) enables a computer system to determine, automatically and with certainty, that program code provided by another system is safe to install and execute without requiring interpretation or run-time checking [8]. PCC has applications in any computing system in which the safe, efficient, and dynamic installation of code is needed. The key idea is to attach to the code an easily-checkable proof that its execution does not violate the safety policy of the receiving system.

Mobile Cryptography: Agents are produced by converting a agent specification into some executable code plus initial encrypted data. Since the attacker cannot break the encryption of the data, it cannot read or manipulate the original data.

Time Limited Black box Security: A Black box is an agent that performs the same work as the original agent, but is of a different structure [37]. This difference allows to assume a certain agent protection time interval, during which it is impossible for an attacker to discover relevant data or to manipulate the execution of the agent. After that time interval the agent and some associated data become invalid and the agent cannot migrate anymore, which prevents the exploitation of attacks after the protection interval.

Some Partial or Limited Solutions: Where the perceived threat or risk is relatively small, the following methods should be relatively easy to implement. However, it is unlikely that they will fully protect a system from a skilled determined intruder.

- Bar malicious agent platforms from further intercourse: This serves as a damage control mechanism and can only be effective if the attack can be detected early enough.

- Allow agents to travel only among a trusted network of platforms: This approach is limiting, in that it might not allow the agent free access to roam at will.

- Obfuscate agent code: In this approach, the code can be scrambled using a Black Box scrambler [32] so as to confuse any potential cracker. A serious problem with the general technique is that there is no known algorithm or approach for providing true Black box protection.

To this point, this work has dealt with various facets of mobile agent technology; commercial applications, a survey of available systems and the omnipresent issue of security provisions. In the next section, the focus will be shifted to the applicability of mobile agents to various tasks in network management. In particular, simulations are described which evaluate the utility of Mobile Agents in network routing and knowledge acquisition.

2.4 Network Management: The Task at Hand

One of the most notable advancements in modern communication networks has been the increase in communication bandwidth of the network backbone infrastructure. Additionally, new switching technologies, such as Asynchronous Transfer Mode (ATM) and Packet over Sonet(POS), coupled with improved network protocols, have been developed to take advantage of the increase in network capacity. Some of the resultant benefits are the capability to provide different Quality of Service (QoS) to different streams, the ability to provide Real Time steaming Multimedia, efficient Collaborative Learning and viable Virtual Private Networks (VPN).

This drastic increase in available network bandwidth and the development of technologies which have the capability to optimally utilize the available resources, even in dynamically changing communication environments, underscore the need to obtain and maintain an accurate picture of the network state over a period of time. Consequentially, the maintenance of large systems assumes primary importance. While a single Administrator, or a group of suitably qualified individuals, could conceivably monitor and administer and configure systems of reasonable size, the work becomes exceedingly complex and cumbersome to accomplish as the complexity and /or size increases of the network increases. Some possible contributing factors are inter-node communication issues, fault tolerance, node routing characteristics, quality of service and co-ordination between Administrators etc.

In this context, the main challenge is the development of distributed monitoring algorithms that can provide complete and precise system information which can be used for making further decisions relevant to the network management tasks at hand. Currently, monitoring algorithms are a bandwidth hog and can have adverse effects on utilization, due to the excessive amount of "flooding" traffic during the acquisition of resource information Figure 2.1 illustrates a typical scenario . Additionally, the use of traditional mechanisms which rely on a single management station can raise congestion (Figure 2.2) related issues at links connected to the management node.

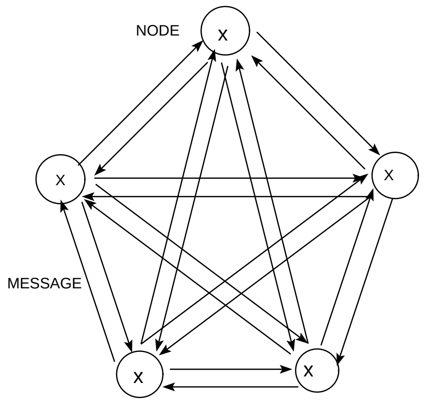

Figure 2.1: Excessive Flooding Traffic

The relevant issues seem to indicate some sort of data-crunching algorithm. It would seem plausible to approach this problem using a Divide and Conquer approach [38], where the network can be partitioned into successively smaller regions of increasing manageability. Unlike a conventional Divide and Conquer problem however, there is no "final static solution" to the issue since the network needs to be constantly monitored. In addition, the network parameters are constantly changing and as such, the actions to be taken vary accordingly. It is also worth noting that networks can span a large geographical area and as such the the solution could involve a distributed algorithm.

Lastly, while it has been stressed that the dynamics of network information variation presents a challenge to knowledge acquisition and representation strategies, the size and complexity of the network as a whole often changes (usually increasing) with time as new entities are connected and sometimes removed from the system. Therefore, a scalable solution is needed to address this issue.

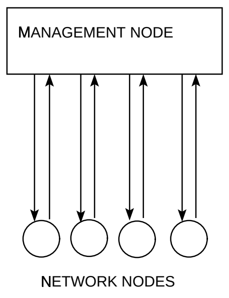

Figure 2.2: Congestion at a Node

Agents, which are essentially computer programs that can perform or assist in the execution of tasks otherwise carried out by users, offer a viable alternative. Agents can perform routine work for users or assist them with more complicated tasks. They can also mediate between incompatible programs and thus generate new, modular and problem-oriented solutions. Since the agents in focus can propagate through the network in order to perform their duties, they can also be referred to as *mobile agents.* The use of Mobile Agents enables the movement of both code and data thus allowing the programmer or user to optimize between the requirements of a low-bandwidth, high-latency or unreliable network connection. Since mobile agents perform a large portion of their work as local computation, they can reduce the amount of bandwidth wasted in message passing, polling etc. They can also reduce the load on a central management stations, a problem with some conventional centralized network management strategies. Figure 2.3 illustrates an instance of a simple management strategy where the agent "walks" through the nodes efficiently executing its tasks with a minimum of network traffic.

The theoretical advantages of using mobile agent systems for network management

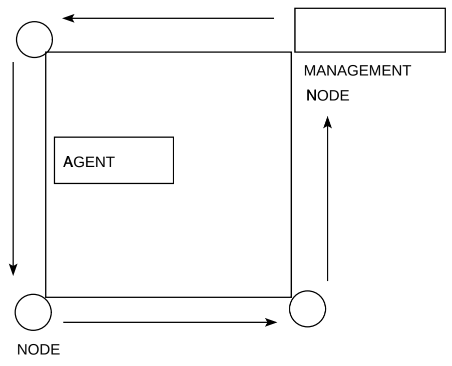

Figure 2.3: A Mobile Agent walking through a network

purposes are manifold. Some benefits are outlined below.

- A well designed mobile agent system can scale well to a change in network topology or size since agents have the ability to adapt to a variety of operational circumstances.
- It is fairly easy to modify or adapt agents to perform duties different from what they were initially intended for.
- If agents can communicate with each other, they can reduce the overall workload by co-operating to avoid redundant work.
- In the case of an increase in workload, agents could possibly adapt by duplicating themselves.
- In the case of a low workload, agents could reduce the network traffic overhead by merging.
- agents can autonomously and independently decide when to return to a central monitoring station to "download" the collected information.

In this work, applications of mobile agents in network management are investigated. The problems investigated are:

1. The Design and Simulation of a Scalable Mobile Agent Based Model for Network Monitoring.
2. The Simulation of the Distance Vector Network Routing Algorithm using Mobile Agents.
3. The design and implementation of a working mobile agent system in Java (JDK 1.2).

In order to best utilize the powerful capabilities of the Mobile Agent paradigm, the agent system should be accommodated at the system level and operate as an integral part of the distributed system. This allows it to transparently perform tasks of vital importance.

2.5 Simulation of a Mobile Agent based Routing System

In order for computers in a network to communicate with each other, it is important for them to be able to "see" each other. This involves a system where packets sent from a source computer can be successfully directed to the destination computer. The inspection and forwarding of network packets is assigned to a class of networking devices known as "routers". Routers are attached to networks and can relay packets to other routers along an optimal path. Often however, there exist multiple paths between routers. In order to achieve optimal routing performance, it is essential that routers collaborate to find the best paths between them. This process is known as "routing".

Various routing algorithms have been developed to determine optimal routing paths. In this work, the Distance Vector Routing algorithm is analyzed and compared to an agent based variant. The input graphs have been generated by algorithms described in the Appendix.

2.5.1 A Brief Overview of the Distance Vector Routing Algorithm

In the Distance Vector Routing algorithm, each node periodically sends out information packets about its routing table to all its directly connected neighbors. The information transmitted by a node includes a list of all reachable nodes, the next hop to get to that node and the cost metric involved in moving to that node. On receiving information from another node, the recipient node compares its routing table entries with that received. If the appropriate cost entry indicated by the received routing table is less than its own cost for some destination, it updates its routing table and sets the propagating node as the next hop for that particular destination.

Since each node goes through the process of sending, receiving and adjusting routing table information, the network eventually attains a state where all nodes know the shortest paths to all other reachable nodes (in the form of the best next hop to the node in question). As the state of the network is liable to change at some point in time, the routers are programmed to continue sending information at regular intervals, regardless of convergence or not.

2.5.2 Simulation of the Distance Vector Routing Algorithms without and with Mobile Agents

In order to study the behavior of the distance vector routing algorithm, a simulator was developed using C++. The desired networks were simulated as an adjacency matrix representation of a graph abstract data type. Each node of the simulation graph has knowledge about nodes it can reach directly, nodes it can reach indirectly, and nodes it cannot reach. All links between nodes are assumed to be bi-directional.

At each iteration of the simulation, every node in the network shares its information with all its directly connected neighbors which then consolidate the information with their routing tables, making changes if necessary. Nodes which cannot be reached are assigned a very high cost, to approximate the value of infinity (which is the theoretical cost between 2 disconnected nodes).

To simulate a version of the Distance Vector Routing algorithm using mobile agents, the agents which are of a fixed quantity, are simulated as tokens moving about the network in a random manner. When an agent relocates to another node, it is assumed to carry the routing table information of the source node (as specified by the distance vector algorithm) to its destination. The destination performs the consolidation and updates its routing table if necessary. The agent then picks up this routing table and migrates to another random neighbor. In this case, the agent-controlled routing eventually converges to a result. The following discussion attempts to compare these two approaches to routing and investigate the results obtained.

In order to compare the performance of these two simulations, it is essential that a common comparative basis be used. In other words, some metric must be identified which equates simulation iterations in the conventional Distance Vector Routing Algorithm to those in the Agent-based simulation. It was decided to equate a single routing message passed between a pair of nodes in the conventional method to the movement of a single mobile agent between a pair of nodes. This is justified by the assumption that the agent code does not contribute a great deal to the amount of bandwidth consumed for routing.

2.6 Simulation of Network Knowledge Acquisition using Mobile Agents

Knowledge Acquisition (KA) involves the determination of the global state of a network system, in terms of various parametric values specific to the individual nodes which comprise the system. The acquired knowledge is often used in other tasks or mechanisms, including resource monitoring, Quality of Service(QoS), Adaptive Routing, Network Management and Fault Tolerance. Conventional centralized approaches to KA do not scale well with an increase in the size an complexity of networks. In addition, they are not suitable for ad-hoc or rapidly changing networks, because they often require a rigid management and communication hierarchy. Mobile agent systems, on the other hand, rely on an inherently distributed approach and should be able to adapt to changing network conditions better.

Knowledge acquisition in computer network systems can be broadly classified into two tasks: the communication between the querying entity and the queried entity (machine address, sequence of machines to be queried, communication issues, reliability and robustness etc) and the actual querying of the information at its source (namely the machine, router etc). The details of information extraction at the location of the network entity is achieved by using data mining and parsing techniques and a detailed discussion is beyond the scope of this project.

Schonwalder [39] has described typical methods by which network entities are polled. He classifies (conventional) network monitoring systems as either passive or active. According to him, passive strategies, while using less bandwidth, are not generally as effective as active, band-width hogging round-robin polling strategies. An Agent knowledge acquisition system could offers potential advantages over the conventional knowledge acquisition strategies just mentioned because:

- It is scalable, adapting dynamically by means of the "wave computation" mechanism to varying network size and configuration.
- The agents can adapt and can switch between a pro-active (cloning) and passive (merging) state as the computation proceeds.
- All the communication is performed locally at the network entity without need

for intervention by any other entities over the network, hence conserving valuable network bandwidth. This contrasts strongly with the approach employed by active network management systems which constantly poll all reachable interfaces over the network.

- Once deployed, the agent based system is completely distributed, spreading the network traffic over the entire network canvas.

In the agent simulation developed, the use of mobile agents in network knowledge acquisition is investigated. A behavioral state model for mobile agents is developed and analyzed. In the simulation, which was developed in Java 1.2, agents are represented as objects which are associated with various nodes in a network, which itself is represented as a graph. Each agent has an associated execution time, and the agents are stored in a priority queue sorted in order of increasing execution time. The simulation is event driven and the simulation time is determined by the time associated with the agent at the head of the simulation queue.

The State Model and Population Control: Since agents in this simulation scenario act autonomously, it is important that a stable and robust population control mechanism be inherent in their behavioral mechanisms. An agent population explosion could result in many of the catastrophic effects observed in the case of worms, e-mail viruses, etc. On the other hand, for instance, a sluggish response to a rapidly changing environment will yield a system that is hopelessly out of date. In the proposed model, mobile agents can autonomously react to various scenarios and still converge to a solution without any of the adverse effects described above. It is also important that the agents eventually deposit the acquired information at some specified collection node.

The agent model and the associated parameters utilized in the simulation are described as follows. A four-state model was used. Two of the states derive from a case where new information is found at a node, whereby the agent decides either to clone or not (the decision is part probabilistic, part time related and part a function of the local node connectivity). The other two states occur when the agent does not discover any new information at a node. Figure 2.4 illustrates the transformation between the different agent states.

Parameters: The functions and constants used in the simulation are defined thus: k - a constant used in a decreasing probability function.
$P(x)$ - a boolean probability function which returns a true value at a frequency proportional to x where x lies between 0 and 1 (inclusive). c - a constant representing the probability of an agent cloning.
p_d - a number between 0 and 1 which decreases with time. It is used to progressively decrease the probability of cloning.
m_x - the minimum number of directly connected neighbors a node x must have, in order for an agent resident at that node to be able to clone.
mi - the maximum time an agent can roam idle without finding any new information.
$t1, t2$ - the lower and upper bounds on which an agent can reside at a particular node.

The algorithm below describes the decision making process when an agent arrives at a node x

1. the agent gets the information value stored at this node

2. define p_d to be a decreasing probability of cloning which is defined to be a function of *time* (simulation time) thus: $p_d = time^{(1-k)}$ where k ≥ 1.

3. if an agent finds new information at a node

   (a) if $P(c)$ and $P(p_d)$ are true and number of neighbors of $x \geq m_x$ let parent and child wait for some time at the current node and then go to a neighboring node, unvisited if possible **STATE 1**

   (b) else wait for some time and then continue roaming the network without cloning or merging **STATE 2**

4. else if an agent arrives at a node which it has visited before

   (a) if it has been idle for a time greater than mi return agent to the original node **STATE 3**

   (b) else increment the time the agent has been idle

5. if there are any other agents at the current node merge with them **STATE 4** then wait for some time and then go to another node

2.7 An Operational Mobile Agent System

During the course of this project, it was decided to investigate the practical operation of a few freely available mobile agent systems. Systems considered included Agent Tcl, Agent Perl and TACOMA. At the end of the trials, the author was of the conviction that none of these agent systems would be ideal for the sample applications investigated in this paper. All these systems were found to be complicated to install and set up. Agent Tcl was found to require the installation of the graphical X-Windows environment in order to operate even simple command line agents! Agent Perl was found to install on some machines, while not on others. Installation attempts with TACOMA proved unsuccessful. In addition, these systems are not platform independent, and run only on UNIX / Linux operating systems. Preliminary investigations indicated the need for a mobile agent system offering features such as:

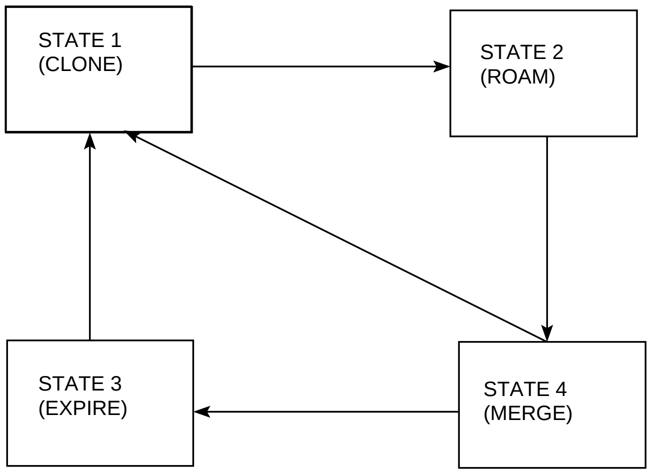

Figure 2.4: Relationship Between Agent States

- Cross-platform: Java and Perl can both operate on both UNIX and Windows platforms
- Strong mobility: The need for a strongly mobile agent system indicated the need for persistent object storage, commonly termed "Serialization". Java 1.2 supports serialization directly, and Perl does via the TOM (Transportable Object Model) library.
- Lightweight: Both these languages offer multi-threading support
- Straightforward and easily customizable: This requirement ruled out Perl which tends to become increasingly obfuscated with increasing code size, even in its object oriented form!

With these factors in mind, it was decided to attempt the development of a new agent system in Java 1.2. Figure 2.5 describes the system architecture and Figure 2.6 illustrates the object hierarchy.

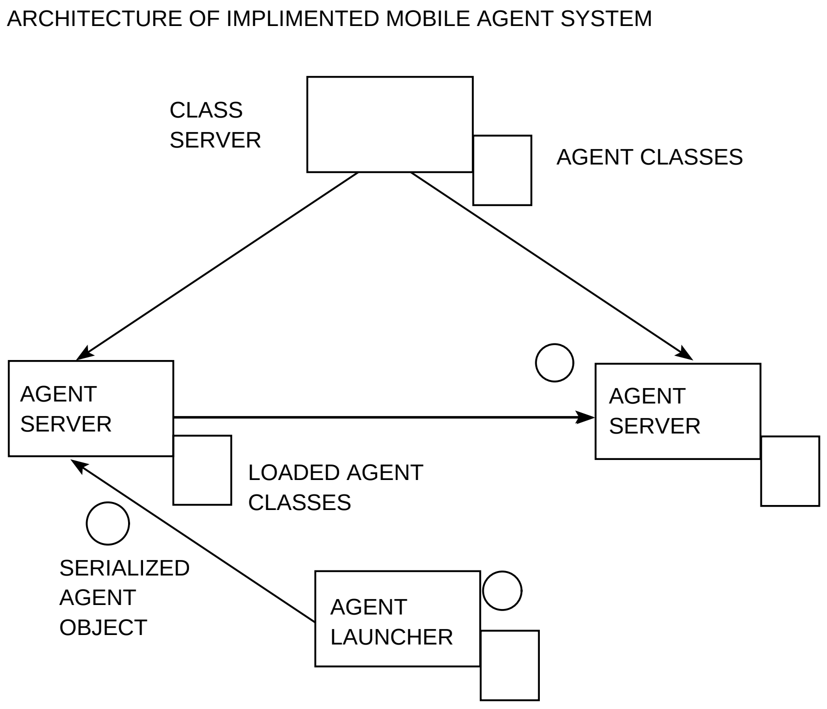

Figure 2.5: Functional Diagram of the Mobile Agent System

2.7.1 System Architecture

The system design in the form of an object diagram is described by Figure 2.5. It consists of the following components:

- Agent Server: This is a multi-threaded Java server which receives the agents, executes them and re-transmits them to the next host, if necessary. The agents are received and transmitted as serialized objects. In order to execute, however, they have to be de-serialized and their "execute()" method invoked.

- Class Server: Since the agent class need not be present at the server (only the agent object and not the class file is transported during serialization), the class has to be fetched over the network and loaded into memory. This is achieved by utilizing a Class Server, essentially a web server which can distribute the agent classes. Initial attempts at loading classes directly into memory using Java's URLClassLoader class failed because this class required instantiation using the "instanceOf()" function, as opposed to constructors. For this reason, when a non-loaded class is required, the bytes are fetched over the network and written to a file of the same name. The Java virtual machine can then load the class normally.

- Agent Object: These are transported as serialized objects and cast to the necessary object type when executed. An agent inheritance hierarchy is utilized for this reason. For example,in Figure 2.6, the mobile agent "FirstAgent" extends a class "MobileAgent" which exists on all servers with dummy function stubs. "FirstAgent" in turn extends a generic agent class "Agent" with no functionality other than being a parent class.

- Agent Launcher: This serves to create an agent instance and transmit it to a server.

The system was successfully implemented in its most essential form. In the next chapter, the performance of this system is discussed. In summary, an attempt has been made in this chapter to define the role of mobile agents in the management of modern networks. The advantages of mobile agents over conventional strategies have been discussed. Simulations have been designed to investigate the performance of mobile agent systems in two applications, namely Distance Vector Routing and Distributed Knowledge Acquisition. The architecture of the developed mobile agent system has been described. The following chapters present and analyze the results of the simulations developed.

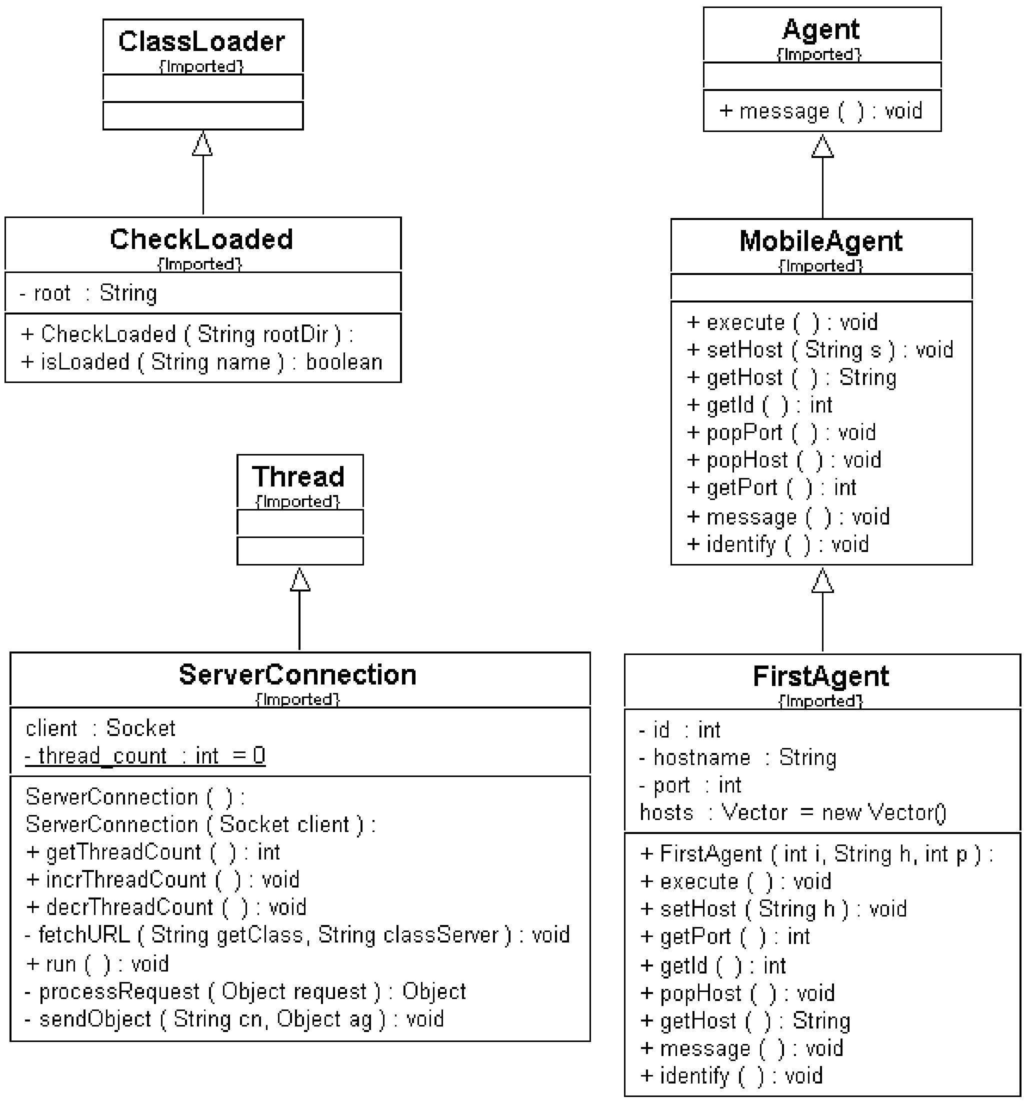

Figure 2.6: Object Diagram of the Mobile Agent System

CHAPTER 3

MOBILE AGENTS IN ROUTING

3.1 Agent-Based Distance Vector Routing

This section presents results from the simulation of agent-based distance vector routing as described in the previous chapter. Various experiments are performed with a view to compare and contrast the performance of agent based routing and conventional distance vector routing with a variation in agent population (which is constant for a particular run), graph size and graph topology. The experimental parameters most suitable to investigate the variation in routing performance have been identified to be:

- The Total Distance Vector Across All Nodes: This parameter represents the sum of the virtual path weights from each node to every other node. This parameter is initially set to infinity (represented by a vary large integer) and converges to a constant value as routing proceeds to convergence.

- The Number of Reachable Nodes for all nodes (denoted by total network discovery or number of virtual paths): As the routing proceeds, each node will be able to communicate with more non-adjacent nodes. Therefore, for an n node system, assuming that at the end of the simulation, each node can communicate with $n-1$ other nodes, there will be $n(n-1)$ virtual paths in the network. This parameter allows us to track the progress of the simulation. When the number of virtual paths is $n(n-1)$, the network is said to have achieved "transitive closure", a state where each node can communicate with any other node in the network.

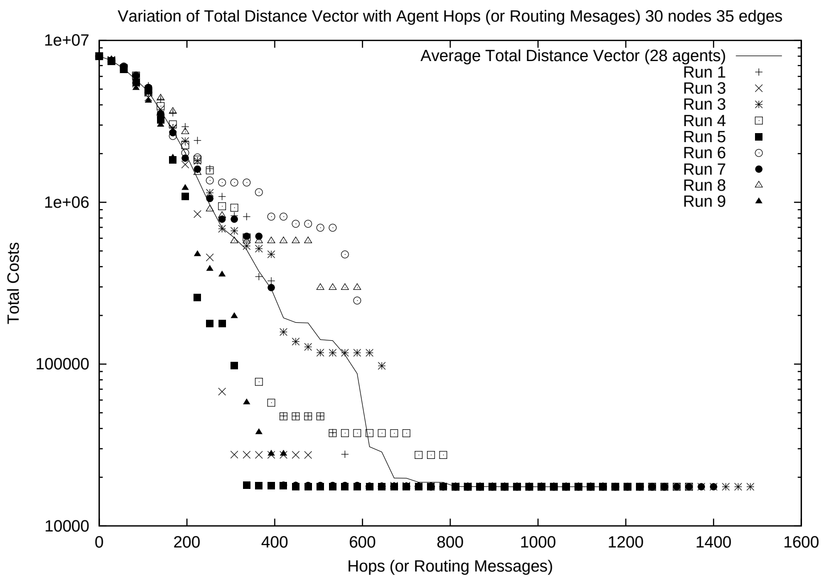

Figure 3.1: Typical run illustrating experimental variance

- Number of Hops (Routing Messages): It is important to choose wisely, a common basis with which to compare the performance of the agent based and routing simulations. The hop count has been identified to be satisfactory for this basis. A single agent making a single move from one node to another is considered equivalent, for the purposes of this simulation, to a single routing message sent from one node to another. The assumption here is that the byte size of an agent is comparable to that of a routing packet.

In most of the following experiments (except in the cases of ring / linear graphs in which the simulations did not converge), the curves presented are the average of 9 experimental runs. Figure 3.1 illustrates the variance in a typical experiment consisting of 9 runs. The arithmetic mean calculated from these runs is typically used in all the following experiments in this chapter.

3.1.1 Variation in Number of Agents

By increasing the number of random agents in the system, several effects could be expected to come into play. With an increase in agent population, the likelihood that the agents will be able to successfully route the network increase, since too few agents may be unable to monitor a large network. At the same time, if the number of agents is high, the agent routing could become inefficient due to random un-coordinated agent motion which wastes bandwidth.

This particular run was performed on a sparse random graph (30 nodes and 35 edges). From Figure 3.2, it can be observed that the conventional distance vector routing performs better than agent based routing (less message passing for convergence). This can be explained by noting that agent based routing (where there are a limited number of routing entities) performs better when agents have a wider choice of nodes to jump to, which implies a greater density of edges in the graph. In later simulations involving denser graphs, this trend is reversed, with mobile agents performing routing with less message passing overhead than conventional routing.

By increasing the number of mobile agents in the simulation, the routing performance as described above is not found to increase dramatically. However, on the average, the likelihood of convergence does improve. This is because when there are very few agents present in the system, they may not be able to cover the whole network efficiently.

Likewise, in Figure 3.3, it is seen that the traditional routing mechanism helps the network attain total reachability, indicated by $n(n-1)$ on the y axis of the graph(there are $n(n-1)$ total virtual paths in a network of n nodes) faster than with agents. Total reachability is an important parameter, because it indicates the ability of the all nodes on the network to communicate with each other. For the remainder of this document, the term $n(n-1)$ will be used to indicate the total reachability (number

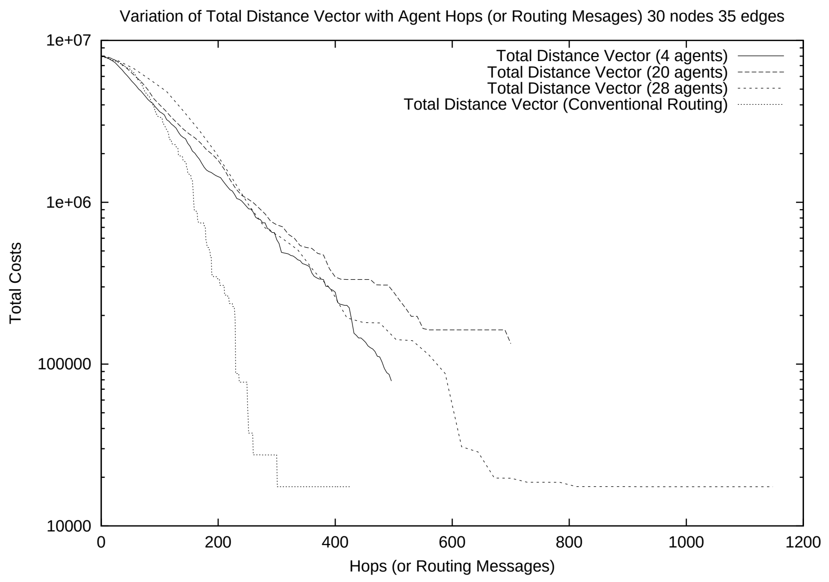

Figure 3.2: Variation in the total link cost across all nodes vs. number of message for varying agent population

of virtual parts). On a comparative scale, agent performance is not as robust, but tends to approach conventional distance vector routing performance as the number of agents in the system increases.

3.1.2 Variation in Graph Size and Density

Another set of experiments were performed on random graphs of progressively increasing density (*edges/nodes*), keeping the number of agents constant (10). All the chosen graphs are denser than in the previous case. In these simulations, the agents based simulations are found to be superior to normal distance vector routing, in the sense that agent-based routing converges with less message passing overhead (Figure 3.4). The other interesting observation noticed in Figure 3.5 is that agent-based routing achieves total connectivity with less message passing overhead than conventional routing.

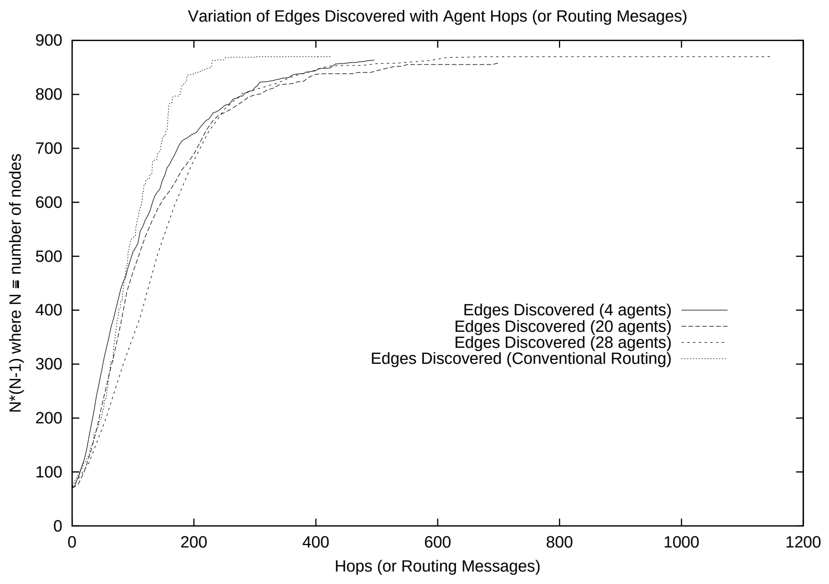

Figure 3.3: Variation in the total network discovery across all nodes vs. number of message for varying agent population

The results of Figures 3.4 and 3.5 can be explained on the basis of these factors:

- As the graph density (the ratio of edges to vertices) increases, so does the average cardinality of the nodes (number of edges connected to that node)
- As the cardinality of the nodes increases, the agents have more choices of nodes to jump to at each iteration, hence increasing the chance that they can discover new nodes (or paths)
- each agent generates only one message per hop. Therefore, in a system with a agents, the traffic generated is proportional to a, network size n remaining constant, and $a \leq n$ in most cases. However, for conventional distance vector routing, each node sends out information to each of its neighbors, thereby generating more traffic.

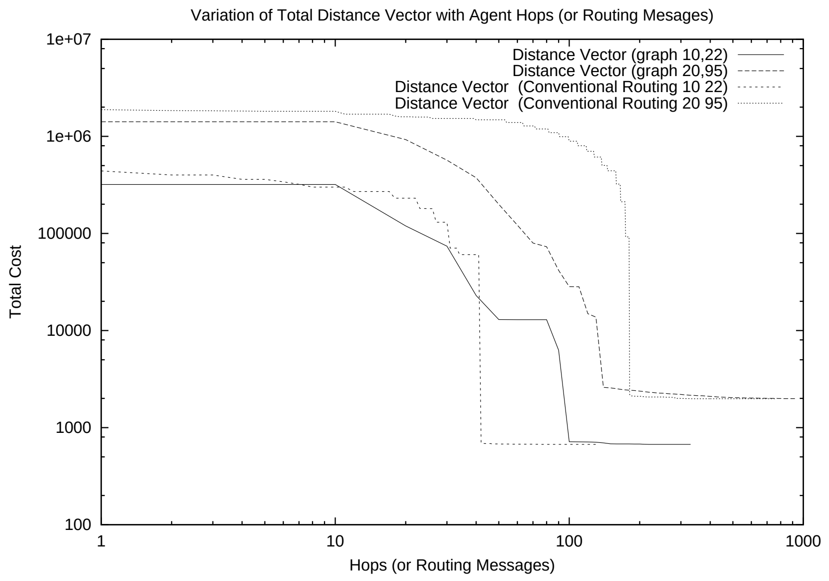

Figure 3.4: Variation in the total link cost across all nodes vs. number of message for varying graph size

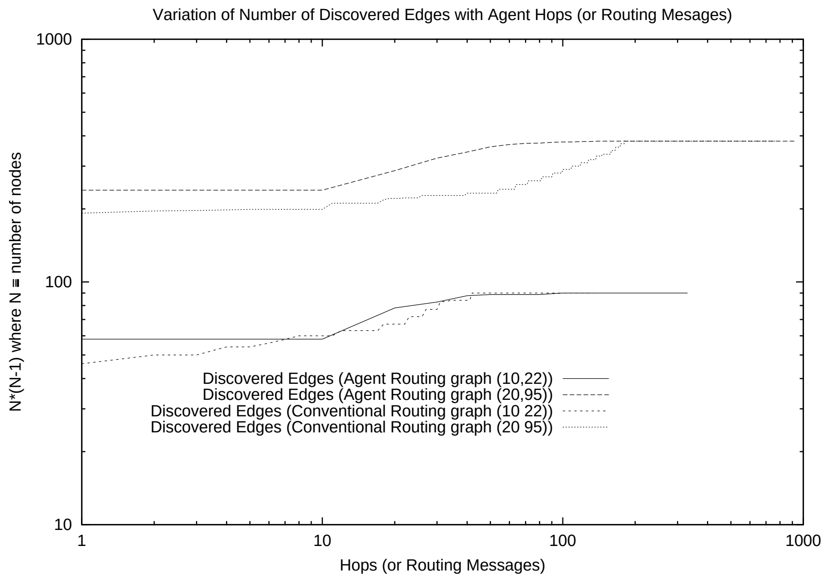

Figure 3.5: Variation in the total network discovery across all nodes vs. number of message for varying graph size

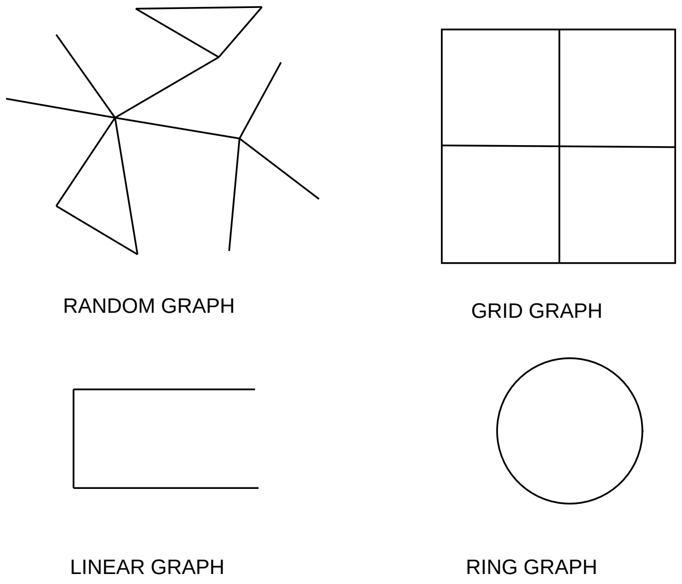

Figure 3.6: Representative Illustration of Various Graph Topologies

3.1.3 Variation in Graph Topology

In this section, the performance of our routing algorithms for various graph topologies are examined. The topologies investigated are random graphs , square grids, linear graphs and ring graphs (Figures 3.6).

As in the previous section, experiments have been performed to evaluate the convergence behavior of the routing algorithms, but for different topologies. Table 3.1 illustrates the graphs chosen for this series of experiments.

The number of edges in the grid structure is fixed for a particular number of nodes. Here, an edge is assumed to be bi-directional in nature. The determination of the number of edges for a fixed number of nodes in a square grid is derived thus:

Let number of nodes in the grid graph $= N$.

Let number of nodes in the side of this grid be $= n$.
Therefore, number of nodes in the graph $= n^2 = N$.
Consider the grid to consist of a number of orthogonal horizontal and vertical lines.
Each of the n horizontal lines covers n nodes and hence $n-1$ edges
Therefore, there exist $n(n-1)$ horizontal edges.
Since the number of vertical edges is equal to the number of the horizontal edges, number of (bidirectional) edges $= n(n-1) + n(n-1) = 2n(n-1) = 2(n^2 - n) = 2(N - N^{0.5})$.

In linear graphs of n nodes, there exist $n-1$ edges, whereas for the same sized ring (circular) graph, there would exist a total of n edges.

The graphs in Figures 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, 3.13, 3.14 illustrate the difference in routing performance for both the conventional and agent routing algorithms for various topologies. On comparing trends across various topologies, it is seen that the random and grid graphs show similar trends though the random graphs exhibit routing with a lower number of messages, probably due to the higher connectivity of these graph types. For the chosen random and grid topologies, agent routing is noticed to operate with a lower message passing overhead as compared to conventional routing.

Table 3.1: Description of Graphs used in the Routing Simulation

Graph Type	Nodes	Edges	Agents
Random	40	80	10
Random	100	500	25
Grid	36	60	9
Grid	100	180	25
Linear	100	99	25
Ring	100	100	25

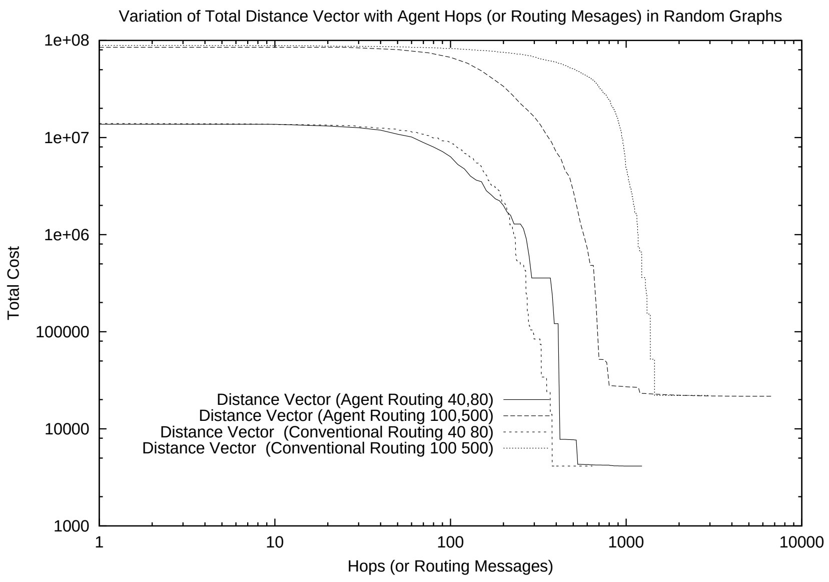

Figure 3.7: Variation in the total link cost across all nodes vs. number of messages for varying graph size (random graphs)

In contrast, linear and ring graphs are always found to require more routing messages as compared to the random and grid graphs. Linear and ring graphs generally have a greater minimum average distance (hops) between different neighbors. Comparing the two, convergence in the ring is better than in the linear graphs, due to the smaller average distance between pairs of neighbors. Agent routing is much worse due to the random nature of the agent motion, resulting in inefficient distribution of routing information between nodes. It is also observed that due to the inefficient routing, agent routing is inferior to conventional distance vector routing for linear and ring graphs.

3.2 Analysis of Agent Based Routing

In agent based routing, there are a fixed number of mobile entities roaming the network (randomly) performing routing calculations. The number of agents is typically

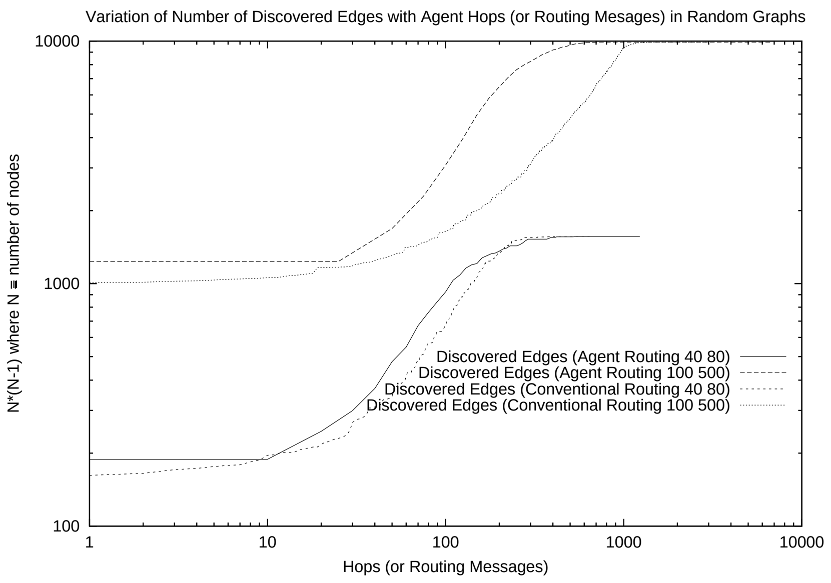

Figure 3.8: Variation in the total network discovery across all nodes vs. number of messages for varying graph size (random graphs)

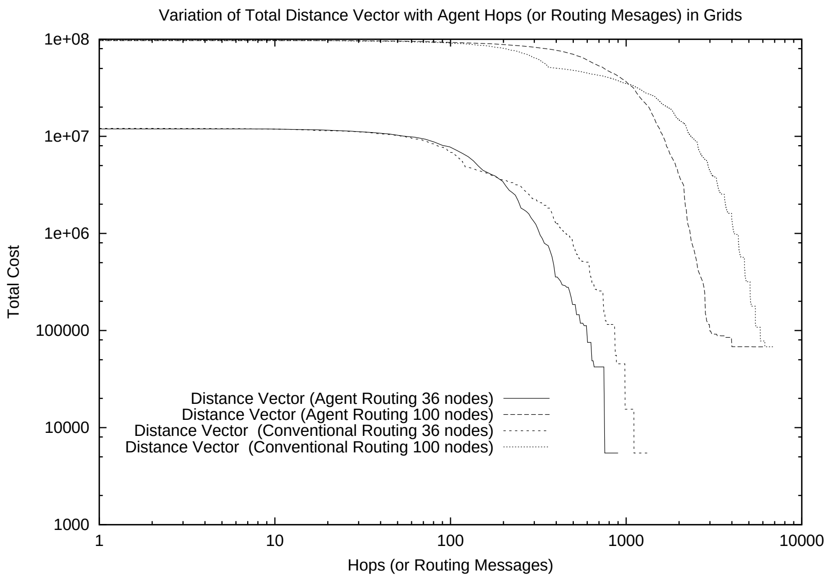

Figure 3.9: Variation in the total link cost across all nodes vs. number of messages for varying graph size (grid graphs)

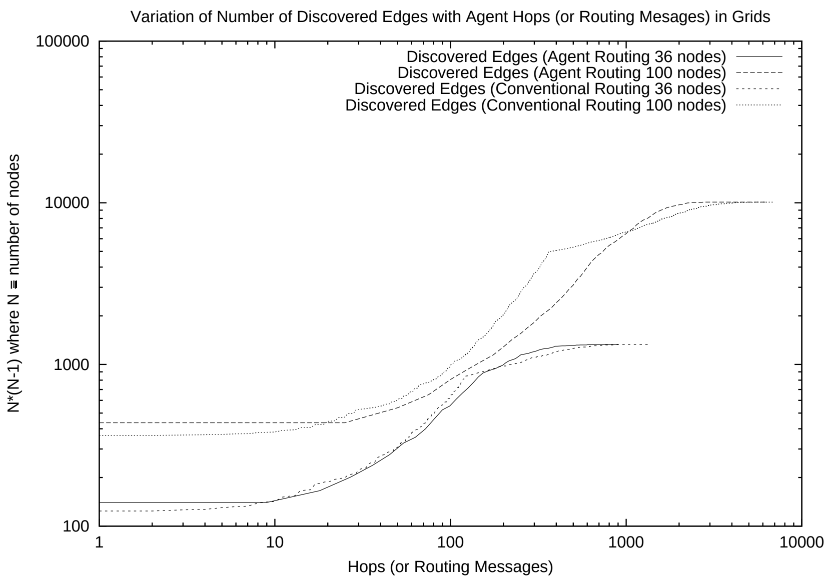

Figure 3.10: Variation in the total network discovery across all nodes vs. number of messages for varying graph size (grid graphs)

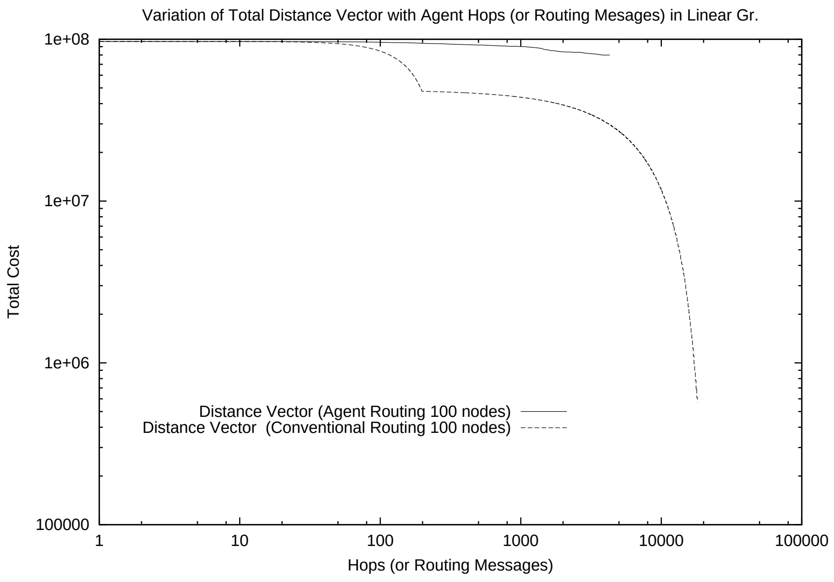

Figure 3.11: Variation in the total link cost across all nodes vs. number of messages for varying graph size (linear graphs)

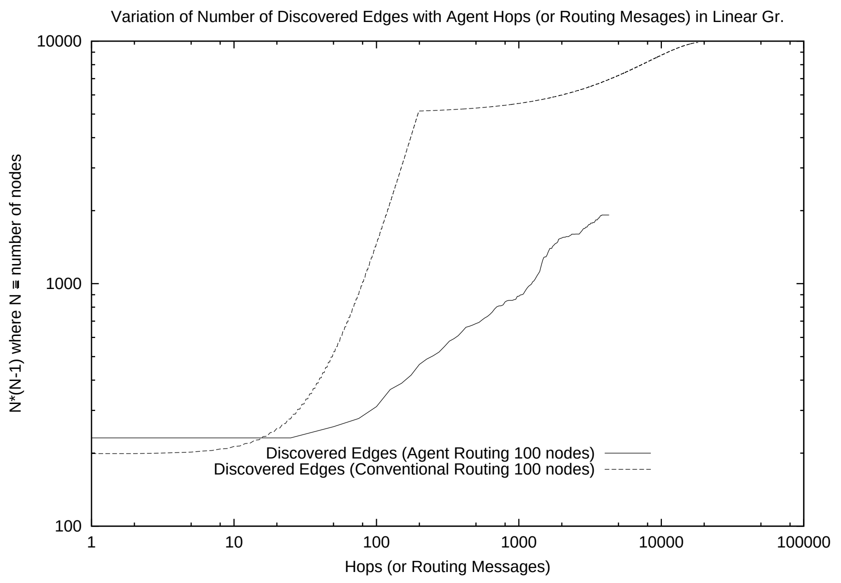

Figure 3.12: Variation in the total network discovery across all nodes vs. number of messages for varying graph size (linear graphs)

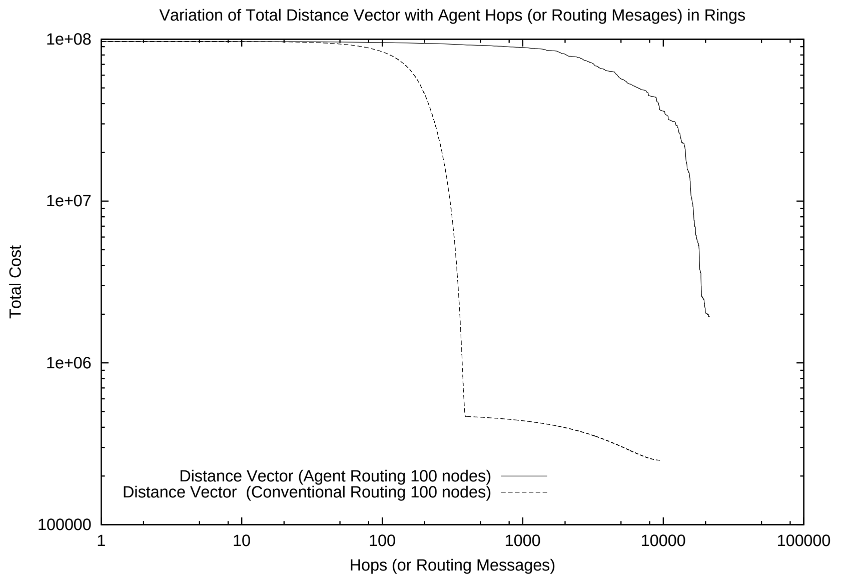

Figure 3.13: Variation in the total link cost across all nodes vs. number of messages for varying graph size (ring graphs)

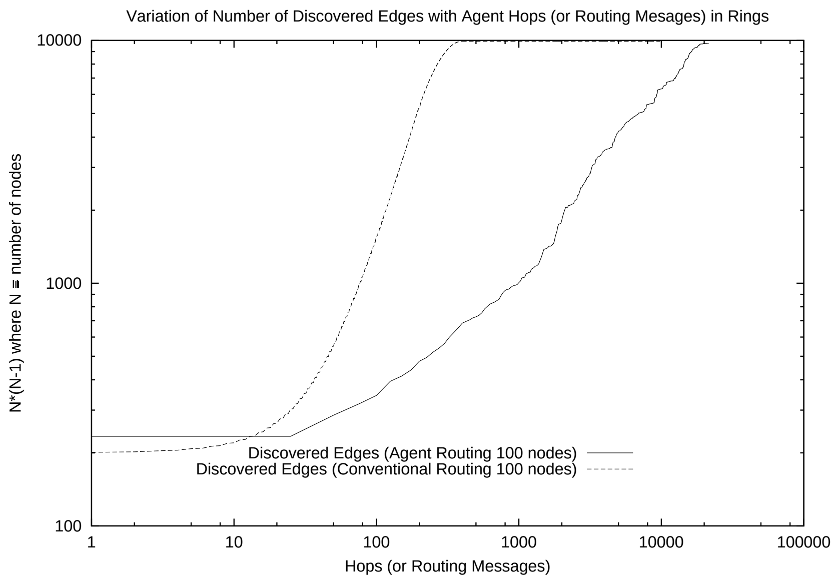

Figure 3.14: Variation in the total network discovery across all nodes vs. number of messages for varying graph size (ring graphs)

less than the number of nodes. Hence, the traffic generated (messages or agent hops) per time can be expected to be less than for an equivalent conventional routing system. Therefore, an agent based routing system could be expected to require a longer time to achieve convergence, as compared to a conventional distance vector routing system. This has been verified by examination of simulation results obtained from a time-based routing simulator developed by Uday Bhaskar Akella at the Network Research Laboratory, UNT. For instance, Figure 3.15 shows that agents walking randomly through the network (and routing) do not perform as well as a conventional routing system. The performance of the agent system can be improved by introducing "Structured Agent Movement" (SAM) capabilities to the agent whereby the agent makes an intelligent choice of which node to perform a routing calculation on next. SAM agents show a definite improvement over random agent movement and approaches the performance of conventional routing. Likewise, from Figure 3.16, it is

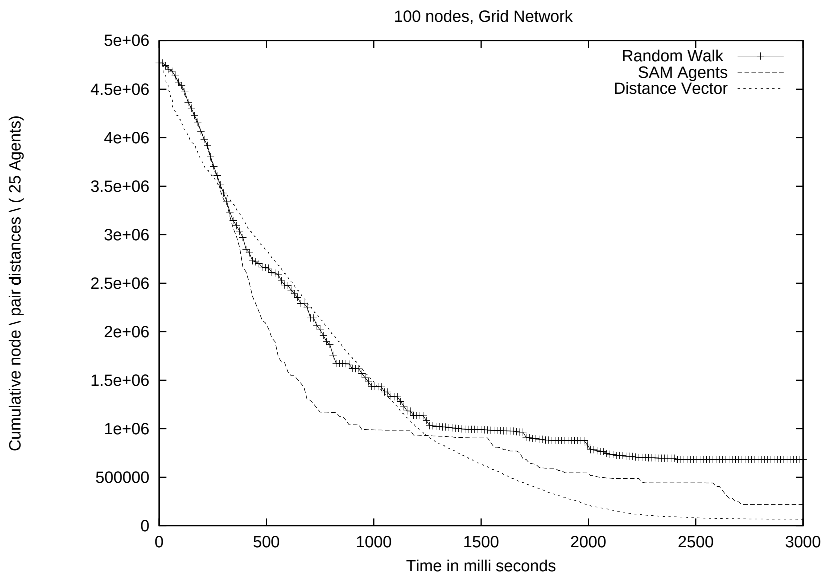

Figure 3.15: Variation in Convergence for a Time Based Routing Simulator

seen that network reachability is obtained faster for conventional routing as opposed to the agent based variations.

Summary: Agent based variants of the distance vector routing protocol require a longer time to converge to a solution than the conventional distance vector method. However, they generate a smaller amount of (routing) message traffic if the network graphs are not too sparse, since there is less unnecessary communication between the network nodes. At this point, it becomes relevant to examine the issue of which is more important: a faster converging network routing algorithm or a low bandwidth-consuming solution. This is a difficult question to provide a direct answer to. Several factors need to be taken into consideration, such as the network size, connectivity, available bandwidth and stability. For example, in a mission critical network with a large amount of available bandwidth, conventional routing would be more applicable. On the other hand, in a low bandwidth network which is plagued by the issue of a

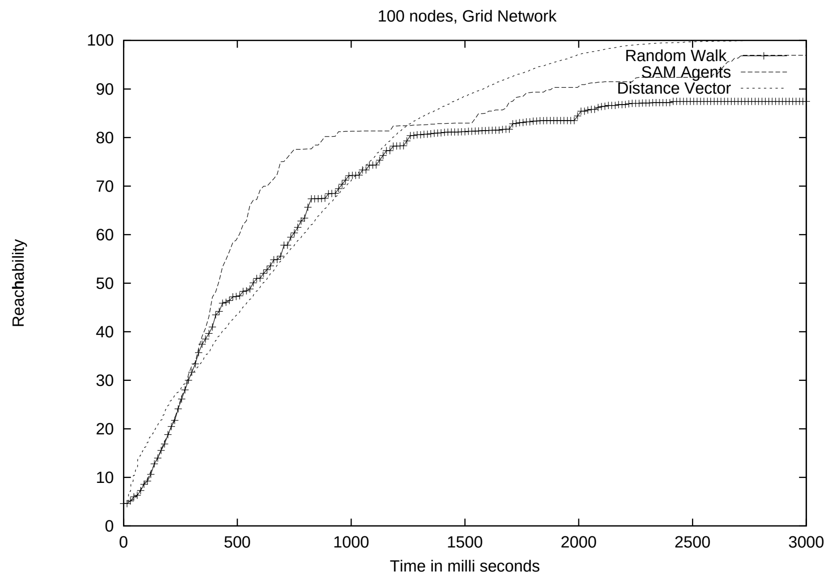

Figure 3.16: Variation in Reachability for a Time Based Routing Simulator

large amount of routing traffic, an agent based system would probably serve better. Another possibility is the development of hybrid systems using conventional routing protocols at startup and at periodically infrequent intervals and agent routing at all other times. Simulation of such scenarios can provide invaluable information about the expected behavior of such systems and can assist the developer in making informed decisions about which systems to develop or implement.

CHAPTER 4

MOBILE AGENTS IN KNOWLEDGE ACQUISITION

4.1 Agent-Based Knowledge Acquisition

This section attempts to investigate the performance of mobile agents in a knowledge acquisition problem, as described in the previous chapter. The agent population changes dynamically, both as a function of user defined parameters and as a function of the state of the simulation, as quantified by certain parameters, namely:

- $init\_agents$ - number of agent at the start of the simulation
- $hnode$ - node at which the agents are initially spawned
- $t1$ - minimum agent residence time at a node
- $t2$ - maximum agent residence time at a node
- max_idle - the maximum time an agent can roam the network without discovering any new information
- $clone\_prob$ - the probability of an agent cloning, all other factors being in favor of this
- $min\_neighbours$ - the minimum number of neighbors a node must have, before an agent at the node can clone
- $change\_node\_info\_prob$ - the probability of the information at a node changing
- $scale\_down\_fertility$ - the fertility decrease factor

Using the general purpose mobile agent simulator developed for the purpose, experiments have been run and analyzed to determine variation of agent behavior, agent population and knowledge acquisition rate with a variation in the parameters described in Table 4.1. Also, in Table 4.2, the main agent-capable functions which were used in the simulation are listed.

Table 4.1: Description of Default Simulation Parameters for Knowledge Acquisition Simulation

Parameter	Value
initial agents	1
min. residence time	1
max. residence time	5
max. roam time	10
cloning probability	1.0
min. neighbours for cloning to occur	1
probability of node info changing	0.1
fertility decrease factor	1.0

Table 4.2: Description of Simulated Agent Functions

Function	Description
boolean clone_decision	Decides whether to clone or not
int findUnvisitedNeighbour	Tries to find an unvisited neighbouring node
void incrDiscoverTime	Increments the idle time without discovering anything new
boolean merge_decision	Decides whether to merge or not
void resetDiscoverTime	Set the time at which something new is discovered

4.1.1 General Description of Convergence Curve

Figure 4.1 shows all the types of information that can be directly obtained from a run. It should be stressed that the information shown for this run does not indicate or imply optimum performance, but merely serves to demonstrate how the behavior of the agents can be inferred from simulation data. In this case, a 36 node random graph is considered.

The simulation has the capability to display a large amount of data concerned with the current and previous states of the mobile agents. The significance of this information is described thus:

- Merge Agents - This represents the count of agents that are willing to merge with other agents (while sharing information). Agents normally transform to this state when they cease to find new information by themselves and attempt to coalesce with other mobile entities in the same state in order to gain access to their information base. The number of merge agents tends to increase when cloning does not take place over an extended period of time.

- Queue Agents - This represents the number of agents which are actively propagating or roaming the network in search of information. Queue agents are normally found to dominate the simulation at the start of the simulation.

- Total Agents - This is the total number of agents in the system and can be used to gauge the convergence of the simulation.

- Total Discovery - This measures the total information discovery by any agent. This means that if every node in the network has been visited by some agent, this parameter will indicate that the entire network has been "discovered" . This parameter serves as a measure of how well the agents are propagating over the network.

- Total Learned - As agents (eventually) roam the network without finding any new information, they are programmed to finally return to the node which spawned them. They then deposit the information they learned. The "Total

Learned" parameter represents the total accessible knowledge gathered by returning agents. This parameter normally lags behind the "Total Discovered" parameter by some amount of time.

- Clone, Roam, Expire and Merge Counts - These curves are useful for debugging purposes and to see which agent state is dominating at any point of time. They represent the cumulative counts of times that an agent has attained the respective states.

- Simulation Time - All the previously mentioned parameters are plotted with respect to simulation ticks, measured on an arbitrary time scale.

As described in the previous chapter, the agent can exist in any one of four atomic states (cloning, roaming, merging or expiring(returning to parent node)) and exists in the simulation either at a node willing to share information (the "merge pot") or in a simulation queue, dormantly waiting to execute at another node. Figure 4.1 shows the variation of agents population in the queue, the merge pots, total agent population (sum of these two), and the cumulative number of times agents have been in the 4 discrete simulation states. The graph also shows the variation in the total amount of information learnt by all the agents as well as the amount retrieved by spawning node (when agents decide to expire, they return to this node).

For instance, since the probability of agents cloning (if possible) is unity, agents never attain state 2. Eventually, the agents enter the expiration state. Incidentally, this graph illustrates the "population explosion" phenomenon, where agents spawn child agents very fast without gaining much knowledge. This process continues through several generations of agents, leading to slow convergence. Due to merging and expiration state, which occur after a certain period of time, the agent population decreases but never converges to unity, due to changing node information

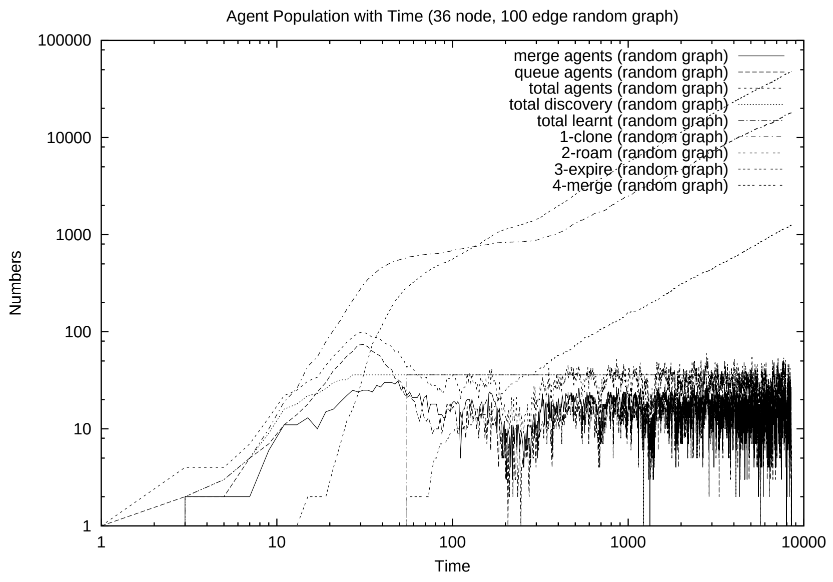

Figure 4.1: A Representative Simulation Run For Knowledge Acquisition in a 36 node graph

(this will be explained later). The cumulative number of cloning steps is seen to follow an exponential trend and is matched by the number of merging and expirations. It is also noticed that all the nodes are visited by some agent or the other at some time sooner than the time when the home node obtains total system information from expiring agents.

The convergence can be seen faster for a smaller graph (9 nodes) (Figure 4.2). Notice that the agent population decreases on the average at large values of simulation time. Also notice from flattening of the cloning state curve, that cloning eventually stops, indicating an equilibrium state achievable with the current number of agents.

To simulate real world conditions, the information at a node has been programmed to change with a probability of 0.1 (by default) every time an agent arrives at it. This explains the heavy variation of agent population (within limits) even late in the simulation after complete system state has been determined.

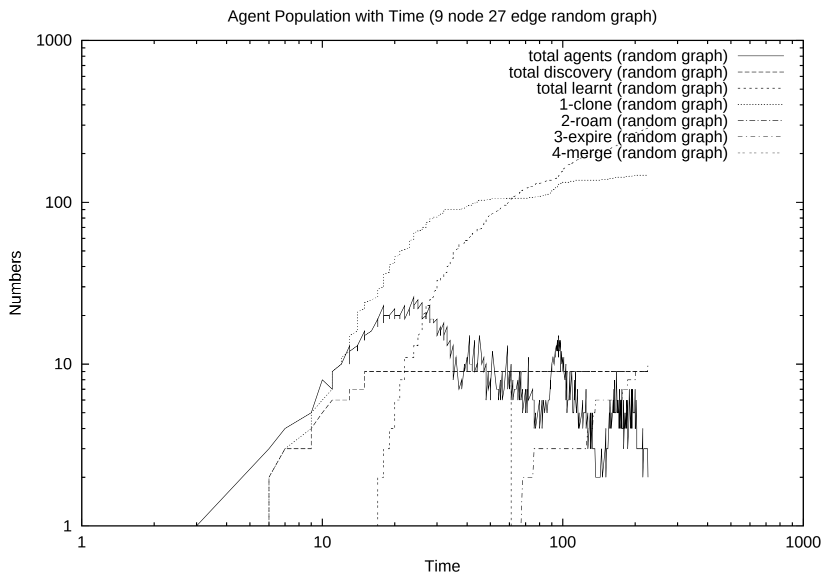

Figure 4.2: A Representative Simulation Run For Knowledge Acquisition in a 9 node graph

4.1.2 Variation in Graph Type

In this experiment, the variation in knowledge acquisition at the home node and total agent population is studied as a function of graph topology. Four different topologies of increasing density have been considered (linear graphs, ring graphs, square grids and totally connected graphs) and the results presented are typical of the observed trends. It is noticed in Figure 4.3 that for sparse graphs, the population explosion of mobile agents is not as severe as in denser graphs, the reason being that there are fewer paths between neighbours. Therefore, the agents tend to travel to nodes they have visited before and not spawn.

Figure 4.4 shows the knowledge acquisition profile for the same experiment. The term "knowledge acquisition" mentioned here and in future describes (unless otherwise mentioned) the knowledge acquired by various agents and deposited at a node on expiration. For this reason, the profiles have a "step-like" appearance since knowledge

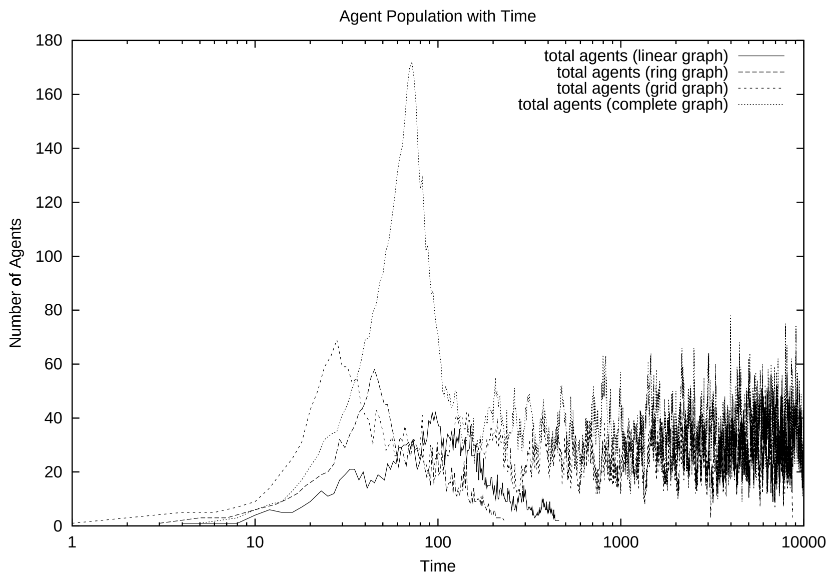

Figure 4.3: Variation in Agent Population with Topology for 36 node graphs

is mostly gained at the home node when agents enter the expire state and deposit information possibly gained from several nodes. The variation of knowledge acquisition over different topologies follows a slightly complicated trend. For topologies such as a grid, knowledge acquisition is relatively faster. However, for very dense (complete graph) or very sparse graphs (ring, linear), knowledge acquisition is slower. This can be explained on the basis of two factors:

1. In very sparse graphs, agents tend to travel along paths they have visited previously, and hence tend to visit nodes they have information about. Hence, knowledge acquisition is slow.

2. In very dense graphs, agents tend to clone too much and hence reach the expire state much later than would be expected.

For the purpose of constructing meaningful experiments in the following paragraphs, it becomes necessary to identify a representative network topology which can

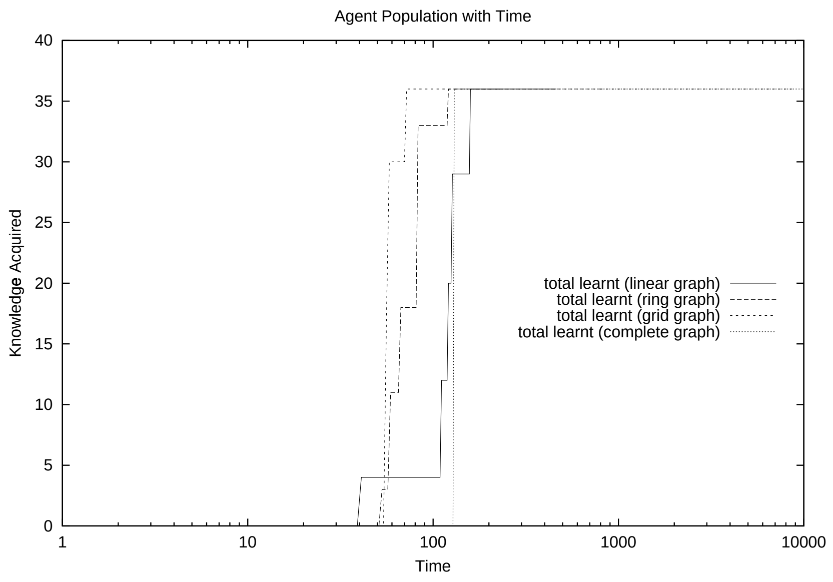

Figure 4.4: Variation in Knowledge Acquisition with Topology for 36 node graphs

easily be replicated and described. The square grid appears to be such a topology. Before it can be accepted, the topology should be tested to see if agent performance varies significantly with a variation in the topology, graph density being kept constant. In order to achieve this, the square grid generator is modified to allow distortion of the network. In other words, each distortion involves the removal of a random edge and the addition of another edge, with functionality built in to ensure the preservation of graph connectivity. Over a large number of distortions, the graph will no longer resemble a square grid. The distortion of the graph can be expected to create some areas of higher connectivity. These could be expected to induce population explosion. Figure 4.5 describes the variation of agent population profiles with network graph distortion in a 81 node graph. As expected, with a distortion in network geometry, the population explosion results in a slightly higher peak agent population. Figure 4.6 describes the knowledge acquisition profile and indicates that the discovery profiles are similar, though the slightly greater population explosion resulting from distorted

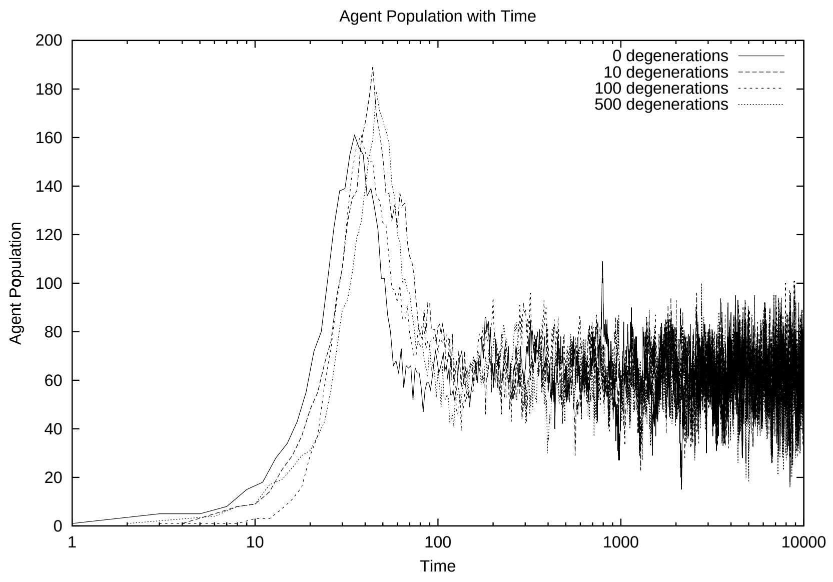

Figure 4.5: Variation in Agent Population with Distortion for 81 node graphs

geometries has actually improved knowledge acquisition time slightly.

4.1.3 Variation in graph size

In this experiment, the behavior of the agent system in square grid graphs of various sizes is investigated. An increase in network graph size would be expected to trigger a larger number of agent spawning. This is corroborated by Figure 4.7. Also, it is observed that with an increase in graph size, the maximum agent population also increases, as a direct consequence of the increased cloning. Interestingly though, from Figure 4.8, the time taken to acquire the knowledge is almost the same, regardless of graph size. This indicates that the agent based mechanism has potential as a scalable mechanism for network knowledge acquisition.

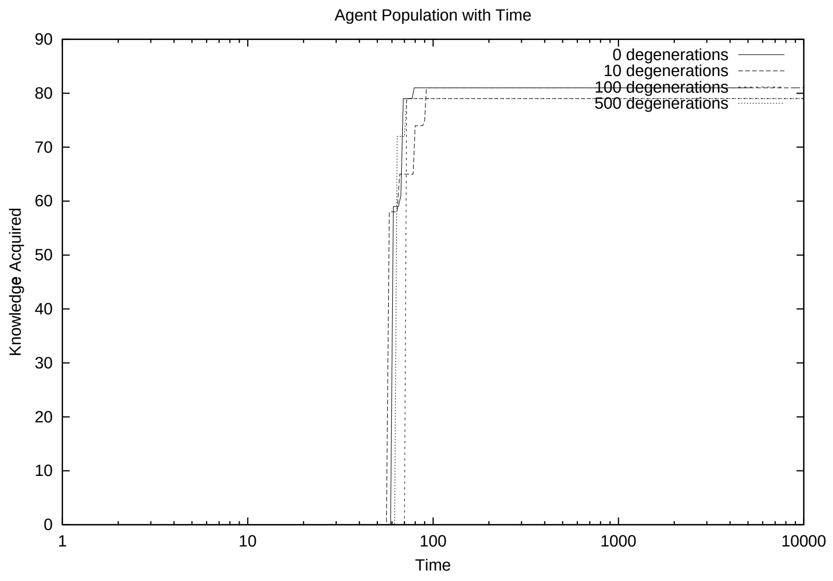

Figure 4.6: Variation in Knowledge Acquisition with Distortion for 81 node graphs

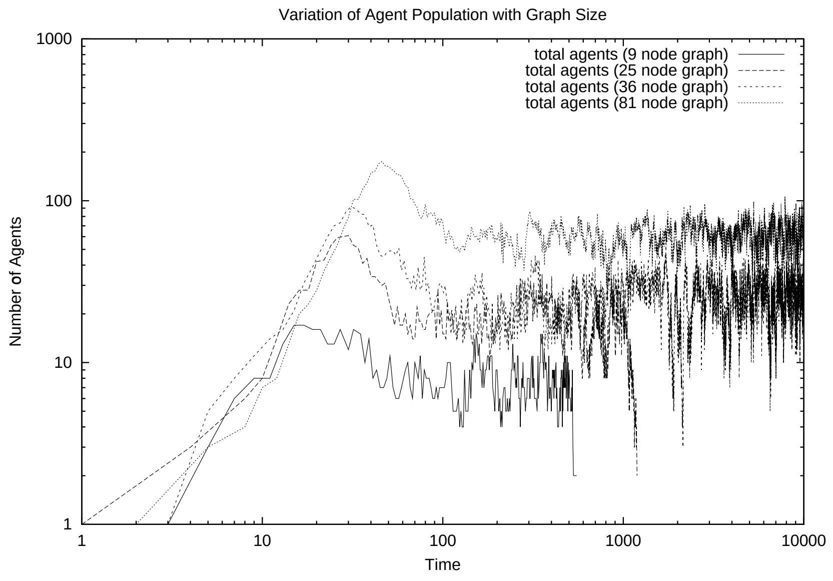

Figure 4.7: Variation in Agent Population with Graph Size

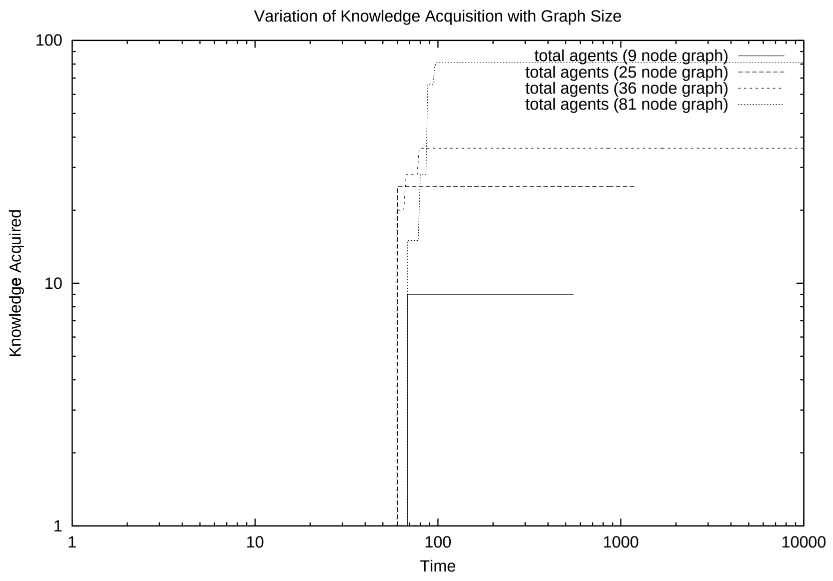

Figure 4.8: Variation in Acquired Knowledge with Graph Size

4.1.4 Variation in agent fertility

The performance of knowledge acquisition is investigated as a function of the decrease in fertility of an agent with age. One of the issues observed in previous simulations was the explosion of the agent population. The resulting multitude of agents further spawn every time they learn more information. Eventually, the tendency to spawn is curbed as the agents obtain more information about their environment. However, since spawning is most useful at a time early in the simulation life cycle (and is detrimental later on), a logical approach to the issue would involve decreasing the agent "fertility" (or tendency to spawn) as a function of its age. This is achieved in this simulation by using a decay factor f, which controls the rate at which the agents become infertile. The decay factor f is used in a a decaying mathematical function $d(f, t)$ which approaches 0 at large times. t here represents the simulated time. The relation is

$d(f, t) = t^{1-f}$

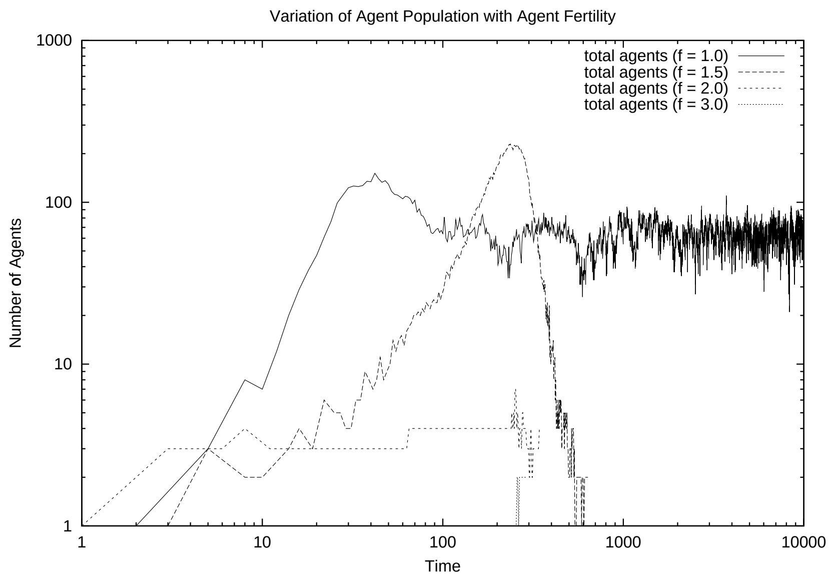

Figure 4.9: Variation in Agent Population with Fertility (81 node grid)

If an agent wishes to clone at this point, the decaying probability of cloning P would be $= d(f, t)$

The results of a reducing agent fertility is shown in Figures 4.9 and 4.10. The sample graph has 81 nodes. It is observed that as the fertility is decreased:

- The agent population has an opportunity to actually converge back to a single agent. This is because no agents are spawned beyond a certain point in time. This is in contrast to the case where the agents are always fertile ($f = 1$) and agents are spawned throughout the simulation.
- In general, as the fertility decay factor is increased, the maximum agent population also decreases, due to an earlier onset of the decrease in agent fertility.
- If the fertility decay factor is high ($f = 3$), not enough agents are spawned and total knowledge acquisition is not easily achieved.

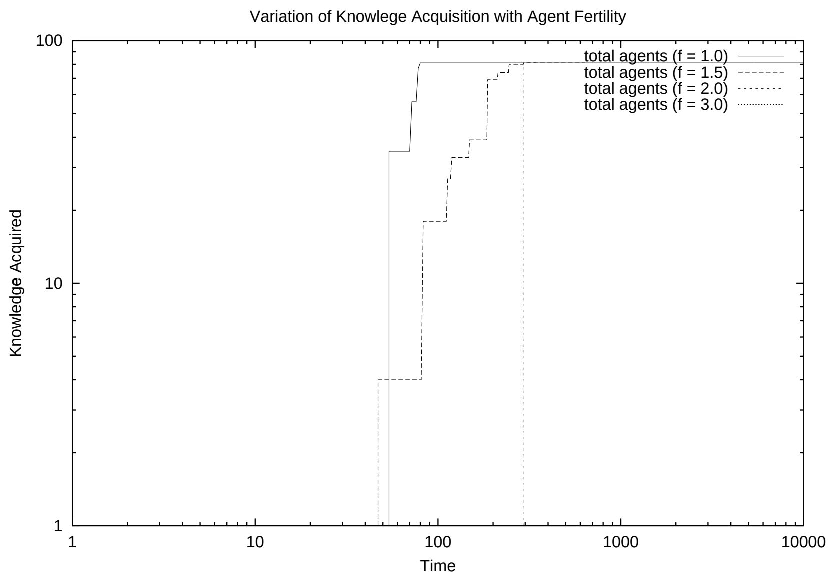

Figure 4.10: Variation in Acquired Knowledge with Fertility (81 node grid)

4.1.5 Variation in number of initial agents

Figures 4.11 and 4.12 represent the results obtained by changing the number of agents initially present in the system (a grid graph of size 81) . There is no decrease in agent fertility in this simulation. It is immediately apparent from 4.11 that the number of agents initially present has little impact on the convergence profile of the agent population profile. The maximum number of agents is virtually the same, irrespective of the number of agents initially present. Interestingly, the presence of a larger number of agents at the beginning of a simulation does not lead to an exponential increase in agent population (population explosion) as mentioned in previous sections (Variation in graph type) . Therefore, the agent performance characteristic is strongly influenced by the graph geometry. Figure 4.12 corroborates this fact by showing that total system knowledge is acquired around the same time, regardless of initial agent population. These factors lend further weight to the scalability of agent based knowledge acquisition.

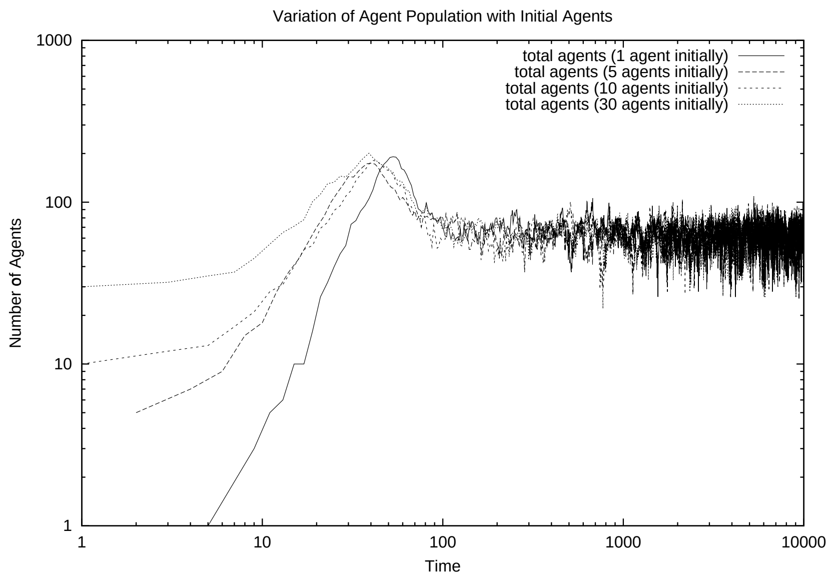

Figure 4.11: Variation in Agent Population with Initial Agents (81 node grid)

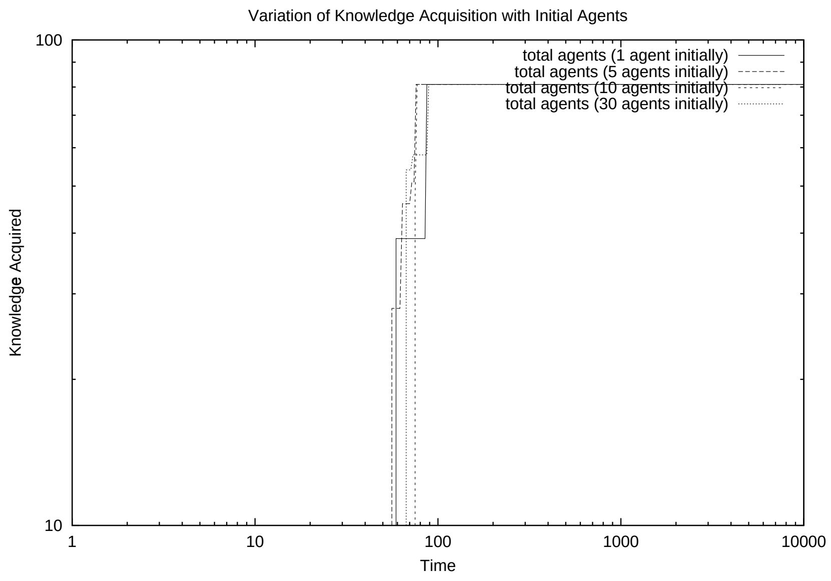

Figure 4.12: Variation in Acquired Knowledge with Initial Agents (81 node grid)

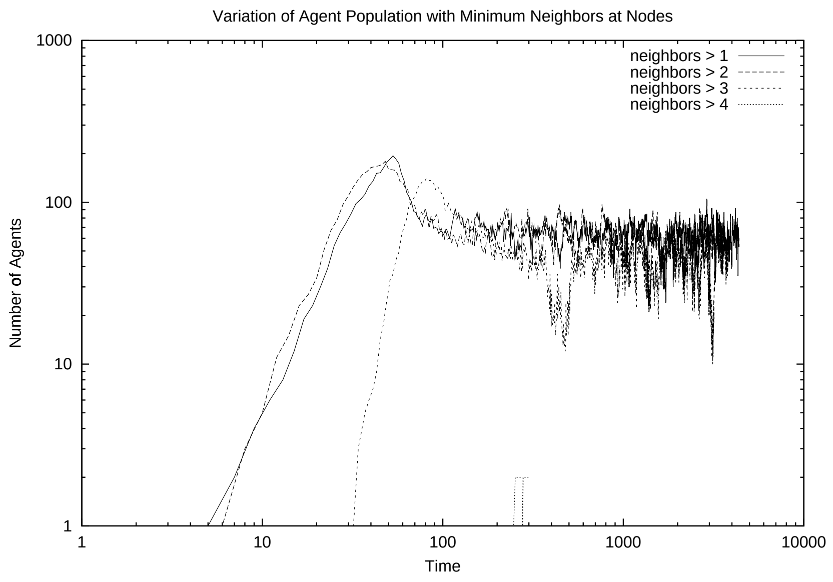

Figure 4.13: Variation in Agent Population with Minimum Neighbours for Cloning (81 node grid)

4.1.6 Variation in minimum node cardinality required for agent cloning

The simulation has the capability to limit the cloning of the agents (when otherwise possible) to situations where the number of directly connected neighbours of its host node (let this be represented as n) is greater than some specifiable parameter. Since the network chosen was a grid, each node has either 2 (corners), 3 (sides) or 4 (interior nodes) directly connected neighbors. As the specified value of n (to the agent) increases, the number of clonings decreases and this slows down the knowledge acquisition (Figures 4.13 and 4.14). If no node has a cardinality which satisfies the cloning condition, knowledge acquisition becomes difficult, if not practically impossible, for large graphs.

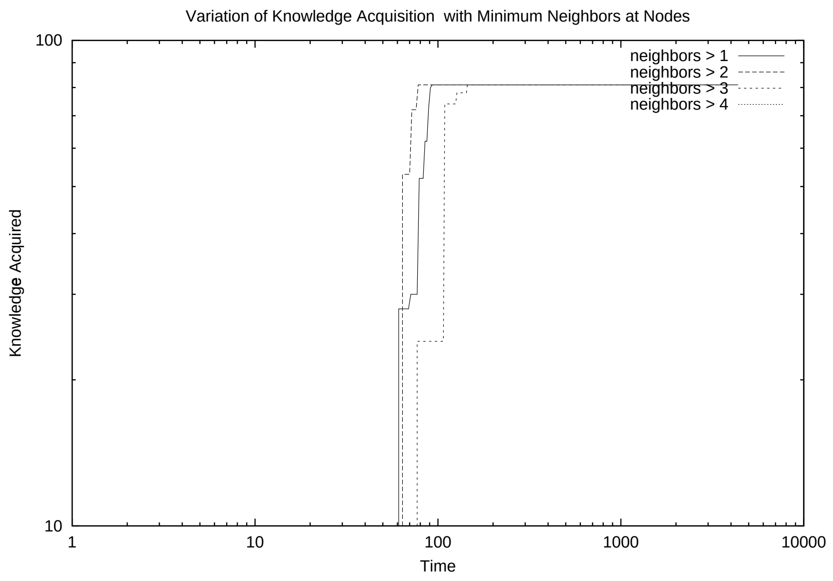

Figure 4.14: Variation in Acquired Knowledge with Minimum Neighbours for Cloning (81 node grid)

4.1.7 Variation in cloning probability

This section investigates the performance of the agents when their decision to clone, other factors being favorable, is also influenced by a probability function. The motivation for the variation of cloning probabilities is to reduce the effects of agent population explosion induced by very dense graphs or rapidly changing node information.

Figures 4.15 and 4.16 illustrates the effect of varying this clonability. It is noticed that as the cloning probability is decreased,

1. The acquisition of knowledge is slower
2. For low cloning probabilities, the agent population converges to a single agent, in spite of network information changing (probability = 0.1). This indicates that the use of low cloning probabilities results in a converging agent system,

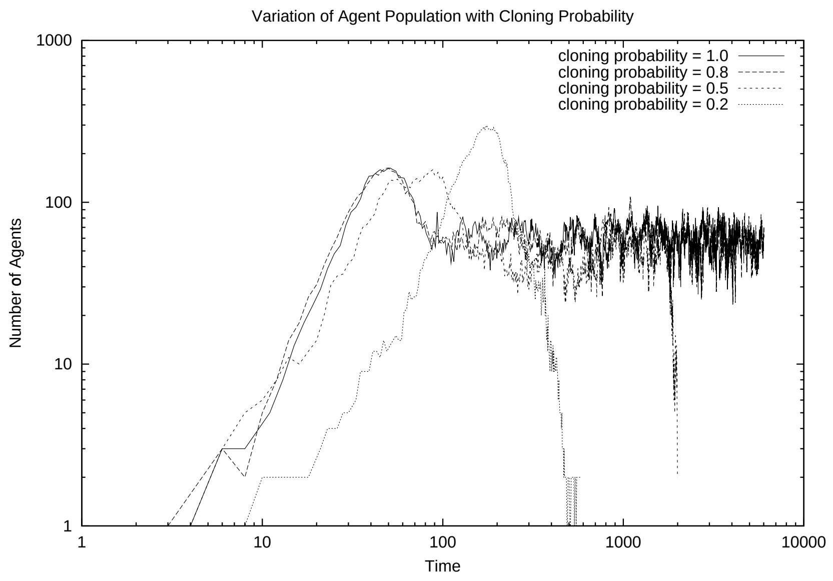

Figure 4.15: Variation in Agent Population with Cloning Probability (81 node grid)

regardless of whether the network information is changing or not.

4.1.8 Variation in dynamically changing information at a node

The object of this experiment was to determine the effect of the network instability (change of information at the nodes) on the agents. For the experiments performed up to this point, the probability of change has been fixed at 0.1 . Here, this value is varied in both directions and the effects analyzed. The experiments have been conducted on a grid graph of size 81. As shown by Figure 4.17, when the network is static (constant information at the nodes), the agent population converges in a predictable fashion to a single agent. When the probability of change is moderate (0.1), the agent population, after peaking, decreases to a more or less constant value. In the case of a highly unstable network where the node information changes often (probability of change = 0.5), the agent population increases to unreasonable values due to excessive spawning. Additionally, complete network information is not obtained in unstable

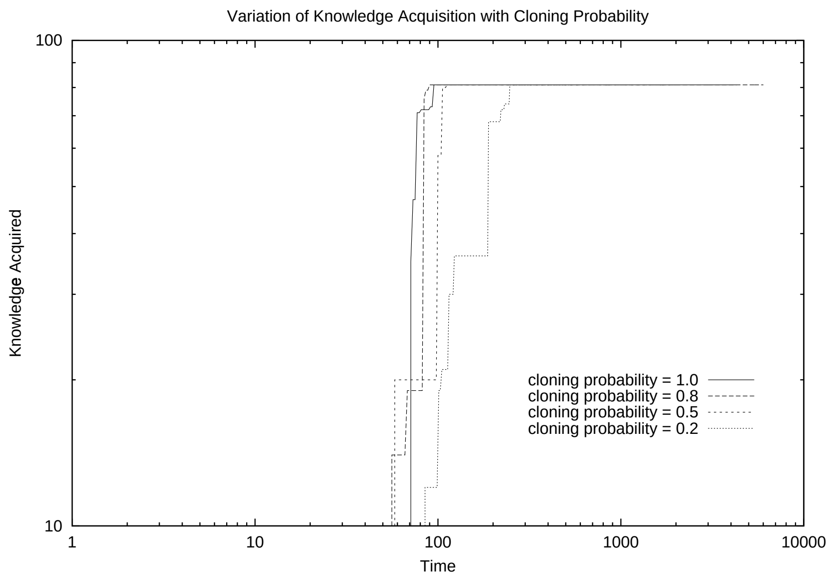

Figure 4.16: Variation in Acquired Knowledge with Cloning Probability (81 node grid)

network because the agents are always on the search for information and do not easily reach the expiration (return to home node) state.

4.2 Analysis of Agent Based Knowledge Acquisition

Knowledge acquisition in computer network systems can be broadly classified into two tasks: the communication between the querying entity and the queried entity (machine address, sequence of machines to be queried, communication issues, reliability and robustness etc) and the actual querying of the information at its source (namely the machine , router etc). The details of information extraction at the location of the network entity is achieved by using data mining and parsing techniques and a detailed discussion is beyond the scope of this project.

Schonwalder [39] has described typical methods by which network entities are polled. He classifies (conventional) network monitoring systems as either passive

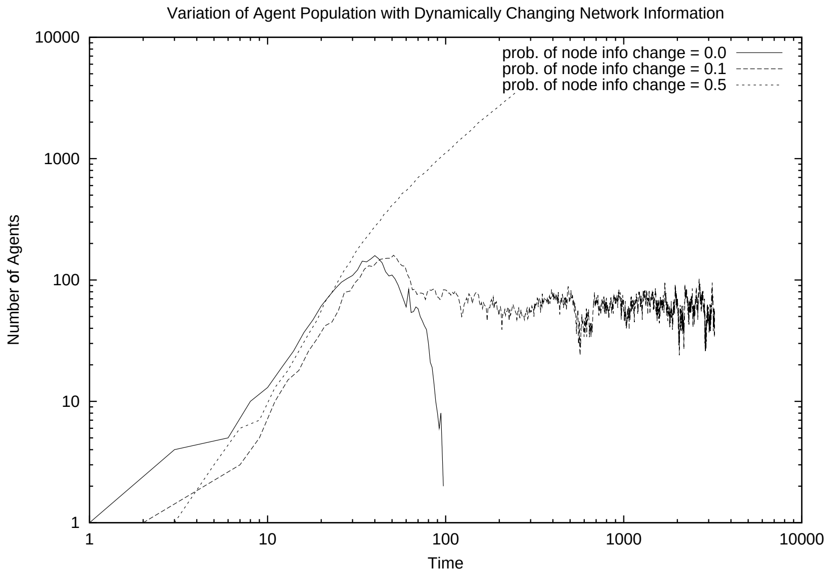

Figure 4.17: Variation in Agent Population with Network Instability (81 node grid)

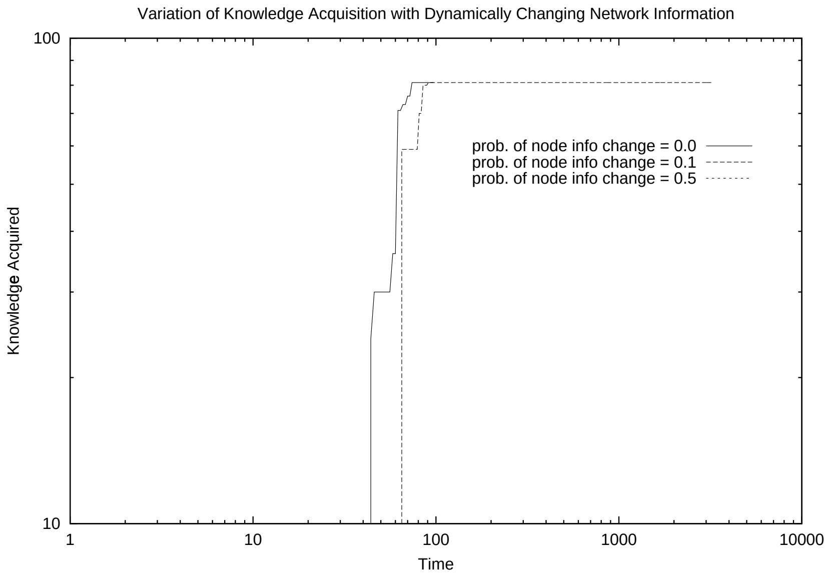

Figure 4.18: Variation in Acquired Knowledge with Network Instability (81 node grid)

or active. According to him, passive strategies, while using less bandwidth, are not generally as effective as active, bandwidth hogging round-robin polling strategies. The difference is basically a contrast between a conservative and a pro-active strategy. The agent knowledge acquisition system offers potential advantages over the conventional methods because:

- It is scalable, adapting dynamically by means of the "wave computation" mechanism to varying network size and configuration.
- The agents can adapt and can switch between a pro-active (cloning) and passive (merging) state as the computation proceeds.
- All the communication is performed locally at the network entity without need for intervention by any other entities over the network, hence conserving valuable network bandwidth. This contrasts strongly with the approach employed by active network management systems which constantly poll all reachable interfaces over the network.
- Once deployed, the agent based system is completely distributed, spreading the network traffic over the entire network canvas.

The rest of the section deals with the analysis of trends observed in the knowledge acquisition simulation described in the previous chapter. The primary intention is the identification of characteristics which give the agent system an edge over conventional packages. In order to study the behavior of the agent model described in previous chapters, the problem is approached from a different viewpoint with the intent of describing the effect of cloning, merging, agent population etc on the performance of the knowledge acquisition process. In addition, the effect of system parameters on these factors is also considered.

4.2.1 Cloning

Cloning can occur when an agent finds new information at a node ie. information differing from its own records. With this in mind, it can be expected that fervent cloning will occur at the beginning of the simulation. This has been noticed and termed "population explosion". The tendency of agents to clone, however, is affected by several factors. For instance, it has been observed that population explosion becomes more pronounced as the graph density is increased. This is because in sparser graphs, there are fewer choices of paths to travel along, and agents tend to jump between previously visited nodes, and do not clone. Since cloning can occur when new information is discovered, it enables the agent system to automatically scale to larger or smaller network sizes. This has been verified by noting the relative insensitivity of certain key simulation parameters (maximum agent population, time at which maximum agent population occurs, time at which complete network knowledge is available) to the number of initial agents, within limits.

Excessive cloning, however, impairs overall performance, because of the high traffic generated by the agents. Additionally, when a large number of agents are spawned early in the simulation, the total amount of knowledge acquired per agent is relatively low. Therefore, the child agents continue to clone, increasing the total agent population unnecessarily. This reduces the efficiency of total knowledge acquisition at the source node, as measured in simulation time units. This issue can be resolved, to some extent, by making the decision to clone (if otherwise possible) a probabilistic one, by employing a fixed probabilistic cloning. This is referred to in the previous chapter as "agent fertility". It should however be noted that the use of very low cloning probabilities results in slow and possibly incomplete knowledge acquisition. A better scheme whereby the cloning probability "decays" exponentially with simulation time has also been implemented, and performs better than the fixed probability scheme.

The amount of exponential decrease is controlled by a parameter, the optimum value of which is best determined by experiments on the simulated network.

Another way to control cloning is to link it to node cardinality and allowing an agent to clone only when its node has more than a specified number of directly connected neighbors. This allows fine grained control over the spatial location of nodes where cloning can occur and is potentially useful to restrict agent activity to certain regions. For instance, in an experiment on a square grid where no node has more than 4 directly connected neighbors, a dramatic difference in agent population profiles as the minimum node cardinality was increased.

4.2.2 Merging and Expiring

When the agent does not discover new information at a node, it attains a "merging" state whereby it resides at a node for a an amount of time and merges with other agents in the same state. If an agent remains in the merging state beyond a certain time, it assumes that there is no longer any information it can extract about the network. At this time, it decides to enter the "expiration" state whereby it returns to the source node where the agent computation was spawned. The merging and expiration behaviors enable the agents to deposit the information at one collection site where a complete picture about the network can be assembled from information harvested by different agents. It should be noted that this mechanism does not require that every agent visit every node in the network. Instead, it relies on heuristics which try to ensure that every node of the network is visited by at least one agent. If this "reasonable" demand is met, complete network information can be obtained once every agent spawned in the course of the simulation decides to expire and return to the source node.

The danger in such an approach occurs when the agents never , for whatever reason, travel to certain areas of the network. In such a situation, information about such nodes is never obtained. Proposed remedies include the judicious choice of node from which the agents are initially spawned, increase in maximum time agents can roam, multiple agent spawning locations etc.

4.2.3 Changing Network Information

In the event of the agent-based knowledge acquisition system being used for active network monitoring applications, it is very likely that the information being probed for will change over time, either sooner (system load, users logged on) or later (IP address, subnet). It is important to know what degree of system entropy the agent system can successfully cope with. In a system with rapidly changing node information, the agent will clone when it senses new information. This can lead to uncontrollable population explosion if it happens very often with the result that the agents never reach merge and expiry states. Hence, no information would be gathered at the source node, since the agents would never return there. On the other extreme, in the case of a static network, simulation results indicate a smooth well behaved population convergence. Experiments with intermediate situations indicate that the proposed agent system can cope with some degree of change in node information over time, with the result that the total agent population converges to an average value of slightly less than the number of network nodes / entities.

4.3 Performance of the developed Mobile Agent System

Previously, the architecture of the developed agent system was described. In this section, the real-time operation of the system is analyzed.

4.3.1 Operation

A true mobile agent in an operation should have 3 attributes: the code, the state and the location. In the system developed, since classes are fetched from the class server and written to disk before loading into the Java virtual machine (JVM), the need to fetch classes over the network from class servers decreases over time, as they are cached on the local machines and in the JVM. As a consequence of class distribution over the network, only the agent object (roughly equivalent to the agent state) need be transported between network entities. This has the potential to further reduce the network traffic attributable to agent traffic. Additionally, since the class files are loaded from trusted class servers (assumed), it is impossible to introduce malicious agent classes into the system.

In order to further distribute the class loading task workload, class servers could easily be set up to co-exist with the agent servers thereby distributing the task of class server distribution.

There are two potential disadvantages to this agent system, which can be overcome without too much extra effort:

- If it is required to add extra functionality to an agent class after the system has reached steady state (the agent classes have already been loaded into memory or have been distributed over the network), the agent server utilizes the methods existing at the local site ie. it does not recognize to reload the classes. This issue can be resolved by communicating with the server via a communication preamble implemented in the serialized stream.

- In the issue of two (or more) agents arriving at the server and attempting to fetch a class, the server will attempt to fetch the same class file twice and write it to the same local copy. This could cause data corruption. This can be resolved by constructing a simple synchronization mechanism to avoid requesting a download of the same file twice immediately, and / or wrapping access to the file-fetching class in a semaphore-like mechanism.

- Support for cloning, merging , inter-agent communication and multicast / broadcast agent transmission have not yet been implemented.

Since the agent system is implemented completely in Java, it can easily be extended witth little or no additional effort to operate cross-platform, a useful capability not exhibited by many of the other agent systems studied.

4.3.2 Potential Agent Applications

Here, an attempt is made here to describe a few sample applications for the system developed.

Knowledge Acquisition   The system is naturally suited to this application. It would be relatively straightforward to write an agent to accomplish this. Cloning and merging, if implemented would enable dynamic distributed agent population control.

Data Distribution   The use of class servers to deliver class file content to various network machines has already been mentioned. Another novel application stemming from this capability is the use of agents to also download other files (possibly used by other applications) to their host machines, thus automating the task of information distribution on a network.

Distributed Computation   Agents could distribute computational programs to various computers, compute the programs locally, and then transmit the results to some specified server waiting to receive the results.

Summary From the analysis of the simulations on agent-based knowledge acquisition, it appears that the agent based knowledge acquisition algorithm can be applied to yield superior results as compared to conventional polling techniques. The effect of various experimental parameters on the performance of knowledge acquisition with agents has been studied and useful information on observed trends has been gathered. The next chapter applies closure to this project by summarizing the findings and suggesting improvements and possibilities for future work.

CHAPTER 5

FUTURE WORK AND CONCLUSION

5.1 Conclusion

The project so far has attempted to evaluate, through simulation and experimentation, the applicability of mobile agents to knowledge acquisition and routing. The advantages and disadvantages of the mobile computing approaches have been identified. The simulation tools developed have been found useful to predict the effects of various parameters on system behavior.

As long as very sparse networks are not used, gent-based routing was observed to be more efficient in terms of network bandwidth consumption, thereby freeing the network for other tasks. A hybrid routing method consisting of a compromise between agent and conventional routing techniques could offer the benefits of both, without the disadvantages of either.

Simulation studies on the use of agents in knowledge acquisition indicate that optimal agent performance is observed when monitoring static or relatively persistent data from the network, although the proposed system can be adapted to dynamically varying networks by reducing the fertility of the mobile agents. The population control mechanism studied was found to behave robustly under a wide variety of operational circumstances.

A working cross-platform agent system architecture was designed and implemented using the Java Development Kit (JDK1.2). The design has been successfully tested and the system can be applied to a variety of tasks in the distributed domain.

To conclude, the results of this project indicate that mobile agents can match, and even improve upon, performance standards set by the network management strategies which were studied. The development of the agent programming paradigm and logic has been to be as, if not more, important as the choice of agent system.

5.2 Future Work

The successful study of mobile agents in the network domain proves the worth of simulation tools in evaluating mobile agent behavior models. The simulation classes and libraries developed can easily be re-used to study the behavior of other agent models with a view to develop a better understanding of the dynamics of mobile agent behavior in different scenarios.

This research has already demonstrated the applicability and scalability of intelligent mobile agents to network monitoring and knowledge acquisition. However, there is room for further research and improvement in some areas. For example, the concept of explicitly specifying the cloning probabilities may not be suitable in all cases. An alternative possibility to this could be the implementation of algorithms where the agents could dynamically and implicitly determine when to clone or shrink based on the relative amount of information acquired over time. Here, the doubling of an agent's information base could serve as the impetus to clone. Such mechanisms would be self-regulatory and not require the use of explicit probability information. Similar schemes could be devised for merging, expiry etc.

Once a suitable agent model for a particular application has been developed and studied, the next step would be the implementation of real-life agent systems and the verification of the accuracy of the simulation results. In this project, the focus has been on the development of a simulation model for the acquisition of knowledge in a dynamic network and the development of a distributed scalable agent population control mechanisms. Currently, there exist several problems in the distributed computing domain which merit investigation using the intelligent mobile agent approach. A key advantage of using multiple co-operative agents in solving distributed tasks is the implicit fault tolerance that arises from the redundancy resulting from multiple agents working on the same task. In the following paragraphs, some likely applications are described.

Agents in Aggregated Clusters: Intelligent mobile agents could be utilized in the management of large computational infrastructures, such as clusters. These management of such systems require highly scalable, flexible , robust and fault tolerant mechanisms. Mobile agent systems have demonstrated these qualities in the simulations performed so far and could be integrated into a mobile network management layer in future cluster system environments.

The Scheduling Problem: In situations where multiple sites contribute cluster resources to a larger community with a view to constructing a large distributed virtual super-computer, the co-ordination between various job schedulers raise several issues. These issues are related to online resource monitoring and trading. Mechanisms must be designed to acquire and represent the global resource state in order for the cluster schedulers to operate and co-operate efficiently. Mobile agents could be applied to solve some of these issues such as:

- Distributed Cluster Resource Management: In a distributed cluster environment, jobs can be submitted to other clusters if turnaround time and global scheduling could be improved. however, the issue of job migration raises a number of challenging questions. For instance, some jobs cannot be migrated to other clusters because the data files would need to be transferred. Therefore, jobs would have to be classified as remote executable or strictly local. Additionally, in addition to scheduling jobs currently in the local queue, a cluster must be able to advertise idle resources which can be matched with jobs from other clusters. A third problem is the estimation of process progress. This can be rather speculative and the incorrect estimation of jobs can reduce cluster utilization. In a distributed cluster environment, runtime estimates also need to be weighted across different weighted clusters depending on the computational power of the clusters.

- Resource Monitoring: This is one of the most challenging tasks in distributed cluster computing. The basics of resource monitoring have already been described earlier in this project. Mobile agent systems could be deployed in large aggregated cluster facilities to monitor and acquire relevant data and deliver it to the user as required.

Finally, with regards to the developed agent system , its functionality could be further enhanced to provide the developer greater power and flexibility in designing agent applications. The addition of multicast and broadcast capabilities to the server to enable rapid agent deployment, if so desired. The implementation of an agent communication mechanism would allow agents to exchange information either with other agents on the same host or even over the network.

APPENDIX A

Graph Generators

This chapter describes the programs used to generate the more complex network topologies used in the simulations.

Random Graph Generator   This program has the ability to create random graphs of varying density (given the number of nodes and edges). The program has two parts. The first generates a spanning tree thus:

1. Take two arrays. One is full with node numbers (UNCONNECTED) and the other(CONNECTED) is empty initially.
2. Take out randomly two nodes from UNCONNECTED, put them in CONNECTED and add the edge to matrix.
3. Now take one node from each move the node in UNCONNECTED to CONNECTED and add the edge to matrix repeat 3.
4. After matrix has N-1 edges we have a spanning tree.

The second part then adds edges to increase the density of the graph.

1. take two random nodes if the edge exists search again else add edge.
2. repeat till number of edges = required number.

Grid Generator   This program generates square grids of size n^2, given the number of nodes on an edge (n) as input. It operates by determining the position of each node in the end graph and its neighbors and printing out the edges between adjacent nodes. Each node can have either 4 (enclosed), 3 (edge) or 2 (corner) directly connected neighbors.

Grid Distorter   This program is actually a modification of the grid generator program which allows the user to distort the square generator initially created by varying amounts (supplied as a command line parameter). Each distortion involves the removal of a random edge whose vertices both have a cardinality of 2 or more. This ensures that the removal of the edge does not separate the graph into disconnected components. The removal of this edge is then followed by the addition of another random edge connected to a vertex which was previously connected to the removed.

BIBLIOGRAPHY

[1] "Modern Operating Systems", A. Tanebaum. Prentice Hall.

[2] "Agents: Not just for Bond anymore", JavaWorld April 1997. http://www.javaworld.com/javaworld/jw-04-1997/jw-04-agents.html

[3] Mobile Agents White Paper, http://www.genmagic.com/technology/techwhitepaper.html .

[4] "Mobile Agents: Are they a good idea?", C. Harrison, D. Chess, and A. Kershenbaum, IBM 1995

[5] N. Muller, "System Management: The Emerging Role of Intelligent Agent", June 23, 1999
http://www.ddx.com/agents.shtml

[6] K. H. Kramer, N. Minar, and P. Maes "Tutorial: Mobile Software Agents for Dynamic Routing", Mobile Computing and Communications Review vol.3 no.2, 1999.

[7] T. Sander and C. Tschudin, "Protecting Mobile Agents Against Malicious Hosts", Lecture Notes in Computer Science. 1998, V 1419, p 44.

[8] G. Necula and P. Lee, "Safe Untrusted Agents Using Proof-Carrying Code", Lecture Notes in Computer Science, 1988, V 1419, p61.

[9] "G. Vigna, "Cryptographic Traces for Mobile Agents", Lecture Notes in Computer Science, 1988, V 1419, p147.

[10] M. Abdadi and J. Feigenbaum, "Secure Circuit Evaluation", Journal of Cryptology, 2(1), 1990.

[11] D. Johansen, R. van Renesse, F. Schneider, "An Introduction to the TACOMA Distributed System", Computer Science Technical Report 95-23, Universitet Tromso, Institute of Matehmatical and Physical Sciences, Department of Computer Science, University of Tromso N-9037, Tromso, Norway, June 1995.

[12] P. Holger, T. Stolpmann, J. Nehmer, "Ara: Agents for Remote Actions", http://www.uni-kl.de/AG-Nehmer/Ara .

[13] J. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley, 1994.

[14] "Readings in AGENTS", M. Huhns and M. Singh, Morgan Kaufmann Publishers Inc. 1998.

[15] A. Fuggetta, G. Picco and G. Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering, vol 24, no. xx.xxxx 1998

[16] R. Gray, "Agent Tcl: A transportable agent system" Proc.of the CIKM Workshop in Intelligent Information Agents", Baltimore Md., December 1995.

[17] H. Peine, T. Stolpmann, "The Architecture of the Ara Platform for Mobile Agents", In: Rothermel K., Popescu-Zeletin R. (Eds.), Mobile Agents, Proc. of MA'97, Springer Verlag, Berlin, April 7-8, LNCS 1219, pp 50-61.

[18] R. Borgia, P. Deganom, C. Priami, L. Leth and B. Thomsen, "Understanding mobile agents via a non-interleaving semantics for Facile" Technical Report ECRC-96-4, European Computer-Industry Research Centre, 1996.

[19] Sun Microsystems "The Java Language: An Overview," Tech. Report, Sun Microsystems, 1994.

[20] D. Lange, "Java Aglets Application Programming Interface (J-AAPI)," IBM Corp. White Paper, Feb. 1997.

[21] J. Baumann, F. Hohl, K. Rothermel and M. Straber, "Mole - Concepts of a Mobile Agent System", World Wide Web, Vol. 1, Nr. 3, pp. 123-137, 1998.

[22] L. Cardelli, "A language with distributed scope", Computing Systems, vol 8 no. 1, pp 27-59, 1995.

[23] K. Bharat and M.H. Brown, "Building distributed applications by direct manipulation", Proc. UIST'94, 1994.

[24] K. Bharat and L. Cardelli, "Migratory Applications" Proc. of the ACM Symposium on User Interface Software and Technology '95, 133-142, 1995.

[25] M. Ranganathan, A. Acharya, S. D. Sharma and J. Saltz, "Network-aware Mobile Programs" Proc. of the USENIX Annual Technical Conference, Anaheim,California, 1997.

[26] A. Acharya, M. Ranganathan, and J. Saltz, "Sumatra: A Language for Resource-aware Mobile Programs," In Vitek and Tschudin [731, pp. 111-130.

[27] R. Gray, D. Kotz, G. Cybenko and D. Rus , "Mobile Agents: Explanations and Examples", Manning Publishing, 1997.

[28] F. Knabe, "Language Support for Mobile Agents", PhD thesis, CMU, December 1995.

[29] S. Miyashita "About e-Marketplace", http://www.trl.ibm.co.jp/aglets/emplace/emplace.html .

[30] H. Peine "The Ara Platform for Mobile Agents", http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/index_e.html .

[31] P. Tarau, "Inference and Computation Mobility with Jinni", http://www.binnetcorp.com/Jinni/ .

[32] W. Jansen and T. Karygiannis, "NIST Special Publication 800-19 Mobile Agent Security", National Institute of Standards and Technology Computer Security Division Gaithersburg, MD 20899.

[33] J. Ousterhout, J. Levy, and B. Welch, "The Safe-Tcl Security Model," Technical Report SMLI TR-97-60, Sun Microsystems, 1997.

[34] J. Tardo and L. Valente, "Mobile Agent Security and Telescript ", Proceedings of IEEE COMPCON '96, Santa Clara, California, pp. 58-63, February 1996, IEEE Computer Society Press.

[35] R. Gray, "Agent Tcl: A Flexible and Secure Mobile-Agent System", Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96), pp. 9-23, July 1996.

[36] G. Karjoth, D.B. Lange and M. Oshima, "A Security Model For Aglets," IEEE Internet Computing, August 1997, pp. 68-77.

[37] "F. Hohl, "Time Limited BlackBox Security: Protection Mobile Agents from Malicious Hosts", Lecture Notes in Computer Science, 1988, V 1419, p99.

[38] R. Kooijman, "Divide and conquer in network management using event-driven network area agents", Technical Report, TU Delft, March 1995.

[39] J. Schonwalder, H. Langendorfer, "How To Keep Track of Your Network", Configuration Proc. 7th Conference on Large Installation System Administration (LISA VII) Monterey (California), November 1993.