AN ANNOTATED BIBLIOGRAPHY
OF MOBILE AGENTS
IN NETWORKS

Sandhya Sriraman, M.A.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2002

APPROVED:

Armin R. Mikler, Major Professor
Tom Jacob, Committee Member
Azzedine Boukerche, Committee Member
Paul Tarau, Committee Member
Karl Steiner, Committee Member
Robert Brazile, Graduate Advisor
Krishna Kavi, Chair of the Department
of Computer Science
C. Neal Tate, Dean of the Robert B. Toulouse
School of Graduate Studies

Sriraman, Sandhya, An Annotated Bibliography of Mobile Agents in Networks. Master of Science (Computer Science), December 2002, 254 pp., 2 tables, 34 figures, 212 titles.

The purpose of this thesis is to present a comprehensive colligation of applications of mobile agents in networks, and provide a baseline association of these systems. This work has been motivated by the fact that mobile agent systems have been deemed proficuous alternatives in system applications. Several mobile agent systems have been developed to provide scalable and cogent solutions in network-centric applications. This thesis examines some existing mobile agent systems in core networking areas, in particular, those of network and resource management, routing, and the provision of fault tolerance and security. The inherent features of these systems are discussed with respect to their specific functionalities. The applicability and efficacy of mobile agents are further considered in the specific areas mentioned above. Although an initial foray into a collation of this nature, the goal of this annotated bibliography is to provide a generic referential view of mobile agent systems in network applications.

ACKNOWLEDGEMENTS

I could not begin this work without first conveying my heartfelt gratitude to Dr. Mikler, my advisor, guru, friend, philosopher and guide. You entrusted me with this unique concept, backed me against all odds and have been my mentor every step of the way. You taught me to think “out of the box”, made this work not only possible, but also a memorable and enjoyable experience for me. Thank you, Sir.

To my committee members, Drs. Jacob, Boukerche, Tarau and Steiner, I would like to extend both my thanks and my sympathies - you have had to go through this not insignificant document and you have taken the time to give me your valuable suggestions, and be on my committee. Thank you all for your efforts!

This thesis would not have been written without Prasanna. He encouraged and motivated me even in the direst of circumstances, and made me see this through till the end. In addition, he sincerely and painstakingly edited every single sentence of my work - only the second person to do so - at his own peril!

My room-mates - Shubi, Anu, Sandhya, Vivek, Deepak, Satyam, Prasanna - your support, bolstering me and my confidence when required, taking care of me, suffering my numerous tantrums and being my second family here has meant the world to me. I will always cherish our memories together. Vandana and Cliff, my deepest appreciation for your time and input on my work. My peers and office-mates (Ke Xu and Saqib in particular), thank you too for humoring my idiosyncracies! To all those that I have not mentioned here, I second this.

The Computer Science Department faculty and staff has been instrumental in providing me this opportunity to work on and complete my Master’s here. Dr. Tate, my graduate advisor, thank you for insisting that I take all my prerequisite courses, then trusting me enough to teach the same here and for always giving me your time when I needed to talk... Sharon, Pam, Mary, Lisa, thank you all for patiently tolerating my not infrequent visits to the office, being ever-helpful and always making me feel welcome.

Hima, Tejo, Giri, Ranjeesh, Abida, Priya - my friends from home, with the efforts that you all took to constantly communicate with and encourage me, I have never been alone in my work here. I could not end this without a mention of the invaluable support that my family has given me always. My decision to change majors, to be independent here, to yet always know that I would be backed in any situation has only been a result of the unconditional love that my parents have given me, and of course, the exceptional examples that my sisters have set. Thank you, Amma, Appa, Ajju and Papu - you have been the best role models I could ask for.

PREFACE

The area of mobile agent research is considerably vast, and significant advances have been made in this field. Many applications in Artificial Intelligence (AI), Distributed Artificial Intelligence (DAI) and distributed computing in general, employ mobile agent technology to enhance performance. Several models of agent systems have been developed and implemented to achieve the goals of these applications.

Building a mobile agent system, implementing it in an application-specific environment, and testing its performance is an endeavor undertaken by many proponents of mobile agent systems. While by no means trivial, building such a system would only add to the already profuse collection of multi-agent systems in existence today. What is required however, is an objective view of the systems that are already available, and an analysis of the same to help in achieving standardization.

The quest for the perfect agent system is ongoing. With this Annotated Bibliography on Mobile Agents, an attempt has been made to classify, correlate, and cohere the current agent system models used today, with a special focus on the functions of mobile agent systems in networks. Modest modifications and enhancements have been suggested to the agent systems reviewed herein, in order to promote the underlying common thread of impeccable, intelligent performance by the mobile agent systems.

TABLE OF CONTENTS

Page

Chapter

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Agent is as agent does. An agent is characterized by what it can do, and what it actually does. Agents are entities that act on behalf of other entities. The possibilities of agent technology are boundless and can be used in a wide variety of applications in almost every sphere of life today.

The scope of this work is to trace the mobile agent paradigm from its conception to its subsequent use in various applications in distributed computing systems, with special focus on agents in networking. The concept of agents was first realized in an abstract manner with the idea of robots - physical entities that carried out predefined tasks for humans. This soon took the shape of static pieces of computer code equipped with specific features to carry out defined tasks, with the advent of computing systems. Agents in Artificial Intelligence (AI) and Distributed Artificial Intelligence (DAI) soon became widespread, and many applications in ecommerce, industry, etc. use the agent paradigm extensively. Mobile agents are pieces of code with the ability to migrate between locations and perform actions as required. Agents are being used to improve and enhance the performance of systems, and they are provided with attributes of intelligence, communication and adaptability to this end.

History of Agents

Agents have been defined by many researchers in a variety of different contexts. The term *agent* automatically brings to mind an outside entity carrying out and performing tasks on behalf of someone else. Indeed, human agents, such as travel agents, are still in vogue today. The term as we recognize it in the field of computer science however, is that of a mechanical entity or a program, performing pre-defined tasks and interacting with the user to perform these tasks.

The term *robot*, commonly interchangeably used for an agent, was coined by the Czech playwright, Karel Capek, in his play called Rossum's Universal Robots [32], as early as in the 1920s. The word robot, derived from *robota* in Czech, means servitude or work, and the word rossum derived from *rozum* in Czech means reason or intellect. The play introduced the concept of "cheap labor, and cheapening output" by using "living and intelligent working machines". The play goes further to reiterate that

> to manufacture artificial workers is the same thing as to manufacture motors. The process must be the simplest, and the product must be the best from a practical point of view... the cheapest. The one whose needs are the smallest... Mechanically they are more perfect than (we) are, they

> have an enormously developed intelligence...

This then is the basis for all subsequent artificial workers that have been created.

While the robots described above refer to a mechanical entity or humanoid, a software agent or softbot, derived from the term robot, refers to a computer program that gathers information and provides a specific service to the the user. Software agents were first believed to have been conceived by John McCarthy in the mid-1950s [126]; the term was officially formulated by Oliver G. Selfridge a few years later [175]. This signified the birth of the early agents in Artificial Intelligence. These *agents* were actually computer software programs that would carry out specific operations, interact with the human user to ask for advice when "stuck" and produce the required results [107]. They were restricted to the computer world, and their only interface with the real world was the user of the computing system.

From the robots in Capek's play to Frankenstein and Pinocchio, there have always been human-like manifestations of automated *life* in machines (anthropomorphism). Each of these inventions has been attributed human qualities, and the ability to perform tasks for their creators. Isaac Asimov [7], in *'I, Robot'* has traced the gradual acceptance of robots in human society from complete technophobia to where the robots themselves rule supreme. These excerpts of fiction are lucid in describing how programmed objects may misinterpret commands, run amok and cause chaos if not controlled explicitly and clearly.

From these robots to the agents in computer science has been but a short transition, and one that has involved many years of rigorous research and radical thinking on the part of many computer scientists.

Agents in AI

Agents were initially introduced and used in Artificial Intelligence (AI) as a natural extension to the embodiment of intelligent computing. Carl Hewitt's *actor* model of an agent [91] is the earliest reference to the use of agents in AI. In the *Actor Formalism*, Carl Hewitt, et al. defined an agent as a "self-contained, interactive and concurrently executing object" which had an "internal state and could respond to messages from similar objects" [92]. Thus, as early as in the 1970s, agents were conceived as being able to inter-communicate and act upon the basis of these communications.

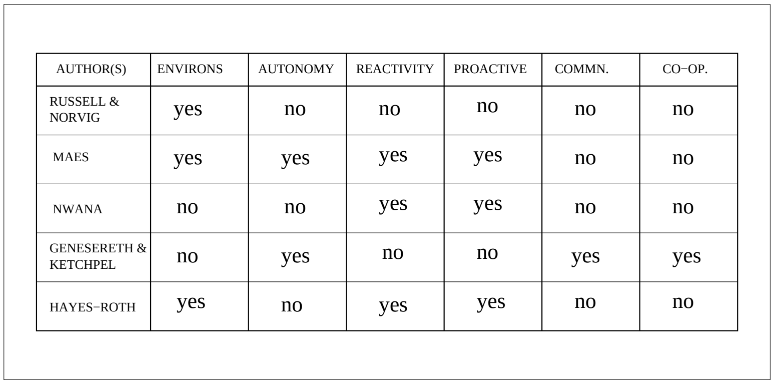
Agents in AI have been developed and engineered for use in a variety of commercial and research applications. Software agents and multiagent systems form a part of DAI and extensive research has been conducted in these areas. To this end, agents have been designed to be modular, fast, reliable and flexible [146]. Agents have also been attributed various qualities of human beings, such as knowledge, intentions, and abilities or wants [156]. This was a new tack of programming, reasoned and expounded in John McCarthy's publications [125]. Since then, the typology of agents has become more attribute-oriented and agents can be delineated based on their behavioral aspects.

Agent Taxonomies

Several taxonomies of agents have been put forth, most notably those of Franklin and Graesser [71], Genesereth and Ketchpel [74] and of Nwana [145]. While no two definitions of agents match exactly, they mostly agree on some key aspects and attributes of agents, such as autonomy, flexibility and the ability to communicate [100]. These attributes have of course come to be associated with the more intelligent agents that are being developed today. While a spellchecker program in a word processing editor was also heretofore (albeit incorrectly) considered an agent, the computing world is awakening to the very real and powerful abilities offered by agents, and agent designs and applications are becoming more ambitious.

The crucial notions associated with agents are further expounded with an example in Lenny Foner's paper [67]. Julia, the agent developed by the MIT Media Lab, is an agent that has sociological characteristics that typify her agent-behavior. Foner enumerates the qualities of agents based on the example of Julia as an agent. An agent then, should possess autonomy, personalizability, the ability to discourse, be capable of taking risks and reposing trust in, possess a domain of interest, cooperate with the user, fulfill expectations and gracefully degrade.

Agents have been further classified on the basis of their characteristics. There are agents that are mobile or static depending on their ability to traverse different locations. Agents can also be considered either deliberative or reactive. Deliberative agents are those agents that can plan and negotiate with other agents in order to achieve their goals. Reactive agents on the other hand respond to the environment depending on how they are programmed. They do not possess internal processing, and are a condition-reaction based model. The qualities of autonomy, learning and cooperation are also attributed to agents. Autonomous agents are those that do not require user-intervention, and can carry out their tasks efficiently and with a degree of flexibility. Agents can imbibe environmental changes and act according to the changes. These agents communicate the changes to other agents to enable the planning of an itinerary according to these changes, if required. These characteristics of learning about the environment, communicating the changes and cooperating and coordinating with other agents is invaluable for autonomous agents to perform their tasks.



AUTHOR(S)	ENVIRONS	AUTONOMY	REACTIVITY	PROACTIVE	COMMN.	CO−OP.
RUSSELL & NORVIG	yes	no	no	no	no	no
MAES	yes	yes	yes	yes	no	no
NWANA	no	no	yes	yes	no	no
GENESERETH & KETCHPEL	no	yes	no	no	yes	yes
HAYES−ROTH	yes	no	yes	yes	no	no

Table 1.1: Attributes of Agents as envisioned by different researchers, table taken with permission from Mark van Assem [199]

The characteristics attributed to agents by various researchers can be summarized in the table (see Table 1.1) depicted by Mark van Assem in his paper, Limitations of Agent Classifications: Subjective Objectivity [199]. These characteristics may often overlap to form multiple definitions of agents. An agent could be classified therefore as a mobile deliberative agent, or as a static reactive deliberative interface agent, and so on. There are certain attributes that are implied in the presence of other attributes, for example, an agent could be considered as being communicative if it is known to be cooperative.

However, it is necessary that a formal definition of agents and agent systems be laid down, alongwith a list of attributes that are necessary or optional. Agents are not just programs; rather they are autonomous entities that can reason and be executed based on necessity. They have been recognized to possess not only reactive ability, but also the ability to be proactive. An agent can sense its environment and the changes therein, and take decisions accordingly [164]. Further, Pattie Maes requires that agents must not only carry out the required tasks, but also realize the goals set for them [117].

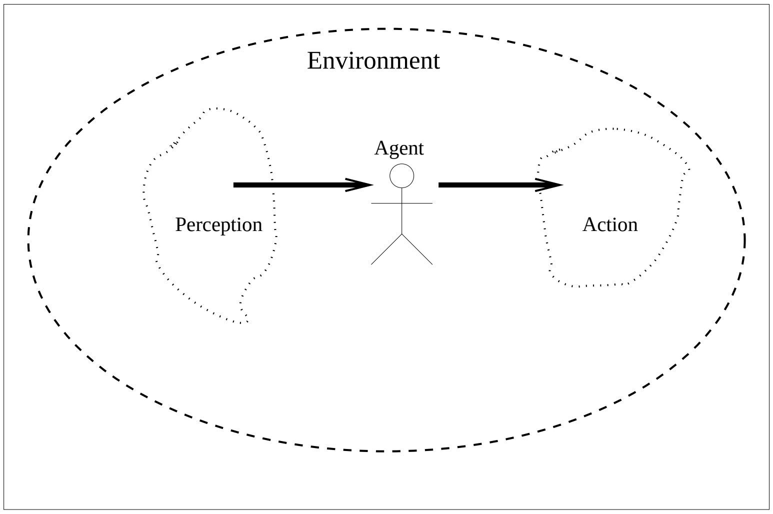


Figure 1.1: Percept Sequence of an Agent that effects/affects its environment

An agent can perform actions based on what is termed as its *percept sequence* [164]. This refers to the perceptions of an agent from the start of the specification of the task to be performed, which determine what action the agent should ideally take in order to optimize performance (see Figure 1.1). There also needs to be a suitable architecture to build the agent. Thus, an agent has been mathematically and extremely generally defined as the aggregation of a program and an architecture.

$$Agent = Program + Architecture$$

Agents have been broadly classified in AI as being reflex-based, goal-based, or utility-based [164]. Reflex-based agents perform actions following a set of condition-action rules, which they are pre-equipped with. Goal-based agents, on the other hand, take actions depending on their goal, and are therefore more flexible than the reflex-based agents which are confined only to those condition-action rules that they contain. Utility-based agents improve on goal-based agents by demarcating between the different ways of achieving goals, and determining which method yields the highest utility or *happiness* in terms of efficiency and performance.

Brustoloni's agents [28] have been classified as regulation agents, planning agents or adaptive agents. The regulation agents are the agents that have the most knowledge and always know what to do under given circumstances. These are however static, and can only function based on the very specific subset of information that they contain, and can be equated with the reflex-based agents defined above. Planning agents have the capacity to plan their course of actions, based on different algorithms which could be derived from AI or case-based (condition-action), or random. They are therefore slightly more developed, and more complex to implement. The adaptive agents are capable of planning and learning, and can therefore act autonomously. These agents are the closest to the intelligent agents that are in use today.

Agents have been accorded various attributes as means to achieving the goals set for them. Agents among other things, can be autonomous, goal-oriented, collaborative, flexible, self-starting, communicative, adaptive, mobile and possess character and temporal continuity [64]. It has further been recognized that in addition to these properties, agents also possess the characteristic of being able to effect the way in which they sense the environment in the future [71].

Multi Agent Systems

Multi-Agent Systems (MAS) are defined in DAI as systems that consist of a network of agents interacting with each other (see Figure 1.2) to solve problems that are out of the bounds of their individual abilities [62]. MAS are characterized in consisting of agents that have only limited capabilities, lack of global control, decentralized data and asynchronous computation [100]. There are several design issues associated with MAS - a problem needs to be allocated among different agents effectively, and the agents need to interact and exchange information and results. Further, the agents should be able to solve problems of conflicting interests in an acceptable fashion; engineering of practical and non-chaotic solutions is often the pressing need with MAS.

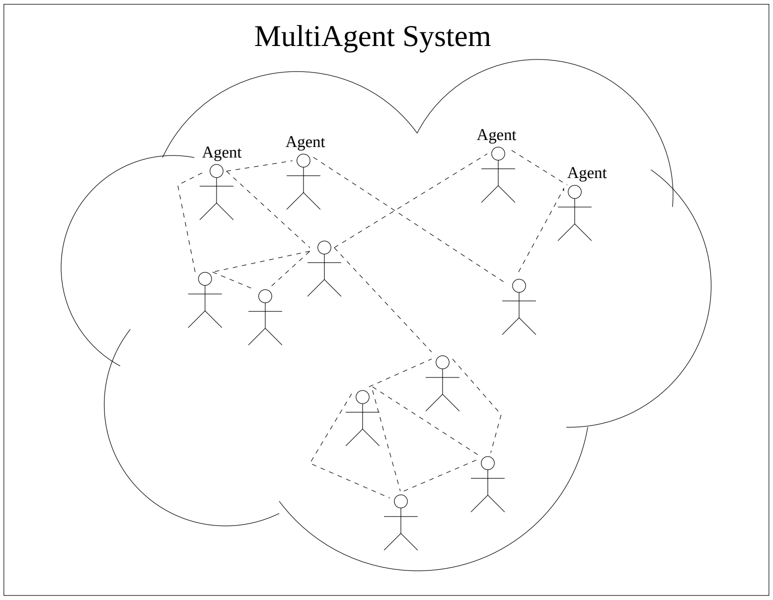


Figure 1.2: Interacting Agents in a Multi Agent System

Agents in Applications

One of the earliest application of agent systems was in the area of air traffic control. Conflicts between agent groups were resolved, and parallel solutions were applied to construct flight plans of aircrafts that would maintain a safe distance from each other. Additional constraints such as minimizing the fuel consumption in reaching the respective destinations [30] were also satisfied.

In yet another domain, a distributed group of agents was used to develop the Distributed Vehicle Monitoring Task (DVMT). Therein groups of agents monitored areas for traffic movement and sensed the movement of traffic based on the sounds made by the vehicles. Characteristic sounds were associated with vehicles, and the agents could produce approximations of vehicle movement in the areas they monitored. These were used to improve reliability and prevent redundant tracking of vehicles in common regions [61].

Georgeff [75] introduced an agent system for the recognition of subgoal options and interactions of agents and accordingly avoid them. In this model, a synchronizer agent is aware of the goals of all the other agents, and tries to allocate and synchronize their tasks so as to avoid the unnecessary multiple blocking of resources or poor performance. This synchronizer agent is also greatly useful in the aviation problem described above, as it can avoid collision courses for the aircrafts by completely preventing interaction (colliding).

Blackboards have ben introduced in many agent applications to facilitate the necessary coordination. Blackboards were developed for the agents of MAS to coordinate and communicate with each other. Agents could thus write data onto the data structures, which would then be viewed and used by the other agents. Several works improved on this technique of agent communication by introducing specialized blackboards. Lesser and Corkill [53] used two blackboards for agent communication - one for data and one for agents' goals. The MINDS project also used two blackboards so that agents could share their tasks and information in order to achieve their goals [95]. Three blackboards, one each for agent perception, control and reasoning respectively have also been used in a different application [87].

Partial Global Planning (PGP) is another agent-based flexible approach to coordination that allows the dynamic coordination of nodes [61]. Agents communicate their goals and plans to each other, and form suitable expectations of future behavior. Thus, all agents form plans according to the new future expectations, and the entire system is dynamically coordinated to achieve the set goals. The use of these coordination mechanisms are manifested in many applications, a few of which are detailed in the next section.

Examples of Agent-based Applications

There are several identified industrial applications of agents such as the Yet Another Manufacturing System (YAMS) which is used to control manufacturing [151]. In the manufacturing system, individual workcells that perform a distinct function are identified. Several of the same type of these workcells are grouped together to form a functional manufacturing unit, and all manufacturing units are combined to form the factory. YAMS is used to control the factory and each of its units. Multi-agents represent the manufacturing units, and the production is controlled by managing parameters such as resources, time and finished products. The agents have plans and capabilities, and are given goals to achieve. Tasks are delegated so that agents can collaborate, control and achieve the set production goals.

Agents have also been applied to process control applications, such as electricity transportation management [200] and particle accelerator control [99]. ARCHON®[1]is one of the earliest multi-agent systems that has been used in real-world applications and is field-tested. Agent-based process control systems have also been to monitor and diagnose faults in nuclear power plants [205], spacecraft control [174] and climate control systems [47].

Several commercial applications, such as the management of information databases, and electronic commerce such as online trading, stock exchanges and various businesses have incorporated multi-agent systems. There is an overload of information available today, and agents are used to gather and filter all the information so that only the relevant data is made available to users. Email agents can effectively manage all

[1] ARCHON Group LP, June 9, 1998, Limited Partnership Delaware, Irving, TX

the incoming electronic mail for a user [115]. These agents actually "prioritize, delete, forward, sort and archive mail messages on behalf of the user". Financial portfolio management agents have also been developed and implemented; these agents monitor stock quotes and financial reports, analyze them, and apprise the user periodically of the same.

Electronic commerce has become very popular in recent years, and there is an increasing level of such commerce being undertaken by agents. Kasbah®[2] [40] is an electronic marketplace introduced in the MIT Media Lab, where agents are the main consumers and sellers of goods. There are several other such commercial agent applications available, such as BargainFinder [109], Jango®[3] [59], MAGMA [198], and other interactive agent-based shopping catalogs [173, 189].

There is no limit to the use and applicability of agent systems in the world today. Indeed, agents have also been used in creating interactive games, and in interactive theater, where the user and agents can enact role models in plays, operas, etc. There are agent applications that are used to provide patient/health care. Patient monitoring systems have been developed to monitor patients in Intensive Care Units in hospitals, so that they receive the constant expertise and care equivalent to that of specialists [94].

[2] Kayon Corporation, May 29, 2001, South Carolina

[3] Orang-Otang Computers Inc, Corporation Delaware, January 1, 2002, Tennessee

Agent-based technology has been widely adopted, adapted and implemented to further object-oriented distributed computing, and to provide flexible and feasible solutions to a variety of problems. The agents that have been considered so far are used in AI problems and environments, and are mostly static or non-mobile in nature. While they exhibit intelligence and possess the ability to communicate with other agents, these agents remain at one location and pass messages to one another and to information databases to obtain data. Researchers have keenly felt and explored the necessity for mobility in agents, in order to make them further qualified to achieve their goals.

Mobile Agents

Agents have been imbued with the quality of mobility in order that they may migrate from one place to another to facilitate the solution-finding process. Mobile agents form the major part of present day applications, and it is therefore important that a definition of these agents be put forth. The advent of mobile agents has eased the arduous task of data collection, and mobile agents are being used in various applications such as distributed computing and networking to aid different functionalities in them.

The earlier constraint on agents that they execute only on a single computer, or on a homogeneous computing platform is outdated today, due to the increasing heterogeneity and distributed nature of tasks to be performed. Agents are required to move around learning about and from their environment, and creating new techniques of accomplishing tasks. Mobility is thus referred to as the ability of an agent to move from one computer to another [137]. An agent is therefore required to transfer its state and be able to execute correctly on another computer, in order to be termed mobile.

Mobile agents have been developed to cope with the large amount of distributed tasks to be performed. The autonomous and flexible nature of their execution enables the mobile agents to be optimal solutions to many problems. While mobility is a given attribute in current day agents, early mobile agents were radical innovations that soon found universal acceptance.

Agent TCL at Dartmouth [83] was the earliest mobile agent to be developed. The mobile agent model provided by Agent TCL was that of an agent suspending execution at one point, migrating to another location by transporting its code and state, and resuming execution at the new location. Since Agent TCL, mobile agents have undergone many revisions in model and characterization. There are new issues to be considered, such as those of the size and performance, while also taking into account the mobility of agents. Mobile agents are therefore not just simple pieces of executable code that are capable of moving around and getting executed at will; rather, they are complex structures of various parameters, including intelligence and autonomy. Researchers are untiring in the quest for the perfect mobile agent model, and improvements on existing models are ongoing projects in many agent research laboratories.

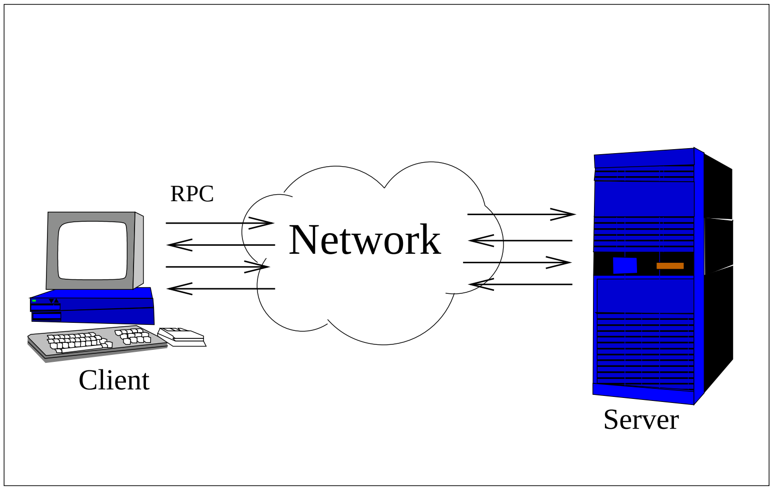


Figure 1.3: Remote Procedure Calls in a Client-Server Paradigm

Traditionally, Remote Procedure Calls (RPCs) [121] were used to execute procedures on other computers from one machine. RPCs involved communication between two machines to form the client-server pair, and the exchange of messages between them to execute procedures [17]. The client machine would message the server and send data to it, so that the server could execute the appropriate procedure using the data provided. The server would then compute the results, and send the same to the client requesting them (see Figure 1.3). This messaging between the client and the server requires the establishment of a protocol and persistent connection between the two machines to ensure continuous communication. The repercussions of these requirements are high overheads and considerable bandwidth utilization in the establishing of the communication pathways between the client and server.

The considerable overheads associated with RPC have made remote programming, or the use of mobile agents more attractive and feasible to implement. As with RPC, a protocol is established between the client and the server. The mobile agent is then sent alongwith the data to the server. The mobile agent consists of not only the data, but also the necessary code, which is downloaded and executed on the remote location. The results of this execution are conveyed back to the client by the mobile agent after the processing is completed (see Figure 1.4). The code or procedure that is conveyed by the mobile agent to the server is typically one that was initiated by the client and is required to be continued by the server.

Using mobile agents ensures that bandwidth on the network is not used in exchanging messages constantly between the client and server. Once a mobile agent has been deployed to the server after the initial protocol is set up, the network is free of further communication between the client and the server. The mobile agent functions

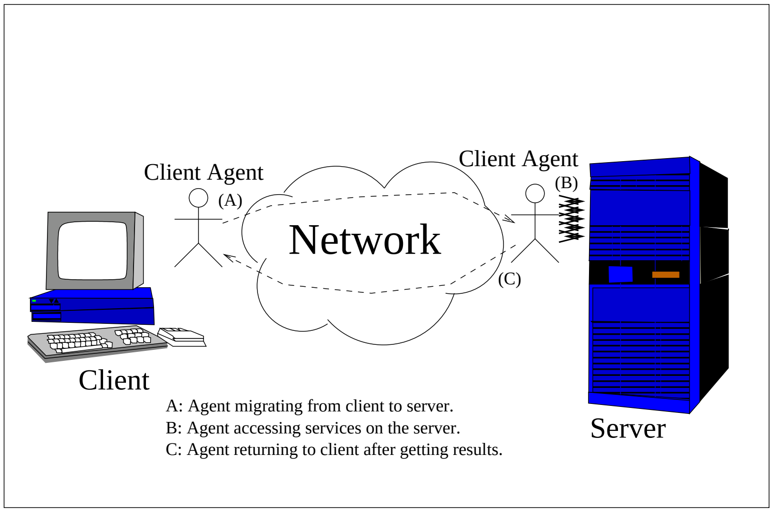

Client Agent
(A)
Network
Client Agent
(B)
(C)
Client
Server
A: Agent migrating from client to server.
B: Agent accessing services on the server.
C: Agent returning to client after getting results.

Figure 1.4: Remote Programming by a Mobile Agent

locally and returns to the client only after it has completed executing on the server and gathered the required results.

Definitions and Models of Mobile Agents

There are several definitions of mobile agents propounded over the years. Agents have been viewed as persistent entities dedicated to specific purposes [182]. Michael Coen [49] views agents as programs that "engage in dialogs and negotiate and coordinate transfer of information". Further, according to Thorisson [194], agents allow for the intelligent and autonomous decision of which sophisticated tool to use without user intervention. The addition of mobility to the other features agents were already endowed with only made it easier for agents to be adapted for use in systems rather than exclusively be used in AI applications.

Mobile agents have several states during their entire life-cycle. An agent is instantiated on a host, and goes into what is termed as a *process* state, i.e., the agent executes code, and performs various actions. When an agent decides to migrate, it suspends all its activity and stores its current state in memory. All the data and code is stored, and the agent then migrates to the next host system. Once the agent arrives at the new host, it resumes its process state, and continues execution.

Several models of mobile agent structures have been considered, which depend on the primary aim of the mobile agent - there exist security models, communication models, and computational models. However, each of these models is not stand-alone, but functions in conjunction with other models. A basic mobile agent structure is depicted in Figure 1.5, which emphasizes that mobile agents must possess a computational core, upon which rest the other models. Thus, each agent relies on a basic computational model, on which may lie other features such as communication, security and navigation. Each model is dependent on the model beneath it, and requires the support of the underlying model to achieve its purpose.

The modeling of autonomous agents [116] is dependent on several factors. The

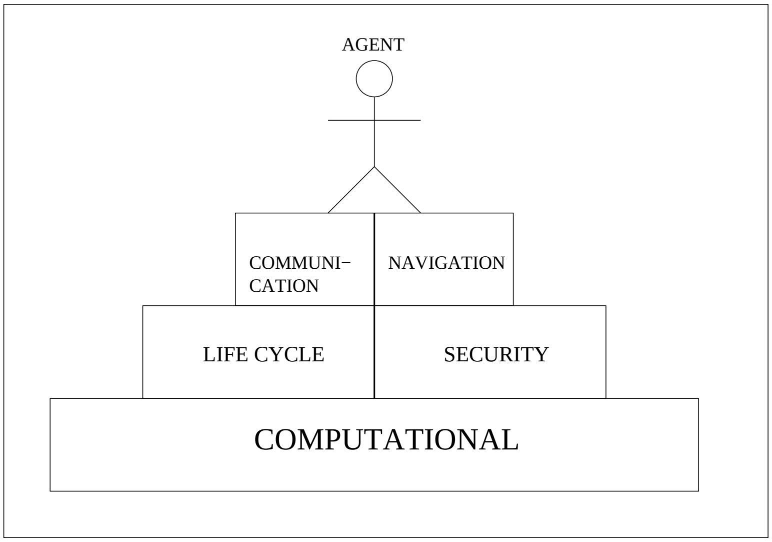


Figure 1.5: Basic Structure of a Mobile Agent, adapted from the TNET Review on Software Agents [84]

agent is viewed in perspective with respect to the whole environment [27], and an integrated system is built rather than separate independent modules. Each module of the integrated system is functionally complete [26] and is used to perform its specific functions. It is reasonable therefore to assume that the environment that the agent is situated in plays a role in developing the appropriate modules.

Mobile agents are especially invaluable in the real-time computing environments of today, where delays are unacceptable and bandwidth is extremely prized. Such environments further require instantaneous action and even faster results. Mobile agents are suited to perform functions in these environments, as they can dynamically determine various target locations for information and processing, carry out their tasks intelligently and with ease of flexibility, and return with the results. The distributed and dynamic nature of their task-performing actions is what is required today.

Agent Life Cycles

Agents undergo the biological life cycles of transitions between *life* and *death*. An agent is *born*, or created on a host or in a system, *lives* through its task/action phase, and finally *dies*, or is terminated. Agents are also capable of reproducing by *cloning* themselves.

The creation of agents requires consideration of the environment and the appropriate agent model. In addition, the tasks that agents are required to carry out may sometimes play an important part in the creation of (more) agents. *Cloning* of agents refers to the creation of new agents in the system by the already existing agents, in order to distribute and parallelize the tasks to be performed. The decision on whether or not an agent should clone is taken based on a number of factors, such as the existing population of agents, the number of tasks to be performed, the frequency of agent-visitation at a specific node, etc.

The new or *child* agents created are exactly the same as the *parent* agents. However, once deployed, the parent and child agents follow different itineraries to optimize the goal. It is extremely important to control the cloning of agents to prevent a population explosion of agents in the system [46]. Different control mechanisms are used to determine the requirement to clone. One example of a cloning control mechanism is *deterministic cloning*, in which mobile agents are created whenever there is new information available in the system. This extra information is viewed as justification for a new agent to be brought into the system for the purpose of processing the new data.

Agents should be able to dynamically control their populations, in view of changes in the system. Once the spate of information input into the system subsides, agents that have been cloned to manage the extra information so far will soon be redundant. *Merging* is a technique that is employed to effectively terminate agents and reduce their population in the system. The simple notion behind merging is that when two agents meet at one location, one of them is redundant. Therefore, the agents each exchange information, and one of them is terminated or killed. It is vital that the agents exchange information before the merging process as otherwise the information carried by the to-be-terminated agent will be lost forever when the agent ceases to exist.

Agents are completely terminated in the system after a period of execution. This is brought about by persistent merging, as the system converges to a single agent. In addition, as the number of agents in the system decreases, the cloning and migration of agents is accordingly reduced, thereby further limiting the number of agents in the system. The system finally has only one agent left with all the information, and this agent is terminated after processing the information that it carries.

Agent Migration

The migration of mobile agents is initiated by the agents themselves; the agents are encapsulated programs consisting of the required data and code. Mobile agents exist in what is termed a mobile agent system, or the environment in which they can migrate and be executed in. The hosts where the agents are deployed need to have the requisite platforms or software in order for the agents to be downloaded and executed on them. The first complete mobile agent system was developed at General Magic with the Telescript®[4] code in 1996 [206].

The mobile agent environment (see Figure 1.6) is required to be made available

[4] General Magic Inc., June 4, 1996, Corporation CA

on all the hosts in the network that the agents intend to migrate to. A conducive environment must consist of all the modules required by the mobile agent to download and begin executing. In addition to these modules, the environment may also provide support services to the agents with respect to the services on the hosts, access to other mobile agent systems and also access to non-mobile-agent-based services [84].

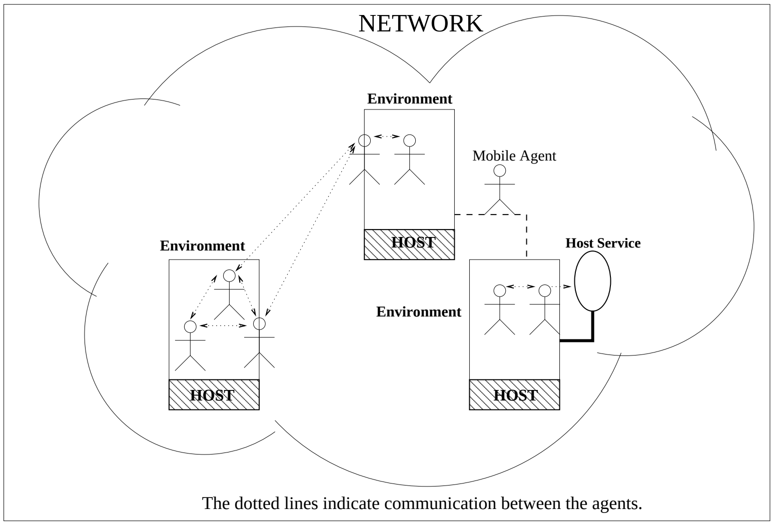


Figure 1.6: Basic Architecture of a Mobile Agent, adapted from the TNET Review on Software Agents [84]

The environment for the execution of mobile agents is provided by the host system. The mobile agents can access services by communicating amongst themselves, or with the host services. Communication between mobile agents can take place on the same host, or on different hosts. The agents can traverse links to go from one host system to another and begin their process execution and communication only if the environment on that host is conducive to the agent. There are several types of agents, and accordingly a proportional number of types of agent platforms. The next section explains in brief the different types of agent platforms that are availabe today, and the basic requirements of each of these platforms.

Agent Platforms and Architectures

Given the heterogeneity of networks today, agent platforms have to be developed so that the agents have the appropriate and requisite environment to execute in. There also needs to be some amount of standardization so that different mobile agents can cooperate and communicate, and be executed on different host systems. The Object Management Group®[5] (OMG) has laid down rules for the interoperability between various mobile agent platforms [130]. The Foundation for Intelligent Physical Agents (FIPA) [66], on the other hand, recommends standards for the capabilities implemented in intelligent agents.

Agent platforms need to have some common basic requirements for mobile agents to be supported [25]. The means for the creation, management and termination of agents is a fundamental requirement for every agent platform. There needs to be a

[5] Object Management Group Inc., Spetember 29, 1992, Non Profit Corporation, Delaware, MA

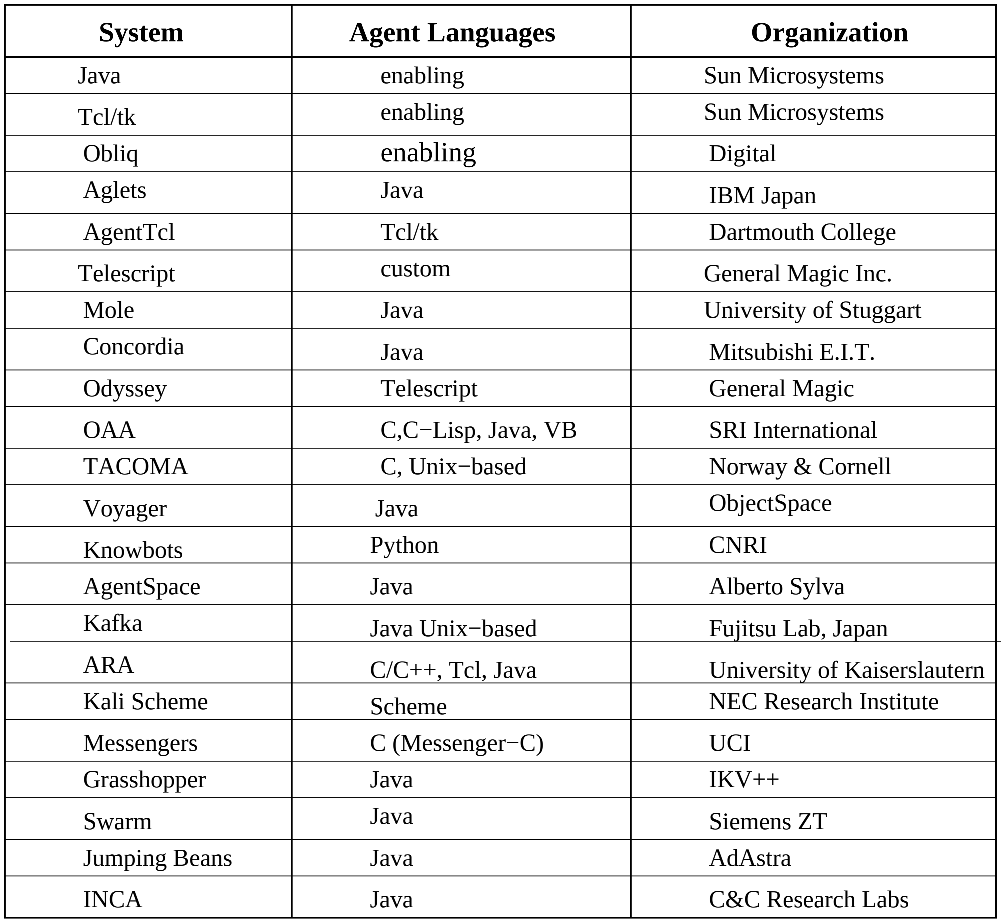
minimal security feature to ensure that the mobile agents are safe from attacks, and also to protect the host itself from rogue agents. Lastly, the platform of course has to provide transport mechanisms to facilitate the migration of the mobile agents from one host to another.

There are already several agent platforms that have been developed and are in use today. These mobile agent systems are developed by various organizations using different languages like Java®[6], Python®[7], Scheme, etc. Table 1.2 gives an overview of some of the popular mobile agent systems that have been developed so far.

The agent systems described in Table 1.2 differ greatly from one another in language and architecture, thereby making it very difficult for any one of these systems to become popular and to proliferate in the computing world. The Mobile Agent System Interoperability Facility (MASIF) attempts to correct this deficit by providing common definitions and interfaces for the different mobile agent systems. An agent system is identified by a unique name and address. An authority that can identify the user for which the agent system acts is associated with it. More than one agent system can be present on a host, and the profile of an agent is described in the agent system type [130].

[6] Sun Microsystems, November 3, 1999, Corporation Delaware, CA

[7] Corporation for National Research Initiatives, May 9, 2000, Corporation DC, VA



System	Agent Languages	Organization
Java	enabling	Sun Microsystems
Tcl/tk	enabling	Sun Microsystems
Obliq	enabling	Digital
Aglets	Java	IBM Japan
AgentTcl	Tcl/tk	Dartmouth College
Telescript	custom	General Magic Inc.
Mole	Java	University of Stuggart
Concordia	Java	Mitsubishi E.I.T.
Odyssey	Telescript	General Magic
OAA	C,C−Lisp, Java, VB	SRI International
TACOMA	C, Unix−based	Norway & Cornell
Voyager	Java	ObjectSpace
Knowbots	Python	CNRI
AgentSpace	Java	Alberto Sylva
Kafka	Java Unix−based	Fujitsu Lab, Japan
ARA	C/C++, Tcl, Java	University of Kaiserslautern
Kali Scheme	Scheme	NEC Research Institute
Messengers	C (Messenger−C)	UCI
Grasshopper	Java	IKV++
Swarm	Java	Siemens ZT
Jumping Beans	Java	AdAstra
INCA	Java	C&C Research Labs

Table 1.2: Different Mobile Agent Systems developed so far, adapted from the TNET Review on Software Agents [84] and the Mole Home Page [135]

Agent systems or frameworks can be classified as either agent-oriented or host-oriented [112]. The agent-oriented frameworks are distributed and depend on the services provided by other agents in the system. Agents in the host-oriented system are provided facilities by the host itself; the host-oriented system is therefore highly centralized. Each host of an agent system actually provides the interface for services, even if it does not directly provide these services itself.

Agent systems need to provide the means for the storage and retrieval of data in addition to communication and transportation facilities for the mobile agents. Data could be stored locally in a storage pool or at a remote site that all the agents on the system have access to. Security may be enforced by allowing only one agent to access the data stored in the agent system.

Examples of Agent Platforms

The mobile agent platforms given in Table 1.2 are a few platforms that have been developed by various organizations. There are several other mobile agent systems that have been designed, and implemented. A comprehensive list of most of the mobile agent systems that have been developed so far can be found at the Mole Home Page [135].

The TACOMA agent system, developed in the University of Tromsϕ [102] provides operating system support for the implementation and execution of mobile agents. The TACOMA system runs on several operating systems, such as HP-UX®[8], Linux®[9], FreeBSD®[10] and Solaris[11], and can support agents that are written in C, Scheme and Tcl/tk. Data is stored in the TACOMA system in hierarchical data storage units. Agents store data in *folders* and groups of such folders are called *briefcases*. There are also *cabinets* of data located on the agent system. *Server agents* are agents that are local or static on the system, and provide services to the *client agents*. The TACOMA system provides a migration facility, and also ensures adequate security to the agents.

The Mobile Agents Platform (MAP) developed to aid the management of mobile agents is presented by Puliafito et al [153]. In MAP, both the platform and the agents that are supported on it are written in Java. This system consists of different components (see Figure 1.7), the most significant of which is a *Server* object that comprises the other objects in the platform. There may be more than one Server on a host. A Server allows the host node to accept and execute mobile agents. A *Daemon* in the Server creates instances of the incoming agents/messages and passes the instances to the *Context*. The Context in the MAP is a fundamental part of the platform, and is used to create, instantiate, execute and terminate the agents. The

[8] Hewlett-Packard Company, November 2, 1993, Corporation CA, CA

[9] Linux Technology Limited, December 10, 1999, Taiwan Branch Corporation, Taipei, Taiwan

[10] Walnut Creek CDROM Inc., February 13, 1996, Corporation CA, Concord

[11] Sun Microsystems, June 25, 1991, Corporation Delaware, CA

Context enables communication between agents on the platform by determining the instance of the object passed to it by the Daemon. It conveys instanced messages to the local agents, and provides the necessary environment for agents to be executed, even if the code for the agent is not available on the server. A *NetworkClassLoader* is present in the MAP to load classes for agents to be executed if they do not have their class present on the server. The NetworkClassLoader searches all other MAPs in the entire network, appropriates the required class and stores it in its cache, so that agents can be instanced using the necessary class. A *CodeServer* is an entity in the Context that is instanced each time a class is requested by the NetworkClassLoader. The CodeServer maintains a list of all the classes available in the Context.

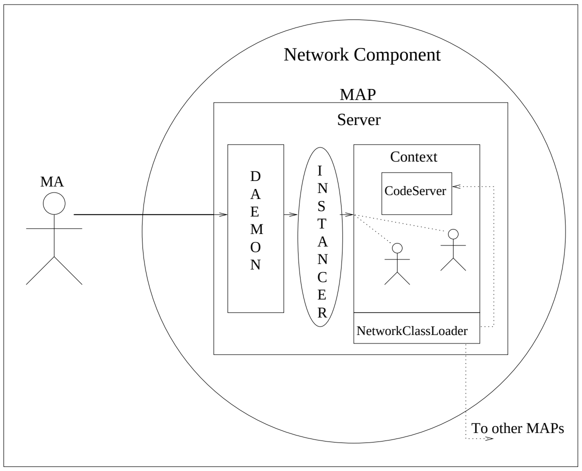


Figure 1.7: The Mobile Agent Platform (MAP) Architecture, adapted from [153]

The MAP presents not just the structure of a Mobile Agent Platform implemented specifically by the researchers who developed it; rather it is a general idea of the requirements of any mobile agent platform. The details of the Server and the Context are necessary to any platform on a host node in a network that supports mobile agents.

JAMES [177] is yet another platform for mobile agents on network hosts that has been developed by research scientists at the University of Coimbra, in collaboration with Siemens. JAMES has been developed in Java, and provides a platform for mobile agents to perform network management functions in particular. The JAMES platform attempts to facilitate flexible code distribution, easy upgrading, disconnected operations, fault tolerance and provide mechanisms for resource control and CORBA®[12] [52] support.

The JAMES architecture consists of a JAMES Manager that is executed in the central host, and several JAMES Agencies that are present in each of the network host nodes. The JAMES agencies are executed on a Java Virtual Machine that is located on each network host. The mobile agents executing on the JAMES agencies conduct their tasks and report the results back to the JAMES Manager on the managing station or central host (more on this function of agents is explained in Chapter 2).

[12] Object Management Group Inc., April 7, 1998, Non Profit Corporation Delaware, MA

Aglets®[13] [202], Concordia®[14] [39], Voyager®[15] [204], Swarm [108] and other Java-based agent systems have been compared and contrasted in the paper, The Performance of Mobile Agent Platforms [179]. The Intelligent Network Control Architecture (INCA) [143] is an architecture developed for network control by mobile agents. Each mobile agent platform developed so far has been implemented with the objective of providing a mobile agent application the necessary conducive environment to execute in. Organizations have begun to focus on the standardization of agent platforms to enable popular use in networks today. Furthermore, most of the agent systems developed use Java as it is deemed the most suited for portable, object oriented code, and provides a number of modules that support mobile agents.

In addition to the environment and the defined models, mobile agents require a common execution language and a communication mechanism to conduct their exchange of messages. The persistence of their processes and states is mandatory, as otherwise all information will be lost when the agent migrates. Lastly, security needs to be ensured as the agents traverse various networks, and could be attacked by malicious entities and corrupted [157] (this issue has been briefly examined in Chapter 5).

[13] International Business Machines Coprporation, June 29, 1999, Corporation NY

[14] Mitsubishi Electric Information Technology Center America Inc., May 25, 1999, Corporation Delaware, MA

[15] Recursion Software, www.recursionsw.com

Agent Languages

Agents are written in different object-oriented programming languages. An agent can only execute code and process data in the same language that it is written in. Therefore, in order to be able to execute their code, agents also need the necessary platform to support the language that they are written in. Several agent languages have been developed by various researchers. These languages are either adapted from existing programming languages that support object oriented programming, for example, Java and C++, or are entirely written for the specific agent model in mind, for example, Telescript.

Mobile Code Languages (MCLs) focus on the creation of mobile code applications, and provide migration facilities to the mobile agents. The execution environment is an important factor in the development of these MCLs, as the migration of code and data computations needs to be supported [55]. In addition to the common mobile agent programming languages like Java, Telescript, Tcl/tk [149], there are other languages like Obliq [33], M0 [197], Tycoon [123], Facile [193] and Sumatra [1].

Pieces of executable code and their corresponding object code are bound to only one location in conventional programming languages. The concept of MCLs changes this aspect of executing code. The code, data and the execution state can be transferred from one place to another, if an MCL is used to program the mobile agent.

The MCL also provides for the linking of code and data to the agents on a host.

These basic characteristics of MCLs have underlined the development of a majority of the agent-based languages used today. The mobile agents, using the object oriented code paradigm, also lend themselves to applications in system programming, agent-based operating systems and communication networks.

Agents in Systems

The fundamental necessity of reliable operating systems for various computing services is a recognized fact. Computer systems today consist of processors, memory, network interfaces, and other input/output devices [190]. The Operating System (OS) of a computer needs to manage computing resources (processing power), schedule tasks, and also maintain information in its memory (file management), as required by the programs that execute on the computer. This is a complex procedure, and various operating systems employ different techniques to achieve these objectives.

Owing to the extremely complex nature of operating systems and the necessity to maintain a smoothly functioning OS at all times, there have not been many radical changes proposed to the underlying techniques employed in Operating Systems. However, intrepid researchers have experimented with the concept of deploying intelligent agents to perform OS tasks. Interface agents [103] have been developed to provide the interface between the operating system and the application executing on it. More recently, Compaq®[16] has introduced *Compaq Management Agents* [51] that play the pivotal role in intelligent management. To quote their objective in developing such agents,

"Compaq Management Agents provide the instrumentation that enables fault, performance, and configuration management. Over 1000 parameters are monitored by Compaq Management Agents. The Management Agents provide the overall system status and direct access to in-depth subsystem information. The Agents help customers manage Compaq servers by monitoring processor status, internal temperature, fan status, etc. They also alert IT managers via screen-based and pager messages when there are problems. The Agents can help prevent problems before they have a significant impact on the system. Predictive fault management on Compaq ProLiant®[17] servers significantly reduces server down time."

The Compaq Management Agents can be extended to perform management functions on any operating system. Significant work has been carried out in the development of strategies to ease operating system tasks. With the advent of distributed operating systems these days, the deployment of agents to carry out operating system tasks has almost been considered a foregone conclusion.

[16] Compaq Computer Corporation, January 6, 1998, Houston, TX

[17] Compaq Computer Corporation, January 31, 1995, Houston, TX

Agents in Distributed Systems

Modern computing systems comprise a network of several processors and are known as distributed systems. Distributed systems have become the norm due to their reliability, efficiency, speed, scalability and inherent distributed nature that promotes flexibility in operations, communications and data/device sharing [190]. The growth of industry and the subsequent increasing reliance on computing systems has led to the prevalence of distributed computing.

The distributed nature of processing information is common to both distributed systems and to mobile agents. The ability of mobile agents to intelligently and autonomously carry out tasks, and to coordinate these tasks is invaluable in distributed systems. Considerable processing of information is delegated to mobile agents in the system, thereby promoting efficient time and task management. Intelligent mobile agents have been used for the concurrent execution of many tasks in distributed systems [14]. Agent-based operating systems for mobile networks have been developed to enable the "ubiquitous access to information and applications" [43]. The mobile agent paradigm has been extended to function in distributed systems without any dependence on the underlying technology [38].

Distributed systems suffer from some inherent problems, such as security and the lack of software for distributed access. The pressing concern however, is the management of the network in a distributed system. The communication network upon which the distributed system is built may become congested, lose data, or temporarily lose connectivity. These problems need to be addressed and feasible solutions to them are required for the consistent operations of the distributed system.

Agents in Networks

Mobile agents have been used in networks to promote problem-free distributed system operations. Distributed intelligent agents [187] can retrieve and process the required information, and coordinate with other agents to anticipate and perform tasks in the network. Significant areas of interest in networking research are the management of resources such as bandwidth, the accomplishment of efficient routing between the nodes in a network, and the provision of some fault-tolerance and security in the network to guard against errors and attacks respectively. The management of the network itself is a function that requires monitoring of each part of the network and the awareness of complete network topology.

This thesis explores the use of mobile agents in performing network system operations in each of the areas mentioned above. Chapter 2 is an essay to bring to light the role of mobile agents in network management. The management of resources in the network with the aid of mobile agents has been discussed in Chapter 3. Chapter 4 focuses on the routing issue in networks by deploying mobile agents as the principle path-finders. The discovery of and recovery from faults in the network and in the agent models themselves using intelligent mobile agents is covered in part in Chapter 5. This chapter also deals with the overriding concern of protecting the network and the agents from malicious attacks. A summary of research conducted in each of these areas, and a short justification for the prevalence of the mobile agent paradigm in networks today is presented in Chapter 6.

CHAPTER 2

NETWORK MANAGEMENT USING MOBILE AGENTS

Introduction

Networks are getting increasingly complex in terms of the functionalities and services offered and large in terms of the number of nodes in them. There are also a greater number of users today than there were a couple of decades ago. These changes lead to vast amounts of data on the network, and an increased level of challenges regarding the provision of services, the quality of these services, reliability and security. Network management issues such as the management of data and information of nodes in the network need to be considered in a whole new perspective, given the heterogeneity of networks [15].

Simple Network Management Protocol (SNMP) and Common Management Information Protocol (CMIP) are centralized network management protocols used in networks today. These protocols adhere to rigid hierarchical structures in the network and require considerable data transfers that may result in potential bottlenecks in the already resource-constrained networks. SNMP and CMIP need to be modified to be used in present day networks as they offer low flexibility and cannot be easily re-configured on the fly, both of which are requirements in the increasingly heterogeneous and rapidly growing networks today. The physical resources of the network are managed by a central network management base. Managing these resources leads to a drain on the already strained resources of the network. Collection of data from all the elements in the network, analyzing the data, and ensuring smooth operations is not an easy task, and even less so when the network is heavily loaded. Paradoxically, it is when the network is congested that network management is most required. The flow of information required for network management under such circumstances in a centralized approach leads to further congestion in the network [72].

There have been a number of distributed/delegated technologies developed for network management [122]. There has already been an inroad to the delegation of network management tasks with the use of Remote MONitoring (RMON) [160], and Management by Delegation (MbD) [78, 124]. The mobile agent paradigm has been explored and found to offer a viable solution to network management issues.

Network Management Technologies

SNMP was developed in the 1980s. Network management in SNMP is achieved by sending Protocol Data Units (PDU) to different parts of the network to collect data, and then aggregating and analyzing this data. The SNMP compliant devices store data in Management Information Bases (MIB) and return this information to the SNMP manager. Since network management perforce is required to be both ubiquitous and continuous, SNMP is increasingly becoming the source of latency and bandwidth problems in today's busy networks. The main disadvantage of SNMP is that it consumes resources for network management that could otherwise be used in servicing network entities [183, 184].

CMIP was developed to be an improvement over SNMP, but it proved to consume more system resources than SNMP, and therefore never got past the development and research stages to be used as widely as SNMP [48].

The Internet Engineering Task Force (IETF) has introduced the concept of RMON to decentralize network management. The network is divided into segments, each of which is monitored by RMON probes or monitoring devices. The RMON probes collect information from the network segments they are located in, and thus collectively provide an overall view of the network. RMON provides network managers with a comprehensive view of the network elements, thereby limiting the amount of information that is required to be processed by the managers. There are different groups of RMON devices, each providing specific information to aid common network monitoring requirements. Each RMON device collects data from its segment, and transmits it back to the RMON console or manager [160]. This method is therefore suited to a distributed management scheme in the network.

Management by Delegation (MbD) is another step in research to provide decentralized network management. This involves the downloading of code/scripts to remote locations and executing them there to provide local monitoring or management [78, 124]. This approach can be viewed as a direct precursor to the deployment of mobile agents to aid network management.

Mobile agents are now deemed extremely affordable and suitable solutions to the network management problem. The migration of code for remote execution and the transmission of only the results and not the entire data to the network manager is the overwhelming advantage with mobile agents. Mobile agents can be delegated tasks and are capable of filtering and processing data locally without having to be controlled centrally. However, mobile agents can be expensive if they are required to possess high intelligence and processing power - the increase in their size would result in the agents occupying as much bandwidth as SNMP probes! These considerations have led to researchers devoting significant time and effort in developing efficient agent-based network management strategies.

Mobile Agents in Network Management

The following is a summary of selected papers that address the various issues related to network management using the mobile agent paradigm. These papers are presented sequentially in chronological order, and will help to elaborate on the various ways of using mobile agents in network management.

Management Views by Delegated Agents

Resource: Network Management Views Using Delegated Agents, by German Goldszmidt, Published in the Proceedings of the 6th CAS Conference, Toronto, Canada, Nov 1996 [77].

"This paper describes the design of a system to support MIB computation at the networked devices." The concept of MbD is extended to compute external views of MIBs to transmit to the network manager. Since retrieval of data from the network devices results in inconsistent non-atomic data, there has been an attempt to perform MIB computation at the network devices itself. A View Definition Language (VDL) has been introduced to specify computations over MIBs. SNMP agent extensions that implement the MIB external views specified by the VDL are used to get atomic snapshots of the MIBs. This implementation can be used to filter MIB data, to join MIB tables and to detect attempts at intrusion.

Network management involves the detection of problems in the network. This is traditionally accomplished by retrieving all the relevant information from the network devices and then checking all this information for flawed data values (see Figure 2.1).

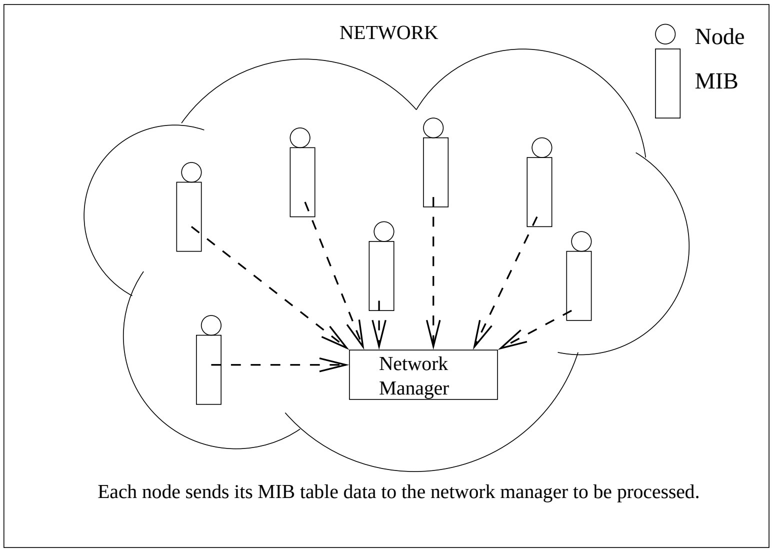
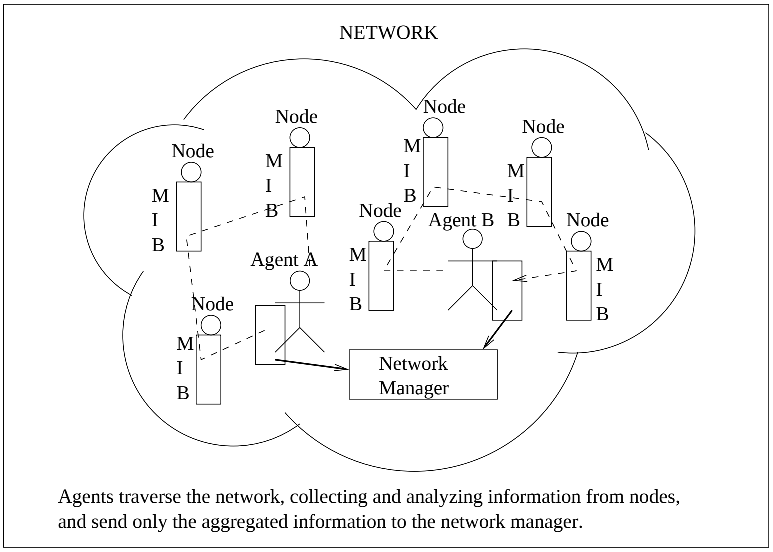

NETWORK
Node
MIB
Network Manager
Each node sends its MIB table data to the network manager to be processed.

Figure 2.1: Traditional Network Management: Nodes in a Network Sending the MIB table data to the Network Manager

A more efficient way to do this would be to perform a filtering operation at the network devices and retrieve only the data that indicate problems. Retrieving information that is scattered across different MIBs requires transfer of all the tables of data in the various MIBs to the manager. This is extremely expensive in terms of the resources and bandwidth utilized, and inefficient too. Instead, data can be collected from the various MIBs and only the relevant information can then be sent (see Figure 2.2). Applications in the current traditional SNMP paradigm do not have any method of accessing and reusing data that has already been retrieved, filtered and computed. There is no external level data repository which can be accessed. This leads to



Agents traverse the network, collecting and analyzing information from nodes, and send only the aggregated information to the network manager.

Figure 2.2: Agent-based Network Management: Agents create a database view of the relevant MIB table data to send to the Network Manager

repetitive and redundant retrieval and re-computation of data.

"A database view is a single table derived from other tables. SNMP management applications must receive raw MIB data and recreate an appropriate data model at the central host." As this method is inefficient and expensive, MbD extensions to the SNMP agents have been used with the VDL (data specification language) for computations over MIBs. Thus, "network management engineers can create external level data models at the managed node side."

This is one of the earlier papers published on using the mobile agent paradigm in network management. The mobile agents have not been used exclusively as a system on their own, but rather as an extension to the MbD strategy. This is a database approach to filter the data that needs to be managed in the network components. This approach uses both MbD and SNMP to accomplish network management. A VDL is defined and used with MbD to compute external MIB views that can then be queried by SNMP-able nodes. The paper does not completely focus on mobile agents, but is worth discussing as it is the first foray into the use of an external piece of code to perform computations and management functions at the network components, instead of having to transfer all the data to the manager. This paper elaborates on the concept of aggregating MIB views for network management presented in [78].

Network Monitoring and its Scalability using Mobile Agents

Resource: On the Performance and Scalability of Decentralized Monitoring using Mobile Agents, Antonio Liotta, Graham Knight, George Pavlou, 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM '99 [113].

Network monitoring is mainly achieved by polling the nodes of the network, which is a centralized action. Large amounts of data need to be transmitted to the management stations. The magnitude of data to be polled poses various limitations such as network congestion, delays, poor performance, and delayed reactions to faults in the network. Mobile agents have been introduced as an effective alternative solution to this problem, as they are a decentralized and efficient method to perform network monitoring services. Thus, data can be monitored and processed at the remote network hosts, before being actually transmitted to the network monitoring station, thereby minimizing the delays and traffic in the network. Further, agents can clone themselves [46] to perform the monitoring, and decrease the deployment time and traffic in the network significantly. However, mobile agent systems are complex and therefore need to be designed and implemented only where their performance far outweighs the cost of using them. This paper compares the performance and scalability of mobile agent systems in network monitoring, with that of traditional polling, and optimal polling systems.

Three mobile agent system models are used - the first where each mobile agent periodically polls the network components and analyzes the results, the second wherein each mobile agent periodically polls the network components, analyzes the results and also informs the other mobile agents or the management station, and the third, where the mobile agents generate data only if a certain event has been detected (event-/alarm-driven). The mobile agent systems are organized according to the method of deployment of the mobile agents. The mobile agents could be organized in a flat (no) hierarchy with or without cloning, or in a hierarchical structure, again with or

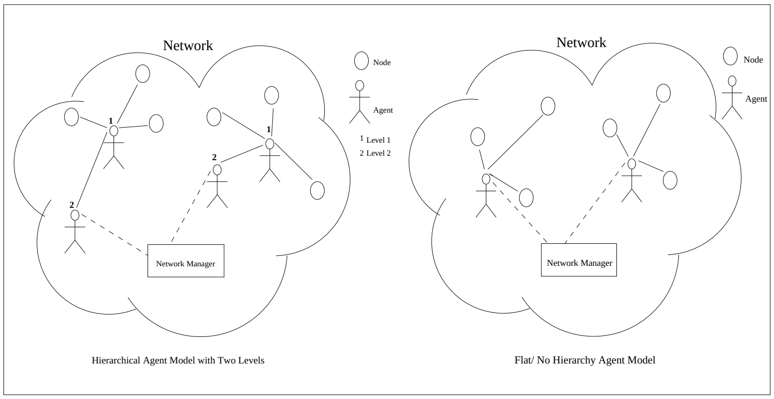


Figure 2.3: Flat and Hierarchical Agent Models

without cloning capabilities. In a flat model, there is no inter-agent communication, and agents monitor the network components and report the results directly to the network manager. In a hierarchical model, agents are present on different levels, and only agents on the first level communicate with the network manager. The agents that perform the monitoring function at the network components can communicate with agents directly above them, and these agents in turn, can communicate with agents on the level above them in the hierarchy (see Figure 2.3). The main issues with the mobile agent systems are the determination of the number of agents to be deployed, and where to deploy them to (location determination).

The metrics for measuring the performance of these systems are the traffic in the network due to monitoring, and the associated delay. Equations were derived for the traffic generated and the monitoring delay respectively for each of the systems - *naive* *centralized polling, optimal centralized polling*, and *mobile agent based monitoring.* Naive centralized polling is when the polling station queries every node in the network. Optimal centralized polling is the process where only the neighbors of each node get polled; therefore the polling station polls its neighboring nodes, each of these nodes poll only their neighbors and so on. Mobile agent based monitoring can be further classified into different models. The different models of agent-based systems produced varying results for deployment traffic and monitoring delays. For example, the cloning models (hierarchical and flat) showed a reduction in deployment traffic (the traffic due to the deployment of agents), whereas the event-based model reduced the delivery traffic of data to the management stations significantly. However, using mobile agent systems is not always the better alternative, as there may actually be an increase in traffic due to the deployment of a sufficiently large number of agents to collect data. Polling was found to be more effective when the polling rate and the duration of the monitoring task were short.

The delay associated with monitoring the network for agent based systems always proved to be less than the polling solutions to network monitoring. This delay does not increase even with an increased network size. In fact, it was actually found to decrease. Cloning of agents can reduce the deployment time of mobile agents, and more sophisticated deployment techniques may be used to reduce the deployment traffic too. Typically, systems with a large number of mobile agents perform better. This improved performance is offset by the increased deployment time, and can be compared with the 'flat, non-hierarchical with cloning' agent model, that has less deployment overhead.

The paper presents an analytical view of the advantages of using mobile agents to monitor networks. It compares various methods of monitoring information in the network, both in the traditional manner, and by using mobile agents. The main parameters under consideration are the traffic and delay in the collection and aggregation of information using mobile agents, as compared to polling in SNMP. An analytical model has been put forth to support the use of mobile agents, and the equations derived herein depict a significant reduction in the information gathering delay and also fewer traffic bottlenecks. The deployment time (time to deploy agents in the network) of mobile agents is reduced by cloning agents to cover large networks. Therefore, the mobile agent system was found to be scalable and more efficient than traditional SNMP. This publication covers the monitoring aspect of networks in detail, and is a well-reasoned, methodical paper.

The paper, Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications [9], supports the view presented in the above paper. The various amounts of traffic in the network generated due to network management in different models including a model using the mobile agent paradigm, are compared. Traditional client-server models, a remote evaluation and a code-on-demand model are the other models used for the comparative analysis. The design paradigm is defined in terms of different components (computational or resource) in networks and the interactions between them. The paper emphasizes that the implementation of a mobile agent system for a network would be specific to the type of network it is implemented in and to the type of management function that is required of the agents for it to be efficient.

Advanced Network Monitoring using Mobile Agents

Resource: Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology, Damianos Gavalas, Dominic Greenwood, Mohammed Ghanbari, Mike O'Mahony, Computer Communications Journal, special issue on 'Mobile Agents for Telecommunication Applications', published January 2000 [72].

This paper presents solutions to three network management problems using mobile agents - aggregating several MIB values, obtaining atomic snapshots of SNMP tables, and filtering the contents of the tables. The authors have created an infrastructure for dynamic network management using mobile code. The mobile agents are extremely lightweight, and transfer only collated information. Since the mobile agents roam the network collecting and storing only pertinent information, they are small in size. The management function for the network monitoring station is rendered simple as only the relevant information is conveyed by the mobile agents. The agents collect information based on two polling techniques:

- *Get 'n' Go (GnG)*, wherein the network is partitioned into different domains and a single agent is assigned to each domain to collect information here. The agent sequentially visits every element in the network domain it is assigned to, collects data and returns it to the manager in every Polling Interval (PI).
- *Go 'n' Stay (GnS)*, wherein the mobile object is broadcast to all the network elements and it stays there collecting information for a certain number of Polling Intervals, before returning to the manager.

Intelligent Filtering Applications of Data by Mobile Agents: The following are some of the applications that use the ability of mobile agents to filter data.

▷ Aggregating several MIB values into more meaningful values: When several different MIB variables are required to be aggregated in a Health Function, the traditional SNMP method would require that each of these values is transmitted to the manager, alongwith the Object Identifiers (OIDs). These data typically occupy significant space on the network links, and at the network manager. On the other hand, intelligent mobile agents can be equipped with the functionality of computing the Health Function at the network elements, and storing and transmitting only the final result to the network manager. This is an obvious optimization of the amount of data required to be transmitted, and maintains the small size of the agents also.

▷ SNMP Table Polling: Due to the increase in size of TCP connection and IP routing tables, it is difficult for SNMP to transmit these tables over the network. The retrieval of all the data in one such SNMP table would require a *get-next* operation per row of data in the table. This leads to significant latency problems, and possible inconsistencies in data, as the time-interval between transmissions is not inconsiderable. The *get-bulk* request does not improve the situation much as the maximum parameter for the amount of data to be transmitted may be set to either too small or too large a value to get accurate results. The Mobile Agent solves this problem by migrating to the network elements and collecting all the information using successive get-next requests, encapsulating this data and then returning to the manager. The agent could move to another host to get more data, or collect data at pre-determined intervals before returning to the manager to make this more efficient, and provide up-to-date snapshots of the SNMP tables.

▷ SNMP Table Intelligent Filtering: Most of the data retrieved by the SNMP agents are processed by the manager and filtered to get the status of various network elements. Thus, a significant amount of bandwidth is utilized in the transmission of this data, which ultimately is used only at the manager's end. To prevent this excessive usage of bandwidth, mobile agents could be used to filter the data before transmitting them to the manager. The mobile agents are provided with intelligent filtering operators that filter SNMP tables and keep only those rows and columns that meet certain criteria, previously determined by the manager. The filtering operation may be performed on a pre-defined table column, or on a health function.

Mobile agents may also perform domain or global level filtering, in which data in the network elements can be compared with the data already collected in order to determine the relevance of the new information. This comparison/merging of data can result in reduction of demand for resources to access data in the network (less data to transmit) and less processing on the part of the manager. The mobile agent's size is maintained small as no redundant data is stored and carried.

This paper presents methods to effectively filter data at the network components using mobile agents, and sending only relevant data to the manager. In addition to these methods, the paper presents examples to demonstrate the efficacy of this idea, and graphs to depict the improvement in performance due to reduced bandwidth usage. The results shown to compare the network overhead in polling data using SNMP and the Health Functions presented in this paper show that the use of mobile agents to perform data processing at the network components does indeed improve performance and reduce the load on the network. The authors have effectively shown the advantages of using mobile agents for the filtering of information at the network components, instead of at the network managers, in terms of reduction in traffic and bandwidth usage, as also reduction in the network overhead.

However, it is not always necessary that an agent with high processing capabilities consumes less bandwidth. Providing the agent with the requisite processing capacity increases its volume, making it too bulky to be easily mobile. Moreover, the run-time environment is rendered complex, as it has to be endowed with higher processing capabilities. The design of such a system is difficult to achieve and improvements are still in progress [42].

In 'Using Mobile Agents to Implement Flexible Network Management Strategies' [152], the concept of using mobile agents to gather information from the network components and compute health functions is investigated. There are four different kinds of agents described herein - a *browser* agent that traverses the network and collects the information required, a *daemon* agent that travels to the specific node to compute a specific health function as required, a *messenger* agent that communicates with other agents and collects their computed data, and a *verifier* agent that performs checks on the constituent network components as per some predefined property. This model is quite flexible as various functions can be added to the different types of agents to provide complete management capabilities. No results have been provided however, and the authors have stated the necessity for more complex functions and dynamism in the system.

Mobile Intelligent Agents for Network Management Systems

Resource: Mobile Intelligent Agents for Network Management Systems, George Eleftheriou, 2000 London Communications Symposium [63].

In this paper, a mobile agent system has been developed for the management of networks and their functions. There are different hierarchical levels of mobile agents that coordinate with one another to perform management functions.

The MIAMI [128] project was developed as an agent model for network management. The model consists of a Virtual Market Place (VMP) where consumers can search for services offered by the network, and avail of the same. The VMP comprises a Virtual Enterprise (VE), an Active Virtual Pipe (AVP) and a Connectivity Provider (CP) (see Figure 2.4).

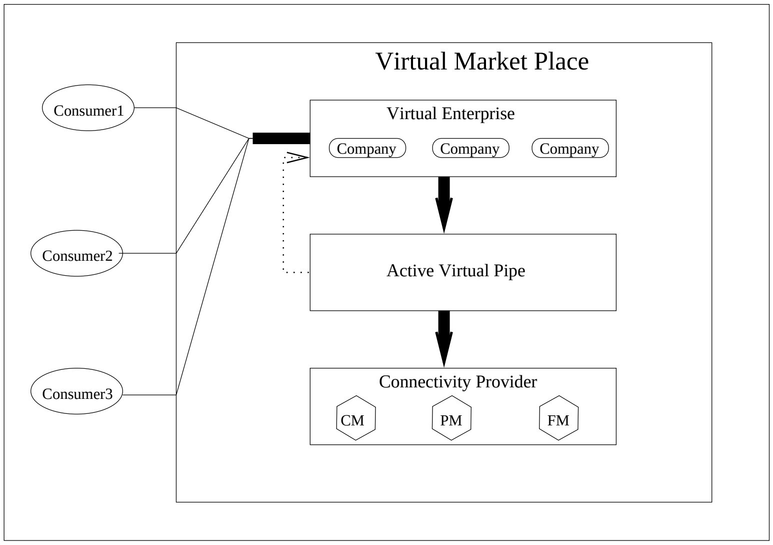


Figure 2.4: The MIAMI Model

The VE is actually a collection of different ‘companies’ that have the necessary resources to achieve the VE’s goal. The VE’s goal is defined by the consumers who use the network and require different services from it.

The AVP provides the connectivity between the VE and its member companies, and also between the consumers and the VE. The AVP is thus the transparent link between the different networks in this model. The AVP provides connectivity to the VE with the necessary Quality of Service (QoS), and can provide connectivity per varying requirements dynamically. The AVP uses the CP to provide connectivity.

The CP comprises the three elements that make providing connectivity and managing the same possible - Configuration Management (CM), Performance Management (PM) and Fault Management (FM). The VE contacts the AVP for a connection, and the AVP in turn contacts the CP. The CP uses the CM component to set up and provide the connection to the VE. It also provides information about the connection that is used by the components in order to facilitate the monitoring of connections and checking for faults in it by the PM and FM respectively.

The Configuration Management (see Figure 2.5) is performed by a Configuration Management Agent (CMA) that resides at network manager level. This agent is static and contains information about the entire network that enables it to provide connectivity when required. Agents are located at each network element to manage the network component, and interact with the CMA to establish or tear down connections. Mobile agents called Bandwidth Brokers (BBs) are agents that migrate in the network, locating and making available bandwidth for use by the CMAs on different network domains. The BBs are capable of communicating with other BBs in the same and different domains of the network. They are intelligent and capable of assessing the requirements of different connection requests, and manage and monitor the same.

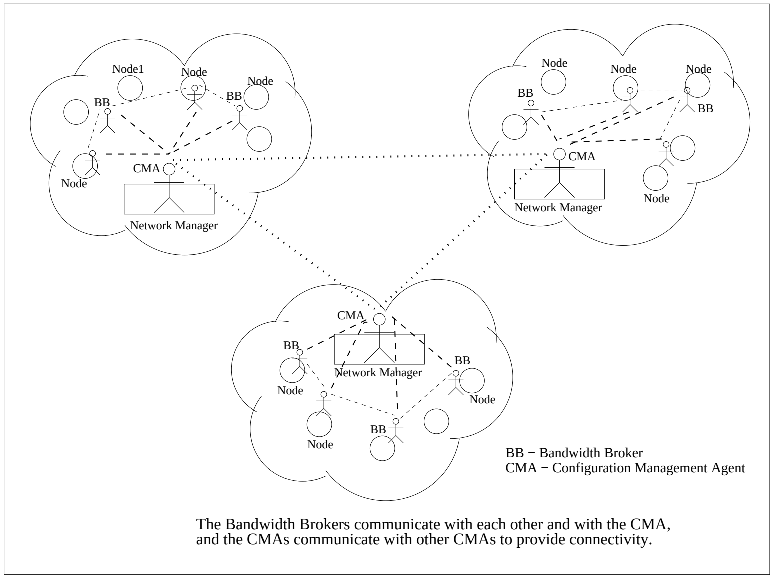

Node1
Node
Node
BB
BB
Node
CMA
Network Manager
Node
Node
Node
BB
BB
CMA
Network Manager
Node
CMA
BB
Network Manager
BB
Node
Node
BB
Node
BB − Bandwidth Broker
CMA − Configuration Management Agent
The Bandwidth Brokers communicate with each other and with the CMA,
and the CMAs communicate with other CMAs to provide connectivity.

Figure 2.5: Configuration Management in the MIAMI Model

This model encompasses an intelligent mobile agent system for network management that is dynamic in nature. The model is scalable and highly decentralized. It considerably reduces the load on the network as the BBs can avail resources in different domains. However, the paper states that the model is expensive in terms of processing power required for the agents and the agent platforms. Further, the ability to be mobile and to communicate with other agents is expensive, causing an unfavorable trade off between the intelligent processing capacity of agents and the cost of deploying such a system.

There is a recognized need to develop this project for the management of IP networks as it has inherent advantages, and to decrease the cost of building and implementing such a model. There are no published results of the performance of this model, and more detailed studies of the various components of this model might help in establishing the practicality of implementing this model in networks today.

Information Gathering for Network Management using Mobile Agents

Resource: A Self-Adaptable Agent System for Efficient Information Gathering, Antonio Liotta, George Pavlou, Graham Knight, Proceedings of the IEEE/ACM International Workshop on Mobile Agents for Telecommunication Applications (MATA '01), August 2001 [114].

This paper uses the mobility of agents in the network to monitor and gather information. The network is partitioned by the mobile agents and these agents clone themselves if required [46], to manage more partitions of the network. Each agent is responsible for information in one specific partition of the network. This paper presents a unique method of relocating the agents that manage the network resources, to adapt to the changing conditions in the network. Thus, agents relocate to optimal locations to minimize traffic and latency in the network. The overhead brought about by migration must be comparable to the reduction in cost, otherwise it would constitute an expensive option.

The optimal number of agents and their locations need to be computed for the monitoring of the network. Further, after the agents have been deployed, they need to autonomously be able to stabilize and control their locations and adapt to the changing conditions in the network. The mobile agents are actually area managers in the networks with knowledge of the routing tables. They are able to autonomously make decisions and roam the network to monitor and gather information.

The monitoring station begins execution with only one mobile agent. This agent computes the routing costs and partitions the network according to the cost. Further, the agent clones itself to create a new agent for each partition. Each agent then migrates to the optimal location (least cost) in each partition. Here, the process repeats, as the agent again calculates the costs and repartitions the network, cloning itself as necessary. The cloning of agents reduces the traffic around the monitoring station, and the agents perform their tasks in parallel after having been cloned. The migration of agents to optimal locations is an ongoing process. Changes in the network leading to loss of links may affect the routing tables, and the dynamic change in routing tables only increases traffic in the network due to the utilization of the alternate routes. In this scenario, the mobile agents display superior performance to the SNMP agents by autonomously computing the cost, and migrating to the location with the least cost. The agents perform active monitoring by periodically sensing the network and estimating the cost of alternate locations.

The monitoring activity is purely localized and restricted to the specific partition that the agent is located in. The agent thus computes cost functions for only the nodes in its partition. While this method is straightforward and reduces the processing overhead, it cannot be applied to global optimization strategies, as the agents only monitor their local network components and do not really exchange this information with one another. This may result in a local optima being insufficient for the global system after a prolonged period of time. In such a situation, the agent system is completely re-deployed and started anew.

Distributed monitoring agents in the network scale well and can be used for network topologies of varying sizes. In all the simulation results, the distributed mobile agent monitoring system has proven to be an optimal solution.

The paper describes the precise way in which agents can be used to partition the network for control and monitoring purposes. Location optimality is computed, and agents are cloned as required to manage different partitions of the network. Results of using this management strategy are presented, and used to prove that the computation of location optimality to migrate the mobile agents using *constrained code mobility* is indeed an efficient solution to network management issues. The scalability of the distributed agent system is evaluated against a centralized system, and it was found to perform significantly better than the conventional centralized model.

Migration overheads were computed and it was found that cloning agents was a cheaper solution to management, than the actual migration of agents from node to node. The number of agents required in the network, the overhead involved with the deployment of the mobile agents, and the traffic and latency are computed and taken into account while presenting the results. The agents in this paper are relatively small in size and therefore do not make the execution unnecessarily complex. The number of agents used is an important consideration as this would directly reflect on agent migration and overhead in terms of computational and memory resources involved. The paper is complete in presenting a comparison of this approach to the conventional centralized management systems.

The aspect of *constrained code mobility*is further evaluated and explained in [23]. An agent is cloned and migrates from one node to the other guided by the *parent* agent. The agent then executes only on the node it migrates to. This *constrained mobility*, according to the authors, lends ease of use and programmability in the nodes, as the functionality of each node can be extended specifically as required by the agents in that partition of the network. Comparisons of the performance of mobile agents in network management on different systems have been presented in terms of traffic usage, memory consumption, overhead incurred for the migration of code and response time. Using these results, the authors propound the argument that *strong code mobility* is deleterious to the performance of network management functions, as there is too much traffic and delay involved. On the other hand, *constrained code mobility* was found to be optimal in terms of performance. There is ongoing work in the rendering of a coexistence between static and mobile code to further facilitate network management.

Performance and Scalability of Mobile Agents based Network Management

Resource: Scalability of a Network Management Application Based on Mobile Agents, Marcelo G. Rubinstein, Otto Carlos M. Duarte, Guy Pujolle, March 2002 [148].

Resource: Evaluating the Performance of a Network Management Application Based on Mobile Agents, by Marcelo G. Rubinstein, Otto Carlos M. B. Duarte, Guy Pujolle, Networking 2002, to be published in Lecture Notes in Computer Science, May 2002 [162].

These papers compare two different methods of gathering MIB-II (second version of MIB for TCP/IP networks) [129] variables on network managed elements - the first is the traditional SNMP method, and the second is using mobile agents. In the traditional method, the network manager node sends a request to an SNMP agent on a network element, and waits for the response. After it receives the response, the manager sends the request to another SNMP agent on another network element, repeating this process until it sends requests and receives responses from all the elements in the network. This obviously involves significant communication overhead and expense in terms of bandwidth. In the mobile agent scenario presented in this paper, there are three different types of mobile agents used - a mobile agent that migrates to the network element, an SNMP agent that is resident on the network element, and a translator agent that is used to communicate between the mobile agent and the SNMP agent. The mobile agent migrates to the network element and communicates with the translator agent using RPC. The translator agent then sends out a request message to the SNMP agent (*GetRequest PDU*) and waits for the response. Once the response is received from the SNMP agent, the translator communicates the same to the mobile agent, and the mobile agent proceeds to the next network element to repeat this process (see Figure 2.6). The mobile agent returns to the network manager only after it has received all the data from all the network elements.

The performance of the two methods was compared on the response time in retrieving a specific MIB-II variable from the network elements to be managed. The mobile agent method of collecting MIB-II variables was found to result in a higher

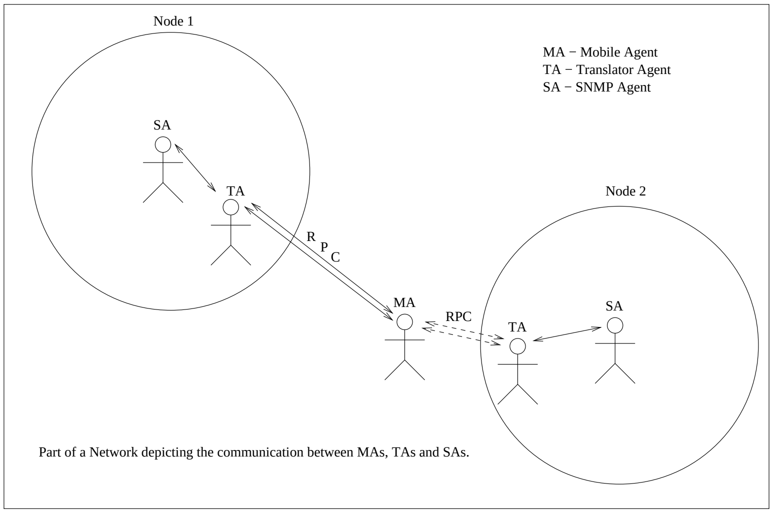


Figure 2.6: Gathering MIB-II Variables Using Mobile Agents

response time than the SNMP version, as the size of the mobile agent increased with each network element it visited. Also, the time to access each MIB-II variable on each of the managed network elements was found to be greater in the case of mobile agents. Further, in the simulation model, SNMP messages are not fragmented as the size of these messages are quite small. However, the agent's size quickly exceeds the Maximum Segmentation Size (MSS) and the agent then has to be fragmented and sent across the network in different packets, damaging the performance. However, it was observed that while SNMP performed better than the mobile agent scheme for a small number of elements, by extrapolation, mobile agents perform better than SNMP when the number of network elements to be managed ranges between an inferior and superior limit. This limit is defined by the number of messages that pass through the backbone of the network and the size of the mobile agent, as it grows collecting information from each element on the network. A mobile agent performs better than SNMP over bottleneck links in the network, i.e., it was found to utilize fewer resource in the resource constrained areas of the network than the SNMP agents.

In an effort to optimize the performance of the mobile agent paradigm in this case, an attempt was made to set an upper bound on the number of nodes to be visited by the mobile agent. Having visited the set number of nodes, the mobile agent would return to the manager to *unload* its data, and then continue to the other network elements to collect the remaining data. It was found that this method of determining an upper bound of nodes to be visited worked better than not having a limit. The fewer network elements visited by the mobile agents per trip, the less response time it took for the mobile agents. Another strategy implemented to improve the performance of the mobile agents was to send the data to the network manager instead of having the mobile agent return to the manager each time. Here too, the response time decreased sharply when the number of network elements from which data was to be collected and sent to the manager was small.

The papers are an unbiased view on the performance of mobile agents in network management functions, and unflinchingly present the inherent deficits in a mobile agent system. However, the authors do propose strategies to improve performance, and believe that constant technological advances and research will render the mobile agent systems ultimately more efficient. The delayed response time and fragmented data as a consequence of using mobile agent systems is offset by the improvement in performance in terms of a decentralized, efficient approach. Moreover, the response time of the agents can be reduced by incorporating certain procedures, such as limiting the number of nodes visited, or sending only the data to the manager. This indicates that the mobile agent systems need more work and improvement to be completely reliable and to justify their use in networks. There is an assumption of the interoperability of the mobile agent systems and the SNMP-agents on the network components in this system.

The paper *Evaluating Tradeoffs of Mobile Agents in Network Management* [163] by the same authors, further describes the situations under which the mobile agent systems perform better than the conventional network management systems. Owing to the growing size of the mobile agents as they travel gathering data, the use of this system is justified only if there are many nodes in the network. Moreover, mobile agents can effectively handle bottlenecks in networks as they do not traverse the congested links very often, rather they trace alternate routes to various network elements before returning to the management station.

Mobile Agents for the Management of Programmable Networks

Resource: Programmable Networks and their Management using Mobile Code, Andrzej Bieszczad, Bernard Pagurek, Tony White, Cheryl Schramm, Gatot Susilo, unpublished [16].

This paper presents a model for the management of network elements using mobile agents. The paper introduces three different types of agents - *netlets*, *servlets* and *deglets*. In addition, there is a fourth type of agent that is referred to - the *piglet*. *Netlets* are mobile agents that provide a predefined functionality on a permanent basis, while *deglets* are delegated a specific task for a period of time. *Servlets* are static pieces of code that provide extensions to the servers. The *piglets* are a network security concern, and could be malicious pieces of code.

The infrastructure to design the management of networks using mobile agents was designed and implemented using Java, and included the following components:

- A Mobile Code Daemon
- A Migration Facility
- An Interface to Managed Resources
- A Communication Facility

- A Security Facility

The Mobile Code Daemon (MCD) is the facility that provides for the transportation of the mobile code. It is a thread that executes inside the Java Virtual Machine (JVM) and listens in on ports for incoming messages. The MCD provides a number of services to facilitate the execution of mobile code :

▷ Security Services

▷ Bytecode Management

▷ Mobile Code Instantiation

▷ Providing an Interface to Managed Resources

▷ Migration Facilities

Upon receiving the bytecode for a mobile agent, the MCD validates the required Java classes, loads them and instantiates the objects for the agent. Figure 2.7 depicts a magnified view of a Network Component, the MCD and the interaction between each of these components and the mobile agent.

The Migration Facility (MF) determines the destination of the mobile agents, and also the migration strategy to be used. The netlets can put forward a request to migrate, and the MF then decides on the appropriate strategy to send the mobile

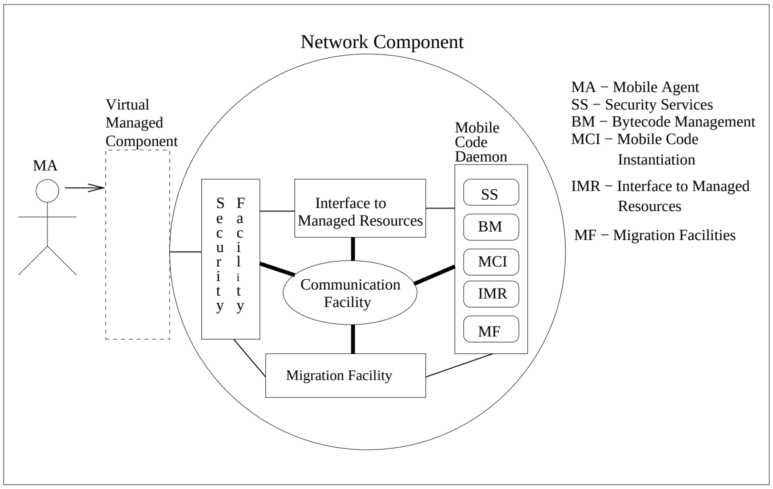


Figure 2.7: Infrastructure for Network Management using Mobile Agents in a Network Component

code to the next Network Component (NC). The MCD calls the appropriate method to migrate the agent, and this notifies the agent that it is about to be transported. The agent therefore has notice to finish its task and notify the MCD in turn whether or not it wants to migrate. If the state is required to be persistent, the agent is packed with its state. Once the agent reaches its new destination, the previous MCD destroys the agent and removes its state. If migration fails, the agent can take a decision on what to do next - change its destination or destroy itself.

The Security Facility (SF) provides security to the mobile agents through authentication, data integrity, data privacy and authorization. These can be implemented easily in Java and provide a level of security to the mobile code in the network. Moreover, the mobile agent is not allowed to access local resources directly, instead, it must access them via the Virtual Managed Component (VMC). Only trusted agents are instantiated and their actions are verified through the Network Security Facility.

A management system requires access to network information at any given point of time. In the mobile agent system, this access to network information is necessarily localized as the mobile agents gather information pertinent only to the network components that they manage. Access to network information is achieved through the Interface to Managed Resources. The agents must use only one of the provided access methods to avail the resources on a network component. A VMC is the only local access point of an agent on a network component, although extensions may be installed on the VMC to allow the export of the interface if required. An NC may have several VMCs. VMCs are self-contained and have no knowledge of the network. They cannot communicate with each other either, unless specifically provided with mechanisms to do so. Remote processes cannot communicate with the Managed Resources unless there is a provision for an extension on the Virtual Managed Component Interface (VMCI).

Communication between agents, and between agents and the environment takes place using a pre-defined ontology. VMCs may be allowed to support various standards such as Guidelines for Definitions of Managed Objects (GDMO) [73], Management Information Base (MIB) or vendor-specific extensions. Agents may communicate with each other using a default blackboard system. A communication facilitator (FAC) provides an indirect communication link between agents on a specific network component. Every MCD must install the FAC to facilitate communication between agents. A mediator (MED) is provided for the communication of agents on two different network components. The mediator can be used to broadcast messages in the network.

Using mobile agents to deal with network management issues enables the system to deal effectively with incompatible protocols, or versions, and to maintain the software required. The manager can easily install and upgrade the agent as and when required. Further, agents enhance the delegated performance of network management, and provide autonomous, on-the-spot solutions to network problems.

This paper presents a detailed infrastructure for the management of networks using mobile agents. Different kinds of agents are identified and used in this system. The system has been developed to accommodate various facilities required by mobile agents in a management function, including state persistence and security. Communication between agents and network components has been facilitated. While the paper tries to introduce and develop almost all the facilities required by mobile agents, the performance of this infrastructure has not been evaluated. It has however been recognized that this infrastructure needs to be used in conjunction with the traditional management systems already in place. Moreover, there are no results nor analyses to demonstrate the use of this infrastructure. The functioning of the system developed and its merits are recognized more by intuition than by actual verification.

The infrastructure requirements presented in this paper, while complete, need to be founded on the existing architecture available in networks. There is a recognized need for the interoperability of the mobile agent systems and the SNMP architecture. The paper 'Integrating SNMP into a Mobile Agent Infrastructure' [181], presents a mobile agent platform called JAMES (see Chapter 1) which extends the functionality of SNMP while providing the requisite infrastructure for mobile agents to conduct their network management functions. While understandably more efficient, this architecture needs to be developed in a more general form [130, 66] to be widely used. There has been an attempt to anticipate and provide for every facility that the mobile agent systems could require in terms of SNMP support. Furthermore, this system needs to be tested extensively to determine its widespread use in networks today.

Summary and Conclusions

The papers presented in the previous section are a precursor of things to come in today's rapidly growing networks. Each of the papers deals with a specific issue in network management and provides a mobile agent based solution/alternative to improve performance. In light of this, the following issues are discussed below:

- Why are traditional network management systems insufficient?
- What is the difference between mobile agent systems and active networks?
- What advantages do mobile agent systems offer in network management?
- How do mobile agent systems compare to traditional network management systems?
- What are the tradeoffs to using mobile agent systems?
- Why have they not been widely used, if they compare so favorably to traditional systems?

Traditional network management systems depend largely on centralized methods of gathering information from the network components. In most cases, there is extensive polling, which leads to a large amount of traffic in the network. Besides, the latency due to information gathering is quite high, leading to probably inaccurate data. Decentralized systems are definitely a better option, but they cannot be easily implemented given the network structure present. Decentralized network managers would be too large and complex, and to scale a growing network would require significant re-configuration and management of such networks can be quite a nightmare.

Mobile agents and active networks [195] are both based on sending code to a remote location for it to be executed. Mobile agents can transfer code to remote locations and execute them, whereas in active networks, the routers are provided the ability to execute code which is injected into the network as special packets. Thus, in active networks, the data is processed by the network components and communication between the nodes is facilitated [191]. The mobile agents, on the other hand, use the communication facilities already available in the network for carrying out the various functions that they are deployed for. The real-time performance of active networks is critically important, as they provide the basic facilities for communication in the network. However, these two technologies are complementary and with mobile agents invading the networks, they seem to be merging into one [50].

Mobile agent systems offer flexibility in terms of being able to move to the network components to gather information. They are an efficient, effective, decentralized means to manage information in the network. Mobile agents can be designed to be lightweight in terms of the code and data they carry, and to have sufficient processing capabilities, to process information and send only the relevant information to the network managers. This would reduce the traffic in the network and as well as the latency in information retrieval. Further, mobile agents offer a degree of scalability that is expedient, and in keeping with the growth rate of networks today. Agents can clone themselves or *die* (merge) to accommodate the needs of the network. In addition, agents can be reconfigured and re-programmed dynamically to meet with changing network needs [85]. This lends added flexibility to network monitoring systems. Agents prevent bottlenecks of information flow to the managers in the network. The use of mobile agents ensures that all the data is not arbitrarily polled by or pushed to the managers of the network. Rather, the mobile agents are capable of filtering the data on network components, and send only the relevant information to the managers.

The rapid pace of changes in networks makes it necessary for equally fast reactions. The traditional reactive and fixed measures may not be sufficient to cope with these issues. Mobile agent systems provide a proactive and autonomous way to deal with the ever-changing environment, and deal with the management of the network components efficiently. Thus, the mobile agent systems provide a degree of robustness in the constantly varying networks, and maintain the equilibrium in the network. An active and dynamic management system is provided by the agents by dint of their communication with each other, and the various communication issues in a network like the broadcast of information, interchange and transfer of data are enhanced by mobile agents providing sophisticated mechanisms for the same [88]. Other management functions may be customized on the fly by the agents, as and when required. Mobile agents for network management are not merely a decentralized tool for management; rather they are a system in themselves, autonomous, intelligent, independent and proactive. Mobile agents should be able to adapt to the network changes, and to whatever functionality is required of them at any given point of time - be it data collection, data processing, or maintenance, in the best possible way.

Mobile agent systems need to be interoperable with the existing management systems in place, like SNMP. Legacy management systems provide access to the managed resources, and if the mobile agent systems cannot communicate with these systems, they cannot access the resources in the network. Further, it is more cost-effective to use mobile agent systems on the existing infrastructure of legacy management systems. Consequently, these agent systems can be used on any network component, without the need to install specific mobile agent platforms, and deal with compatibility problems, or develop new protocols [181, 180]. Thus, mobile agent systems can be viewed as a complement to existing traditional network management technologies rather than independent management entities.

Mobile agent systems for network management must be able to provide a number of extra features to justify their use in the network. A relatively high performance capability and considerable efficacy in fetching data and managing resources [16] is required to warrant the use of mobile agents. Fault tolerance needs to be incorporated in these systems and the systems rendered secure and scalable. Providing agents with mobility and security is a two-edged sword. On the one hand, it is desirable to have agents that are mobile and able to move freely in the network. On the other, mobile agents pose a security threat in the network, as they can be intercepted by potentially malicious elements and their data corrupted, leading to inaccuracies in network management (see Chapter 5). Further, the management of the data itself is an issue that requires careful consideration. Agents may need to transport certain data to the manager, or perform certain computations on the data, and relay only the resultant information to the manager [163]. Either way, the mobile agents need to be provided with the capabilities to either store and transport data, or the computational powers to manipulate data and derive results (see Chapter 3). This leads to an increase in the agents' size, which hinders mobility. The cloning of agents to accommodate increased network size may add performance capacity, but it is applicable in only such cases where the tasks are identical sub-tasks of the main task. Besides, the complexity of the system also increases with cloning of agents [114].

Mobile agent systems for network management have been developed by a number of research groups all over the world. Most of them have used Java to design the agents and have looked into the issues of interoperability with SNMP and optimal performance of the agent systems. There have been many different designs of the agent systems, and various architectures have been put forth. So far, several implementations of these mobile agent systems have proven their distinct advantage over SNMP in network management. Agents are also being used to detect the Quality of Service provided in the networks as a function of their network management role [127].

Mobile agent platforms have been developed for the creation, execution, communication and migration of agents for network management [154]. Agents can implement policies and adapt applications dynamically in the network to provide better service. Different agent communication languages have been developed and used, and ontologies have been defined for these mobile agent systems. Researchers in the mobile agent field have tried to exploit every area possible, from adapting the network components, to data repositories, to network links; taking every factor into consideration and fine-tuning the mobile agent systems to improve their performance.

Mobile agent systems are efficient, provide the perfect delegated work force to perform tasks, and significantly improve performance. However, they are yet limited in usage for several reasons. The mobile agent paradigm is a relatively new concept that not many people are willing to accept very easily. However, given the functionality of networks as they are today, and the rapid pace at which the networks are evolving, it will not be very long before there is a widespread demand for such mobile agent systems, and not just a preference for them. Further, mobile agent systems require the necessary framework/platform to execute on. This differs for different agent systems, and it is therefore difficult to have various frameworks to support different agent systems on network components. There needs to be a standardization [130, 66] of the technology and one agent system model before it is widely implemented.

CHAPTER 3

RESOURCE MANAGEMENT USING MOBILE AGENTS

Introduction

Resources in networks, such as bandwidth, buffer space and CPU time, are a commodity that require careful management and manipulation, especially given the exponential growth rate of networks today. Present day applications today require better Quality of Service (QoS) and reliable resources in terms of bandwidth, and less transmission delay of data over networks. There is a need for efficient means to dynamically discover, possibly reserve, and allocate resources to various applications requesting them.

Networks provide Quality of Service (QoS) to ensure reliable and efficient performance of networking services. This refers to a guarantee of the required resources being provided for a connection to be established in the network. A connection is defined as a provision for communication between any two points in a network. The words ‘connection’ and ‘call’ are used interchangeably in this chapter to mean the same. In establishing connections with a pre-defined QoS requirement, the QoS of connections already present in the network should not be affected. Reliable service may be guaranteed by a dynamic allocation of bandwidth in the network along virtual paths. The dynamic allocation of bandwidth guarantees a path with the required QoS for the most part in the network [201, 134]. In order to be able to guarantee QoS in the network, admission control algorithms are executed to determine the availability of resources in the network. This represents another form of network resource management [140].

Often, there are tasks to be performed on specific network elements that consume a significant amount of the network resources. These network elements may soon be inundated by the amount of data processing to be carried out, and due to the fact that resources are not unlimited, there is a backlog of processing jobs. Under such circumstances, it would be useful if the network element could avail of the available and unused resources in some other parts of the network to carry out its tasks [41].

Several resource management techniques have been successfully applied to distributed systems. Networks not only comprise a part of distributed systems, but may be considered as a distributed system in themselves. The resource management techniques implemented for distributed systems may therefore be applied to networks too; the mobile agent-based resource management systems for the same may also be extended and considered feasible in networks. Most of the work conducted in the area of resource management has focused on resources in distributed systems. This work has been presented here as it is considered significant and parallel to resource management in networks.

Network Service Architectures

The network traffic needs to be controlled and managed (scheduled) to optimize performance of the network, and resources need to be located and diverted if necessary to achieve maximum utilization. The Internet Engineering Task Force (IETF) developed two architectures - Integrated Services (IntServ) and Differentiated Services (DiffServ) to provide resource-based services in the network. Various admission control algorithms and resource reservation protocols like RSVP [18], Boomerang [65], etc. have been developed and are widely used in networks today [188].

The Resource Reservation Protocol (RSVP) is a protocol that has been developed for use over the Internet to reserve resources from source to destination. RSVP provides certain QoS guarantees and the routers on the path along which resources are reserved can schedule and prioritize packets accordingly. RSVP can "allocate and release resources", maintain state in the network components, and "provide other mechanisms" [97] that may be required to facilitate Guaranteed Load Service [176].

Traditional resource management techniques to control traffic, provide bandwidth or delay guarantees which are dependent on the conditions prevalent in the network.

Different networks in different places require specific QoS guarantees and architectures, which generally cannot be pre-determined, and moreover are static in nature. A dynamic strategy is required wherein resources can be managed in a distributed manner to suit the changing network needs. Several dynamic resource management algorithms have been developed in the past [97, 188, 58, 70]. The mobile agent paradigm lends itself naturally by dint of its features to offer solutions to network resource management. Mobile agents proffer an open programmable networking environment, that can be used to control and manage the resources of the network efficiently [201].

Mobile Agents for Resource Management in Networks

Resources in the network are a prized commodity and require judicious control and allocation. Mobile agents provide an apt management solution to the resources, given their pro-activeness, intelligence, and flexibility. The following papers each present a mobile agent solution to the management of various resources in the network.

Distributed Resource Allocation Using Mobile Agents

Resource: Challenger: A Multi-Agent System for Distributed Resource Allocation, Chavez A., Moukas A., Maes P., Proceedings of the First International Conference on Autonomous Agents '97, Marina Del Ray, California, 1997 [41].

The paper introduces a multi-agent system that manages the resources in a network, and executes a distributed allocation scheme to share and utilize the CPU time efficiently. The system, Challenger, consists of agents that manage only those resources that they are locally aware of. The agents communicate with each other to share information about resource availability. The resource under consideration in this system is the processing power or CPU time of a machine.

This system is based on the assumption that there are always some under-utilized or idle resources in the network, even when the network is congested, i.e., only parts of the network experience high loads at any given point of time. There is therefore a need for the harnessing of the unused resources to be utilized, rather than wait long periods of time in the congested parts of the network. Thus, for any task introduced in the network, there should be enough resources available for it to be performed, either locally or remotely.

Challenger performs processor time allocation in order to facilitate the completion of tasks, locally or remotely, dependent on where processing capacity is available. In doing so, the system needs to ensure a minimal mean flow time and response ratio and utilize the CPU to its capacity. The mean flow time refers to the average period of time taken for a job to be completed, from the time that it started. The response ratio is the ratio between the actual time taken to complete a job and the time it takes on a completely free processor. This system, being a distributed model, has no single point of failure. The system is adaptive and can quickly adapt to changing network conditions, and continue to provide at least a minimal quality of service. Furthermore, the agent system was designed to be robust to cope with any failures.

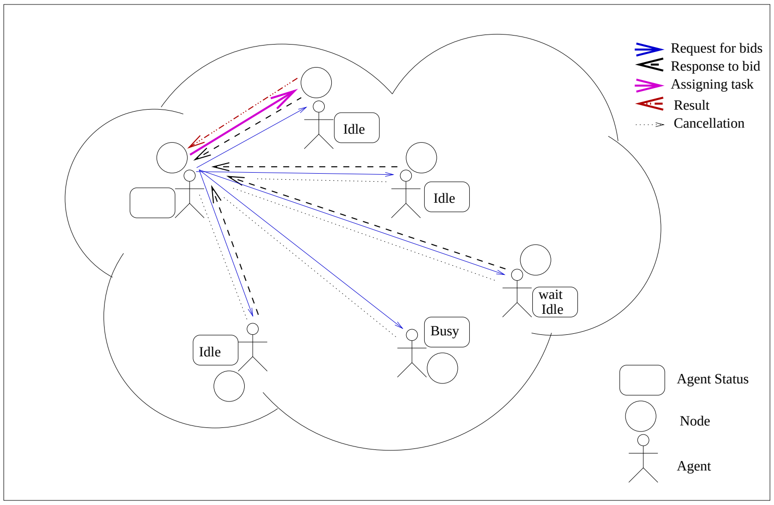


Figure 3.1: Interaction between Agents for Resource Allocation in the Challenger

The design of the Challenger system is quite simple. An agent is present on every machine to manage the processing resource (CPU allocation) of that machine. The agents allocate resources for incoming tasks, and assign tasks to the machines based on the availability of processing power. This resource allocation is carried out in a series of steps (see Figure 3.1):

- Job Origination: The agent on each machine sends out a request for *bids* on available resources to all machines upon receiving a task. This is done in order to determine the capacity of various machines and allocate the task only where there is sufficient processing power.

- Making Bids: Upon receiving the request for bids, an agent either responds immediately if it is idle, or after a wait period, if it is already busy with a task. The agent includes in its response the period of time required to complete the task on that particular machine. When an agent submits a bid or is running a process, its status is busy.
- Evaluating Bids: The agent that sent out the request for bids evaluates all the responses it receives and assigns the task to the machine from which it got the least processing time bid response. The agent also sends out a cancellation message to the other agents.
- Result Return: The agent that was assigned the task returns the result to the original agent from whom the task originated, after completing the task.

The Challenger system ensures that in the event that an agent does not receive a response to all its bids, it waits a pre-determined period of time, and then assigns the task to the first bid it receives. If the agents on the other machines do not receive either an assignment or a cancellation after having sent out a bid, they wait a set length of time, then change their status to idle. In the third eventuality that the originating agent does not receive the result of the task from the assigned agent, it sends out a new request for bids after a timeout period, or if processing power is locally available, it executes the task on its own machine. This provision for any eventuality of a failure enables the system to be robust and fault tolerant.

In all of the simulation results, the Challenger has been found to reduce the mean flow time of a task. The authors took into consideration

- the margin for error while calculating the average job completion time,
- the message delay caused by the network between the transmission and receipt of a message,
- the evaluation delay that is the period of time the agent waits to receive responses from all other agents to its request for bids.

It was found that the mean flow time did not always decrease when using Challenger. The best results were obtained when the message and evaluation delays were low. This is an obvious consequence, as a task can be completed in a short period of time when the time taken to transmit messages is small.

The Challenger agents are also equipped with learning capabilities, and can calculate the network delay, which can then be used to calculate the message delay.

This information is useful to make better task assignments, and achieve a better task completion time. Therefore, the mean flow time for tasks in the Challenger system never exceeds the mean flow time of jobs that are carried out locally. Thus, only very large tasks are sent to remote machines to be executed, while all others are processed locally. A better estimate of the message and evaluation delays promotes judicious assignment of tasks and the mean flow rate is reduced.

The system functions on a number of assumptions, however, not all these assumptions always hold true. The system can only execute one task at a time on each machine. Further, only tasks with a limited execution time frame can be accommodated here, as otherwise, the originating agent would keep timing out and re-sending requests for bids. The system was designed to assign job priorities to each task. Since this specific system was designed with a view to minimize the mean flow time, priorities are also assigned based on the processing time. Thus, tasks with the shortest processing time are given the highest priority. This completion/processing time needs to be computed first before the priorities can be assigned. The agents that submit bids do so only after calculating how long it will take them to complete the task. The agents use the task priority values in order to compute the completion time before submitting bids to the originating agent.

The Challenger system was executed on a network of 2-10 machines. It was found that performance deteriorated when the number of processors was increased. This is indicative that the system does not scale very well. This has been justified in the paper by the argument that a larger number of processors indicates more tasks to be executed, thus making system performance deteriorate. However, it is required and only to be reasonably expected that with more jobs *and* more processors, the system performs well.

The Challenger is a primitive implementation of the use of agents to provide management of resources. Here, the agents perform the simplest form of task scheduling, by waiting for responses about available CPU time from other agents. These agents are not capable of handling more than one task at any given point of time, and they perform pre-determined functions only. They are not dynamic enough, and can function in only a limited manner.

This paper borrows extensively from the ideas propounded in [120], where a task scheduler is implemented to schedule tasks in distributed systems. The authors extend the distributed systems task scheduler to develop this agent model for sharing and allocation of CPU time. They have improved on it by adding some learning capabilities to the agents, in order to facilitate better performance. This project has not been improved upon although it was introduced as early as 1997. There is yet much to be done if this system is to be used extensively in networks today for allocation of processor time.

Distributed Job and Resource Management

Resource: A Multi-Agent System for Distributed Job and Resource Management, Jozef Winkowski [192].

The paper presents a multi-agent system to manage the local resources on a machine, and carry out the various jobs that arrive at that machine. Each job is further divisible into separate tasks which require specific resources. The resources to be managed on the machine are either a form of processing facility (processing power), or data. A local agent resident on each machine is responsible for the tasks and resources of that machine.

The whole system is defined as having pools of resources on each network element. Each pool contains resources of different kinds, such as processing power, storage units and so on. An agent on each of the network elements is responsible for monitoring the jobs that arrive at that specific element and allocating resources for each of the sub-tasks of that job. A job could require resources from different pools located on different network elements. The local agent is responsible for contacting the local resource pool or other agents on different network elements to make available the resources required for the particular task (see Figure 3.2). The system functions such

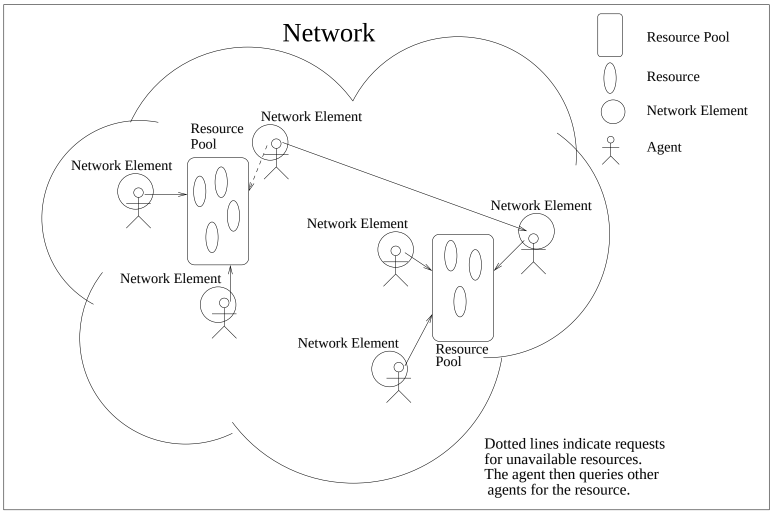


Figure 3.2: Resource Pools for Resource Management by Agents in a Network

that no job is refused, and every job is accepted and processed.

The agent on each machine creates sub-tasks for every job that arrives at the network element. The determination of what resources are required by the job, and the assignment of these resources from the resource pools in the network is carried out by the agent. This is called a plan for the allocation of resources for a particular job.

The system is implemented in a *game-like* fashion, where each ‘player’ waits his turn, in two phases. The first phase deals with handing each job a ticket with a number and the time stamp. The tickets are numbered in ascending order, and the job priorities are thus decided. The lower the number and earlier the time stamp on a ticket, higher is the priority of that job. The second phase is where the agent gathers information about the resources available in the network, and allocates the required resources to the job. The main task of the agent in the second phase is to either send out requests to access resources in the resource pools, or to wait for resources to become available before sending out the access requests.

No task created by an agent can preempt an earlier task to access resources from the resource pools in the network. Thus, it is ensured that jobs that arrive earlier in the system avail the free resources and are completed duly, before other jobs are given resources. The system performs the following sequence of events (see Figure 3.3):

```
Agent creates a new task
Task chooses a plan for allocation of resources from different pools
Task is given a numbered ticket
Task requests to access resources from a pool
Task waits / withdraws request to access resources from pool
Task reserves resources according to plan
```

The system functions under the assumptions that each job is finite, and that no resource can be used by any other job if it being accessed by a particular job. An agent creates a task, and this task makes and follows a *plan* for the access of resources

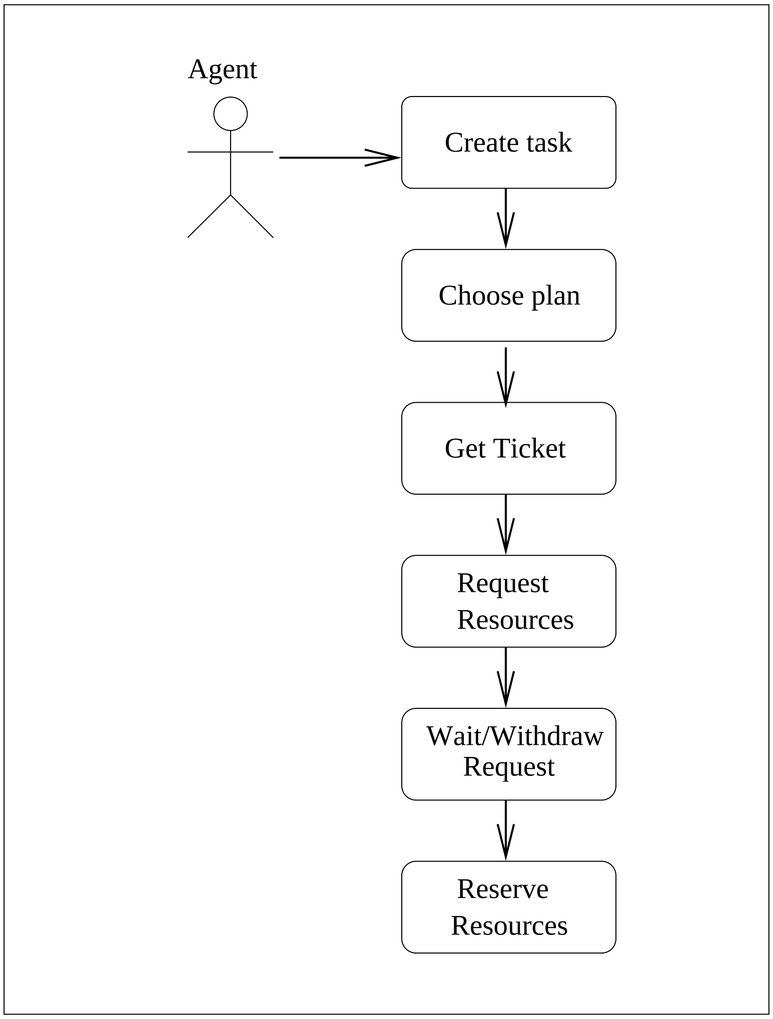


Figure 3.3: Sequence of Events performed by Agents in a Network while managing Resources

from the network resource pools. Resources could be accessed from different pools, or all from the same pool in a plan. The plan is made depending on whether resources are available or not in the pools. This knowledge is made available by the different agents coordinating and communicating with each other. Once a plan is made, then the task has to follow that plan to allocate resources for a job.

The system presented in this paper is very similar to the Challenger mobile agent system [41] for the distribution of processing time. The primary difference here is that there is a provision for resources to be made for a single job from different resource pools, not just from one single local/remote machine. The sequence of events in this system is presented coherently and a set of proofs has been presented to depict that the system does indeed ensure that every job that is initiated is allocated resources and duly completed. However, the paper does not mention how the agents communicate with each other and obtain up-to-date information on the resource availability in the network resource pools. Also, the agents used in this system are not really mobile. They are more static agents resident on the network element, and perform predefined functions.

Congestion Management

Resource: Dynamic Congestion Avoidance Using Multi-Agents Systems, Nicolas

Rouhana and Eric Horlait, MATA 2001 [161].

The paper suggests an extension of Random Early Detection (RED) using multi-agent systems to avoid congestion in the network. The Dynamic Weighted Random Early Detection (DWRED) model proposed by the authors uses agent communication and coordination to dynamically change parameters in the algorithms being executed on a network component in order to avoid congestion.

This model is based on a dynamic implementation of the Weighted Random Early Detection, wherein various parameters are changed dynamically in the algorithm being executed. There are minimum and maximum thresholds defined for the number of packets in each traffic queue. Every category of traffic has a minimum amount of packets in the queue. Thus, service is not denied to any class of traffic. Packets are dropped based on the minimum threshold defined for that class of traffic and for the average number of total packets in the queue. The priority of packets is maintained, and packets with a higher priority have a higher threshold (more packets allowed longer on the queue) to ensure that they are not the first to be dropped.

Each router has agents which communicate with agents on the neighboring routers to control the parameters during execution of the random drop algorithm. The agents determine the class of the packets arriving, compute the average queue size and communicate with the other agents if required, make changes to the minimum and maximum thresholds in the queue, or drop the packet with a random computed probability. Thus, the agents behave in an intelligent and autonomous way to control and avoid congestion.

This dynamic model of agent-based detection of congestion provides a fair distribution of packets in each of the routers in the network. The agents ensure that not all low-priority packets get dropped to accommodate packets of higher priority, thereby providing a high resistance to packet losses in the network. The agents monitor the minimum threshold for each class of packets in the queue, and due to the dynamic control, every link out of a router is utilized equally. This is achieved by dint of the agents allowing packets of all classes on the queue; otherwise, in the event that only the high-priority packets are accommodated on the queue, only those corresponding links would be utilized.

The paper presents results from simulation of the algorithm to indicate its improved performance. The use of agents to promote dynamic congestion control and avoidance ensures against packet losses in the network. Agents are used to cooperate and communicate with each other about the number of packets of each service class in the routers' queues. Using this conveyed information, the agents can dynamically change the minimum and maximum thresholds of the service packets on the queues, and pre-empt congestion. However, there is no clear explanation given as to how exactly the agents are deployed. There needs to be more explicit information and details on the parameters of the algorithm and the changes that are implemented in each.

Resource Reservation

Resource: Resource Reservation Agents in the Internet, Olov Schelen and Stephen Pink [169].

This paper presents a way to reserve resources for incoming requests in the Internet, with the use of agents. The agents check each request, and reserve resources for the same. Also, resources from different sources to a common destination are aggregated and reserved. The architecture proposed in this paper is scalable and designed for uni-directional virtual paths.

The network is separated into domains, and each domain has a reservation agent. The reservation agent has information about the topology of the network, and also obtains information from the routers about the link properties. This requires no additional overhead, as the routers already contain this information obtained by network management protocols. The agents employ admission control mechanisms to determine the source of traffic and accordingly reserve resources.

A reservation request consists of a source, destination and the bandwidth required.

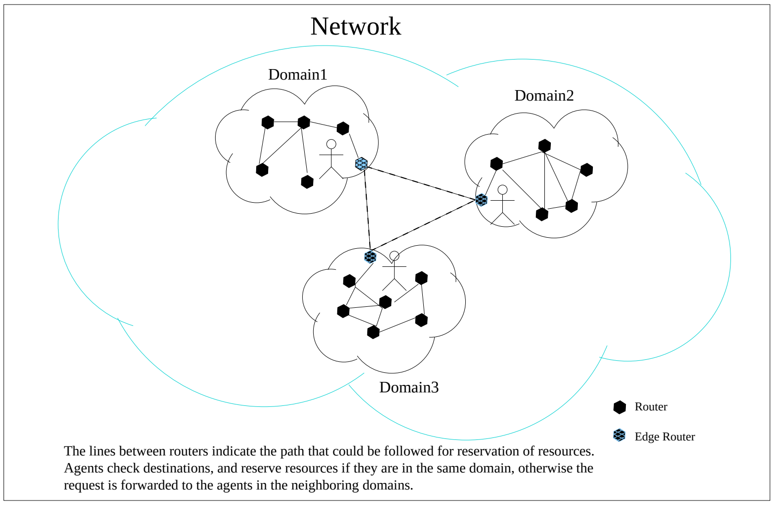

Network
Domain1
Domain2
Domain3
Router
Edge Router
The lines between routers indicate the path that could be followed for reservation of resources.
Agents check destinations, and reserve resources if they are in the same domain, otherwise the
request is forwarded to the agents in the neighboring domains.

Figure 3.4: Reservation of Resources by Agents in Different Domains of a Network

Upon receiving a reservation request, an agent checks to see whether the source falls within its domain; if it does not, it sends out the request to the agents in the neighboring domains. In the event that the source is in the domain of the agent, the agent begins to reserve the required resources in the nodes along the path towards the destination (see Figure 3.4). The path is defined by the routing protocol employed. The path may contain several domains in the network. Therefore, the agent of one domain can reserve resources until an edge router in its domain, after which the request is forwarded to the agent of the next neighboring domain with information about the edge router and reservation request. Thus, the edge router of the previous domain becomes the source for the new domain for the current reservation request.

The neighboring agent has information only about the immediate predecessor source of the request, and not the original source. However, it does have information about the destination and the resources required. This limited information is useful in aggregating resources towards a particular destination. Thus, resources are reserved as a whole towards a particular destination regardless of the source of the request. Priority packets can use these resources to be forwarded to the destination quickly. Aggregation of resources makes it easier for the agent to have a generalized reservation of resources towards a particular destination. The agent then only has to maintain reservations as a whole towards a destination and send any packets destined for that destination through the reserved *funnel* of resources.

The use of agents and the aggregation of resource reservations facilitates the advance reservation of resources. Further, priority packets do not have to wait for resources to be allocated to them, as there are already reserved resources. Aggregation of resources, rather than a per-flow reservation model, improves scalability and does not require agents to maintain the state of each request. Agents can adapt to the changes in topology and routing as required, as they have direct access to the routers' information.

Agents act as police points in edge routers to ensure that only the traffic for which resources have been reserved passes through the domain. Thus, packets are checked on a dynamic basis, either one-by-one or at random. Violations are reported to the previous agent of the domain where the packet came from.

The authors have presented an agent model for reservation of resources in network elements. However, they have presented no analytical or experimental results to compare and elaborate on the performance of this system. Also, there is no information on what action is taken when resources are not available to be reserved. The agent model is described as efficient for uni-cast communications, while RSVP is preferred for multi-cast communications.

This paper elaborates on the ideas presented in [171, 170], by the same authors. The authors have implemented advance resource reservation using an agent-based architecture to promote efficient resource scheduling and usage. Performance measures of aggregated resource reservations are presented in [168]. The results therein indicate that agents can be used without too many changes in the inherent systems. The agents use the routing databases to access information on network topology, and the underlying information about link resources available in SNMP to set up the forwarding state in routers. Hence, agents can reserve resources with little or no intervention from the routers, and do not maintain a reservation state when not required to, i.e., when the reservation of resources does not affect the routing of packets. Owing to the reservation of resources by agents, the actual endpoints of communication do not have to be present until the communication session begins. Therefore, the advance resource reservation agents present an autonomous and independent model to reserving resources in the network to facilitate routing with QoS.

Resource Discovery

Resource: Using Mobile Agents for Network Resource Discovery in Peer-to-Peer Networks [60].

Peer-to-peer (P2P) networks are getting to be very popular these days due to their capacity for more storage, and bandwidth usage. P2P networks also offer greater use of the existing hardware, and more computing cycles. However, resources for such networks need to be identified and allocated. This is an increasingly difficult problem, as networks are getting larger, and resources are more spread out. Further, it is important that only the useful resources are identified in a P2P network. This paper looks at a mobile agent solution to the discovery of resources in the network.

A mobile agent is created and traverses the network, discovering other peers and resources. The exact nature of resources has not been defined. However, every peer is said to contain resources that identify its type and address, and information about other known peers. Other resources may be directories of files contained by the peer, available CPU time, etc..

The agents follow the algorithm presented in the paper to discover resources/peers. The algorithm is briefly represented in the following steps, and each step is explained in some detail thereafter:

```
                              THE ALGORITHM
Create a Mobile Agent
        -> give it a traversal time (t)
        -> give it a branching factor (b)
Mobile Agent gets (n) Peer Addresses to visit
Mobile Agent Clones itself (n times)
Mobile Agent visits Peer, exchanges and processes information
        -> decrease traversal time
        -> if agent meets another agent with same creator node,
           one agent is terminated
        -> agent clones itself again for each of the currently known peers
Mobile Agent returns to the creator when traversal time = 0
        -> updates creator with all the information it has collected
Loop back to Cloning Step
```

A description of each of the steps in the algorithm is given below (see Figure 3.5):

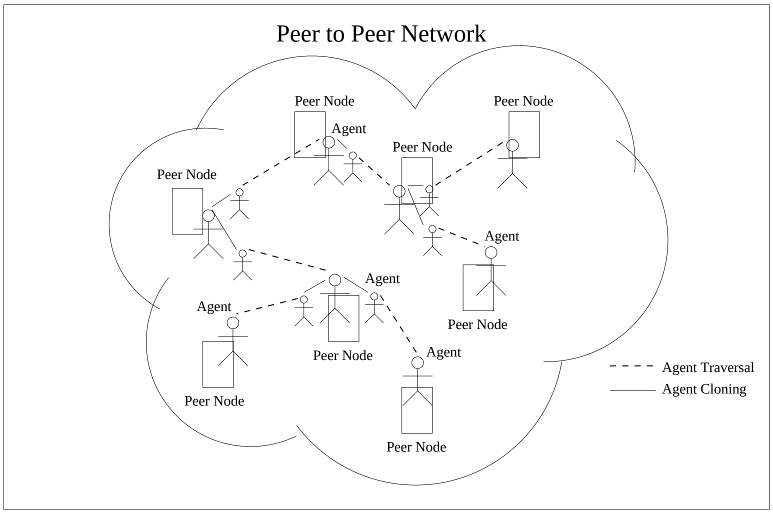


Figure 3.5: Discovery of Resources by Agents in a Peer to Peer Network

- *Creating a Mobile Agent:* A 'creator peer' creates a mobile agent. The agent then stores information about the peer as its home address. The traversal time is determined for the agent as the maximum number of peers it can visit before it returns to the creator peer. An agent is also given a branching factor, which is the number of times the agent can clone itself.

- *Peer Addresses to Visit:* The agent is provided with a certain number of addresses of peers it needs to visit to collect data about them. These addresses are preferably from different subnets in the network, so that the agents can collect information from all parts of the network.

- *Cloning:* Clones of the mobile agent are created to visit each of the peers. Thus, an agent is created to visit each of the initially known peers in the network.

- *Peer Processing:* The agents exchange and update information about peers after reaching a peer. The traversal time is decremented, and if an agent meets a clone from the same creator agent, one is killed. The agent is again cloned to accommodate the number of peers known to the current peer.

- *Agent Return:* The mobile agent returns to the creator agent if its journey time expires (becomes 0). Upon reaching the creator agent, the agent updates the peer with all the information it has collected.

- *Loop back to Cloning:* If required, the entire process is repeated to gather more information. The addresses of peers for whom information is already available are not used in order to prevent re-visitation.

The authors have not included security (see Chapter 5) as a consideration in this implementation. The authors argue that they use the security features provided by the Java Virtual Machine and the Aglet Software Development Kit (ASDK) [6] security model that have been used in their implementation. These inbuilt security features have been considered adequate for the system developed by the authors. The ASDK provides several classes that are extended and used in this paper to implement the algorithm. The agents, the communication between them, and knowledge of resources in the network are available in various classes in the ASDK.

The use of mobile agents has several advantages. Mobile agents consume less bandwidth, and can encapsulate all the information. Further, mobile agents perform local processing of data, and this is useful as global data does not have to be transported. The agents are asynchronous and autonomous and therefore, need no instruction or coordination. They perform their tasks and return to the creator peer upon completion of their task. Mobile agents serve to disseminate and distribute the latest information in the network. They can be cloned or terminated to manage their numbers, and are quite fault tolerant in preserving the information that they or their clones have collected until the point that they fail. At any given point, there are a considerable number of clones in the network, and some of them carry partially duplicate information. While this may seem as extra and unnecessary load in the network, it preserves the state of the network to a large extent in the event of any faults.

This paper presents a very effective method of utilizing the mobility and data encapsulation and storage capabilities of agents to dynamically discover resources in the network. Up-to-date information about the resources are obtained as the agents are always on the move, exchanging and updating information about the resources of all the peers. There has however, been no information given on the overhead of employing these agents, or the performance analysis of this model. Further, this model has been applied to only subsets of the network, and is not easily scalable.

In ‘Agent-Based Resource Discovery’ [104], agents are present on each network element and are aware of each other. These agents exchange information periodically about the resources on the network components. When detailed information is required about any particular network component, an agent is created to gather this information specific to the network component.

An agent architecture for resource discovery is also given in [31]. The management of resources, scheduling and allocation of the same has been equated to the discovery of resources in the network. To this end, this paper explores an agent-based service discovery model that also provides scalability and adaptability. Agents are modeled in a three-level hierarchy (communication, coordination and local management) and manage the resources; an agent is both a service provider and a service requester. Agents are added to the environment whenever a resource is added to it. The agents can be ‘registered’ or connected to the existing hierarchy of agents by maintaining only one connection with the level above it, and however many connections with the lower level agents.

The main functions of service (resource) advertisement and service discovery are performed at the coordination level. Agents can communicate with each other and discover resources to be utilized. This paper has provided the prototype model for an agent-based resource discovery mechanism based on the PACE [144] toolset, but there are no analytical results provided. This paper was considered here due to its contribution to the resource discovery problem using mobile agents in networks.

Network Control and Resource Management

Resource: An Approach to Network Control and Resource Management based on Intelligent Agents [201].

This paper examines a complete network control architecture using agents to perform the tasks required. The agents control the admission of traffic into the network, provide the requested QoS and route the traffic from source to destination according to required bandwidth specifications. The architecture considered in this paper is the Implementation of Agents for Connection Admission Control (CAC) on an ATM Testbed (IMPACT)[1] implemented for network control in ATM networks by Intelligent Agents.

The network is modeled such that it has one Network Provider (NP) and several Service Providers (SPs). The NP is responsible for managing the entire network, while

[1] The IMPACT project is funded by the European Commission under the framework of the Advanced Communications Technologies and Services (ACTS) program.

the SPs provide service and connectivity to clients. Each SP has a Resource Agent (RA) for managing the resources allocated for each source-destination pair. Network resource management is carried out by dynamically (re)allocating bandwidth on the source-destination paths.

The traffic in the network is in the form of requests for connections from a source to a destination, with a specified Class of Service (CoS). Each SP has a set of Virtual Paths (VP) for every source-destination pair. The path is chosen from this set, according to route and not bandwidth specifications.

The network model has several agents at different levels performing various functions to facilitate control of resources. An enumeration of the different agents and their functions are given below (see Figure 3.6):

Proxy User Agent (PUA): The PUAs communicate the user requests for connectivity to the network control (connection) agents. The PUAs can either wrap the information in conventional ATM signalling messages, or send it using an Agent Communication Language (ACL).

Connection Agent (CA): The CA receives information from the PUA about a call request, and accordingly selects an SP to provide the connection. The CA sends out a request for bids from all the RAs in order to find the best possible RA. The RAs of the SPs respond to the bids with information about the resources available to them,

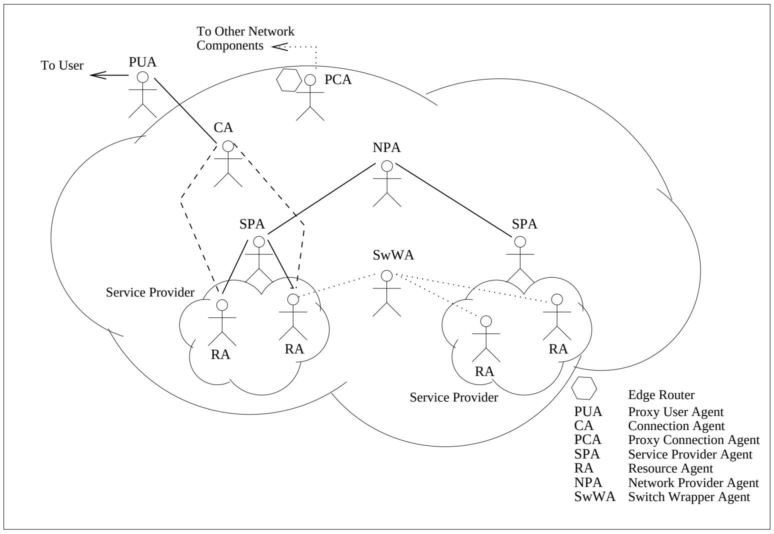


Figure 3.6: Communication between Different Types of Agents in Network Control and Resource Management

and the CA then makes the decision as to which SP to select. The CA also sends a cancel message to the RAs that are not selected.

Resource Agent (RA): RAs are responsible for the main network control strategies implemented in the network. Each SP has RAs for each source-destination pair. The RAs control and manage the resources of the SPs and send out bids for calls to the CAs, depending on available resources and the QoS of the calls. Various strategies for resource management can be implemented in the RAs, depending on user specification. The RAs also offer increased flexibility in dealings between the SPs and the NP.

Service Provider Agent (SPA): The SPAs represent the SPs, and distribute the available resources among the RAs. The SPAs ensure that the RAs manage the resources such that the SPs maximize their profits.

Network Provider Agent (NPA): The NPA is the representative of the NP, and negotiates with the SPs and its RAs on behalf of the NP. The NPA also interacts with the Switch Wrapper Agent to establish VPs.

Switch Wrapper Agent (SwWA): The SwWAs are virtual abstractions of the ATM switches in the network, and provide the interface for network control functions. The SwWAs communicate with the RAs through the ACL. These commands are translated into low-level operations by the SwWA to be executed by the ATM switch.

Proxy Connection Agent (PCA): PCAs have been included in this implementation to allow scalability in the network. The PCAs reside at edge network components, and can communicate with network components outside the network.

The paper has realized an agent model to manage the resources and perform network control in a network. Although the model and each of its components have been described in detail, there are no experimental results provided. The authors have provided an early trial result, wherein the agents attempt to reallocate resources along a path from source to destination to accommodate a call request. However, the overhead incurred in implementing an agent model in the network has not been considered in any of the results provided. The resource reallocation scheme proposed in this paper, carried out by agents, is very similar to the dynamic reallocation of resources suggested in [97]. A comparison of overall results and overhead utilization of the two schemes would greatly help in establishing a superior scheme. However, this system does present a flexible and programmable model for network control.

The paper focuses on resource reallocation using agents on ATM testbeds. With networks carrying more real-time data and traffic, the Asynchronous Transfer Mode (ATM) has become increasingly popular. This transfer medium is widely used to send data and video at high speeds in the network as it extremely flexible in supporting most existing technologies [8]. The algorithm presented in this paper may also be used in IP networks providing QoS, with hardly any changes. Network programmability is the only requirement that this model demands.

'Open and Flexible Control and Resource Management for ATM Networks using Intelligent Agents' [134] presents a network control model in managing ATM networks with intelligent agents. The need for open and flexible network resource control, especially in ATM networks has been recognized. The adaptation of existing standard protocols for specific service requirements or the deployment of complex schemes to maximize resource utilization is not feasible with the current protocols. Therefore, an open control model using intelligent agents for resource management has been introduced and tested on ATM testbeds, and found to provide adequately encouraging results. This paper is an extension to the IMPACT project and elaborates on the techniques used in [201].

Traffic Control and Resource Management

Resource: FIPA-compliant Agents for Real-time Control of Intelligent Network Traffic, [98].

This paper suggests two techniques for load control in Intelligent Networks (INs), using agents that are compliant with specifications recommended by the Foundation for Intelligent Physical Agents (FIPA) [66]. Real-time telecommunication networks are considered, with enhancements incorporated in the FIPA agents to provide effective load control.

An IN is one where the network components are intelligent, i.e., they are each capable of introducing, controlling and managing services in an independent manner. There are four types of nodes in an IN: Service Switching Points (SSPs), Service Control Points (SCPs), Service Data Points (SDPs) and Intelligent Peripherals (IPs). User access to services are provided by the SSPs. The SCPs receive requests from the SSPs and accommodate them as per their requirements. The SDPs provide storage facilities, and the IPs provide an interface for interaction with the users [101].

Load excesses in INs occur when the resources offered to SCPs exceed the resources available. Load control mechanisms employed in the nodes of INs need to be stable, scalable, responsive, fair, efficient and simple. They are designed to relieve the SCPs of excessive loads and improve response times and service availability. There are cooperative and non-cooperative load control mechanisms employed in the nodes. The cooperative mechanisms are executed in the SSPs, which come into effect upon being notified by the SCPs of an increase in load. Non-cooperative mechanisms are implemented in the SSPs as algorithms that are executed when the SSP detects that there is an increase in the response time of requests from the SCP.

The load control mechanisms implemented in the nodes are considered insufficient due to the increasing heterogeneity of network traffic, and the localized node-concentric nature of these mechanisms. Different traffic types require different dynamic methods of control, and further, node-based control mechanisms may lead to the propagation of congestion in the network away from the nodes to elsewhere in the network, affecting the service rates.

The authors have therefore developed an agent-based network load control mechanism, which is knowledge-centered, flexible, robust and adaptable. There are three different kinds of agents - a *Quantifier agent*, that monitors the resources of the SCPs and communicates this information to other agents, a *Distributor* agent, that oversees the resources of the network as a whole and an *Allocator* agent located in the SSPs that execute algorithms to compute load overloads, and also communicate with other agents to control load by throttling service requests.

There are two different resource allocation strategies using the above agent types that have been investigated and implemented in this paper - a cooperative market strategy and an ant-based strategy.

Cooperative Market-Based Strategy

In the cooperative market-based (MB) strategy, the MB-Distributor agent collects bids from the MB-Allocator and MB-Quantifier agents (see Figure 3.7). The MB-Quantifier bids are indicative of how many resources will be available during a future time interval, and the resource requirements for each class of service in a SCP. The MB-Allocator bids, on the other hand, contain information on how many service requests for the different service classes can be expected over a future time interval. Using this information, the MB-Distributor agent allocates 'tokens' to pools for different services at the SSPs. The tokens signify an SCP-service class pair. Tokens are 'sold' by SCPs, and 'bought' by SSPs. An SCP can only sell as many tokens as it has been allocated by the MB-Distributor agent. If there are no more tokens in the pool of an SSP for a particular SCP, then that SCP no longer has resources to process a specific service class of request.

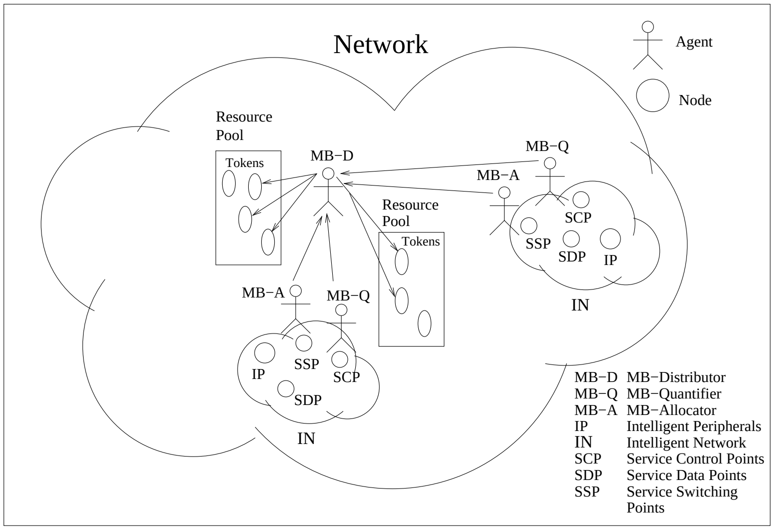


Figure 3.7: Cooperative Market Based Strategy by Agents for Allocation of Resources

There are certain restrictions imposed on this strategy to help maximize the network profit. The MB-Distributor allocates all the available tokens to the SSPs, focusing more on the MB-Allocator agents of SSPs that submit higher ‘bids’, i.e., a higher expected number of requests. The MB-Distributor is also responsible for maintaining a balanced load across the SCPs in the network.

Ant-Based Strategy

The ant-based (AB) strategy is based on the behavior of ants (see Figure 3.8). A mobile AB-Ant agent is generated at regular time intervals for each service class at each of the SSPs, and sent to selected SCPs. Each SSP maintains a probability table for each service class for all the SCPs. This table contains the probability that for a particular service class, a particular SCP will be chosen as the destination. The probabilities are akin to the pheromone levels generated by biological ants. The AB-Ant selects an SCP as the destination based on the entries in the probability table. AB-Ants may also visit any SCP at random, where the probabilities of visiting each SCP are equal, under an exploration scheme.

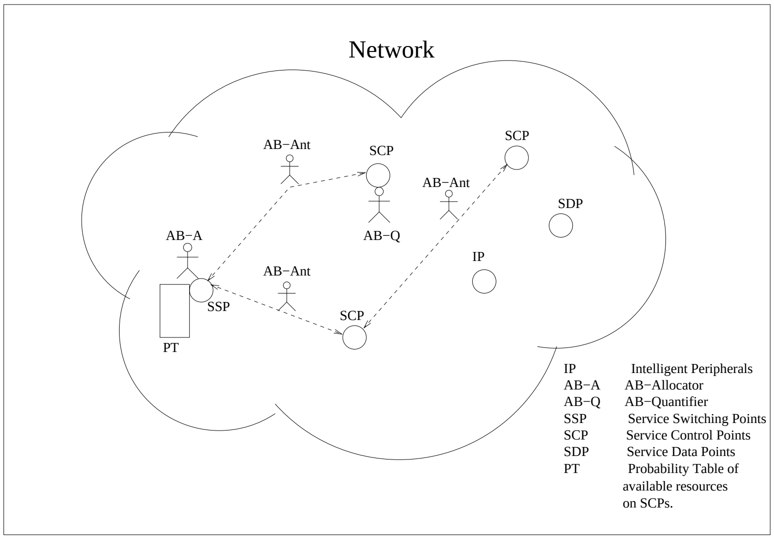

Network
AB−Ant
SCP
SCP
AB−Ant
SDP
AB−A
AB−Q
AB−Ant
IP
SSP
SCP
PT
IP Intelligent Peripherals
AB−A AB−Allocator
AB−Q AB−Quantifier
SSP Service Switching Points
SCP Service Control Points
SDP Service Data Points
PT Probability Table of available resources on SCPs.

Figure 3.8: Ant-Based Strategy Used by Agents for Resource Allocation

AB-Ants interact with the AB-Quantifier agents on each SCP visited, from whom they collect information on the processing time of the SCP for the class of service they represent. This information is conveyed back to the AB-Allocator agents on the SSPs, once the AB-Ants return to their SSPs. The AB-Allocator agents then update the pheromone tables with values computed using the processing times and the travel delay of the AB-Ants. These updated values are representative of the new probabilities with which an SCP can be visited for a specific service type.

Among the two strategies, the market-based strategy was deemed the more effective for real-time telecommunication INs, as it imposes real-time constraints on the agents. A detailed description of the messages exchanged between the different agent types, and also the Agent Communication Language (ACL) used has been given in the paper. The authors have also suggested several modifications they deem necessary to the existing FIPA architecture to make it more suitable for real-time systems. However, there is a noticeable lack of data regarding the results and implementation of the two strategies, no information is given on the overhead consumption of these strategies too. Nevertheless, the paper presents a feasible and seemingly efficient system to deal with resource control in real-time systems.

Resource Control in 3G (Mobile) Networks

Resource: Distributed Intelligent Control and Management for 3G Networks [29].

This paper presents a mobile agent solution to the management of resources in 3G (mobile) networks. 3G networks have variable bandwidth requirements and a more recent radio architecture, wherein bandwidth is allocated to radio cells to prevent local congestion. The Fixed Channel Allocation (FCA) scheme has been modified to accommodate the borrowing and locking of resources from neighboring cells under moderate or heavy traffic conditions. Distributed resource allocation schemes for mobile networks using mobile agents have been developed and used [20, 21, 19, 22], wherein agents reside in each base station of mobile networks, and monitor and manage channel allocation requests and algorithms. This paper considers a similar approach using mobile agents in 3G networks to provide efficient, and real-time resource control.

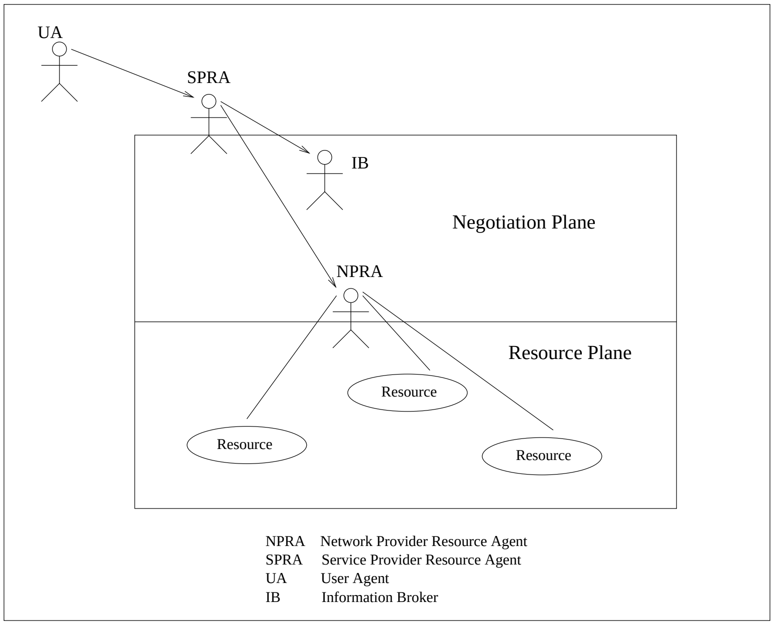


Figure 3.9: Two-Plane Agent Architecture to Manage Resources in 3G Networks

The authors present an agent architecture on two planes - a negotiation plane and a resource plane (see Figure 3.9). The user agents negotiate with service provider agents for the use of resources, and the service provider agents receive resources from the network provider agents in the negotiation plane. Network resources are controlled on the resource plane. Each network operator has a resource plane of its own for its resources.

The network contains Network Provider Resource Agents (NPRAs), Service Provider Resource Agents (SPRAs) and Information Brokers (IBs). The SPRAs communicate with the IBs or directly with NPRAs. The SPRAs monitor the usage of bandwidth, and can also communicate directly with the Service Providers. The authors have implemented the agent system based on the FIPA architecture.

Although this paper does not provide elaborate details about the implementation and results obtained in using mobile agents for resource control in 3G networks, it has been included in this section as a very important indication of the growing widespread utility of the mobile agent paradigm. From wired to wireless to 3G networks, mobile agents are being used to provide parallel and comparable solutions to the traditional methods in place.

Summary and Conclusions

Resources in the network are a prized commodity, and require judicious management and control. With the ever-increasing growth rate of networks, resources are also getting scarce, and more in demand. There is a demand for better and faster resources in terms of bandwidth, storage space, and processing power. Quality of Service parameters have been introduced and implemented in networks. Users are being given the option of paying more to receive better service and not have to wait for resources. However, this does not always guarantee the required resources and services.

Mobile agents have been found immensely useful in the area of resource management. They can be ubiquitously used to provide QoS, and also to locate and allocate resources in the network. Resources may sometimes need to be redistributed/re-allocated for uninterrupted and superior service, which agents can perform easily. Agents provide flexible resource management by implementing the required service policies rather than adhering rigidly to policies that are embedded in the network components. In addition, mobile agents provide fault tolerance (see Chapter 5) in being able to react immediately to sudden events in the network, and re-route or re-allocate resources to accommodate as many connections as possible.

However, it has been observed that most of the resource management and control functionalities have been implemented by multi-agent systems, using static agents, resident on network components, rather than mobile agents traversing the network. Intuitively, it seems as though mobile agents would provide a more efficient system to control resources in the network, as they can traverse the network and search for available resources, rather than having to depend on bidding, and responses from other agents.

Various methods for resource redistribution, resource re-allocation, and resource location have been designed and implemented in the past. These algorithms rely on passing messages and exchanging information to enable adequate resource redistribution. A natural extension to these methods would be to use mobile agents to implement the same algorithms, and compare the results. Further research and work in the area of resource management using mobile agents is imperative, considering the vast amount of resources and the equally high demand for them in networks today.

CHAPTER 4

ROUTING USING MOBILE AGENTS

Introduction

Routing is the basis of all network operations. Routing refers to the exploration of and subsequent deciding on a path from a source node to a destination node in the network. Effective routing can reduce delays in the network, and improve the performance of the network. The decision of a route or path is made by the network depending on the routing algorithm used.

There are several routing algorithms used in networks, and they select and store paths from one node to another. Each node contains a routing table with information about routes from it to any other node on the network. Shortest path algorithms are used to find the shortest path from a source to a destination [44]. The paths could either be the shortest available in terms of number of nodes traversed, or the cheapest in terms of cost per link traversed.

Routing can be viewed as either a global or local task, depending on whether the nodes have partial or complete knowledge of the state of the network. Further, it can be centralized (one node computes the paths and broadcasts the same) or decentralized (each node performs its calculations iteratively), static (routes do not change very often) or dynamic (the network topology and/or loads change often, causing routes to change) in nature. Routing in networks is, in general, based on either of two algorithms - Distance Vector and Link State. There are also routing algorithms inspired by biological ant-routing techniques.

Optimal routing algorithms attempt to provide superior network performance, while maintaining cost effectiveness. There is a recognized need for dynamic routing given the heterogeneity and dynamism of networks today. Adaptive routing, while being flexible and useful in conforming to frequent changes in the network, may also cause routing loops at the nodes. Given the growth of networks and the need for fast and efficient routing, researchers are turning to mobile agent systems to facilitate routing in networks. Mobile agents have been used to implement distance vector, link state, and ant routing algorithms with considerable success.

The Routing Information Protocol (RIP), based on the distance-vector algorithm, and the Open Shortest Path First (OSPF), based on the link-state algorithm are the protocols that are used for routing in the Internet today. These protocols frequently cause bottlenecks as they find the shortest path between a pair of nodes and then proceed to route all packets on that path, thus leaving other paths relatively unexplored and free.

Distance Vector Routing

Distance Vector routing [119] deals with the discovery of the shortest path from the source to the destination. This is more localized and distributed in nature. The nodes have cost information (in terms of distance) of the preferred or shortest paths through each of their neighboring nodes to all destinations in the network. If the node is unreachable, the cost is set to infinity. These costs are used in computing the shortest path and the most likely successor node from the current node to a destination node. The nodes periodically broadcast their routing table information to each neighboring node (see Figure 4.1). The nodes then compare the 'costs' in the routing tables, and choose the minimum cost shortest path.

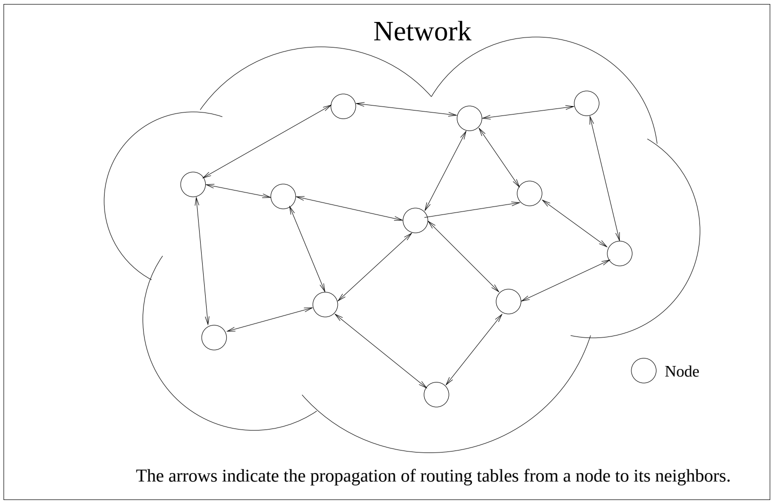

Network
Node
The arrows indicate the propagation of routing tables from a node to its neighbors.

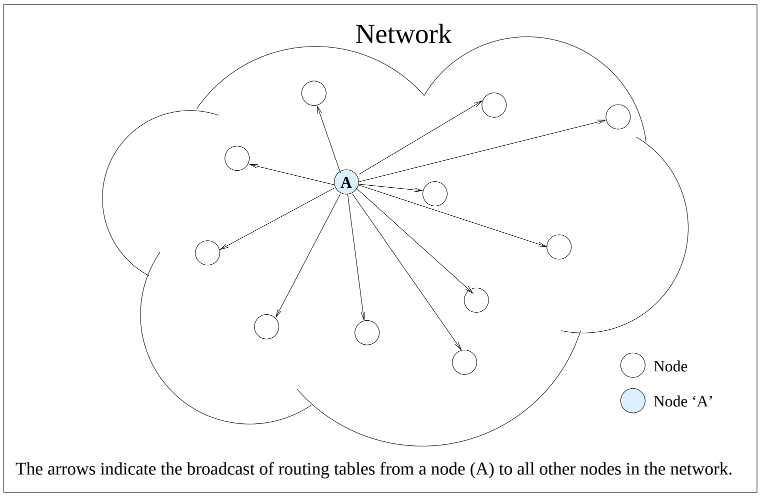
Figure 4.1: Distance Vector Routing, where nodes send their routing tables to only neighboring nodes

Nodes implementing the Distance Vector Algorithm exchange route and link cost information with only their immediate neighboring nodes. Each node follows this method and the information is thus propagated throughout the network. All Distance Vector algorithms are derived from the basic Bellman-Ford [12, 68, 136] algorithm. The Routing Information Protocol (RIP) [89] is based on the Distance Vector algorithm.

Link State Routing

Link State routing is a centralized algorithm, where each node maintains a network-wide view of the link costs. The nodes periodically *flood* the network with these link costs in order to keep them consistent and updated. The link costs are then used to compute the shortest paths to all destinations. The shortest path to a destination is calculated using the Shortest Path First (SPF) [13] or Dijkstra's [57] algorithm.

In the event of an update or change to a link in the network, all nodes are informed through a Link State Advertisement which is integrated in each of the nodes' routing tables to reflect the current state of the network. Thus, there is a network-wide broadcast or flooding of the topology of the network (see Figure 4.2).



The arrows indicate the broadcast of routing tables from a node (A) to all other nodes in the network.

Figure 4.2: Link State Routing, where nodes send their routing tables to all other nodes in the network

Ant Routing

Ant routing in networks is based on biological models of ants’ behavior in foraging for food. Ants discover the shortest path between their source and a destination, by communicating with each other indirectly through stigmergy [81].

Ants deposit small quantities of pheromones while moving towards the destination. Other ants follow the pheromone trails and will naturally take the path where the pheromone levels are high. The level of pheromone indicates how recently the ant passed by, i.e., the stronger the pheromone trail, the shorter the time elapsed since an ant passed through. Pheromone trails are reinforced by the backward movement of ants from the destination to the source. A shorter path would therefore contain

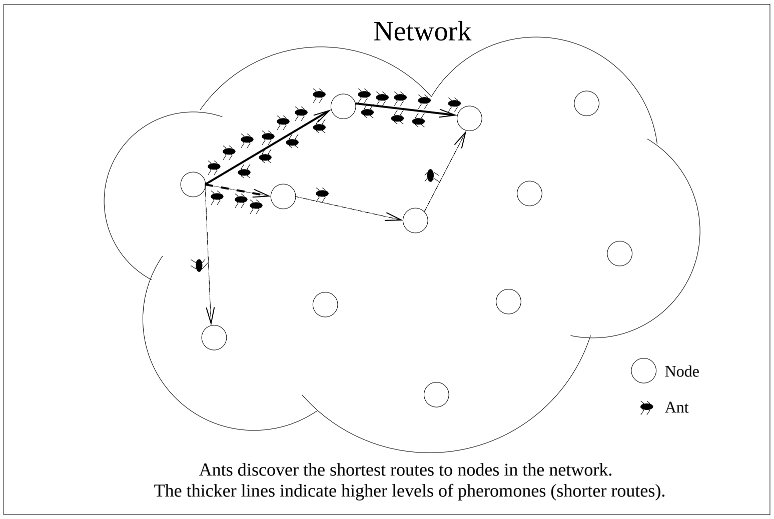

Network
Node
Ant
Ants discover the shortest routes to nodes in the network.
The thicker lines indicate higher levels of pheromones (shorter routes).

Figure 4.3: Ant Routing, where ants lay pheromone trails to discover the shortest routes in the network

almost double the quantity of pheromone than the longer path (see Figure 4.3). Thus, a pheromone trail helps ants discover the shortest path to the destination.

Mobile Agents for Routing in Networks

While each of the above algorithms perform several degrees better than static routing methods, there is nevertheless a lack of a more dynamic routing mechanism that can adapt easily to network changes, and propagate only the updated values in routing tables. Mobile agents are now increasingly being considered to undertake routing tasks such as updates to routing tables, as they are mobile, flexible and can maintain state.

This section examines the use of mobile agents in routing. Communication networks require efficient and scalable routing mechanisms in order to maintain a high level of performance. While distance vector and link state routing have been traditionally used quite effectively in networks, they are increasingly becoming unwieldy to scale in terms of memory in growing networks, and do not react as quickly as required to changes in the network. Therefore, mobile agents have been considered as a viable solution to implement either of distance vector or link state routing algorithms. Ant routing is considered parallel to mobile agent routing, where the ants are actually pieces of mobile code (agents) carrying out routing operations.

The following papers evaluate the use of mobile agents to facilitate routing in networks. The papers are segregated in terms of the type of routing conducted - ant routing, distance vector routing, and other routing algorithms.

Ant Routing with Mobile Agents

Routing requires efficiency, stability and robustness. In addition, a good routing algorithm should be able to provide the required Quality of Service (QoS). The discovery of the shortest path from a source node to a destination can be a time-consuming and tedious procedure, causing excessive resource usage. The foraging activities of biological ants has been modified and implemented to facilitate routing in networks.

AntNet: A Mobile Agents Approach to Adaptive Routing

Resource: AntNet: A Mobile Agents Approach to Adaptive Routing, by Gianni Di Caro and Marco Dorigo, Technical Report, 1997, [34].

This paper presents a network model in which mobile agents mimic the path discovery of ants to discover routes in the network. A datagram network with irregular topology and no congestion or admission control is modeled for simulation. The AntNet system consists of two divisions of homogeneous agents - forward and backward ants or agents. The agents sense the environment and react accordingly.

The AntNet algorithm consists of the ants or agents roaming the network at regular pre-defined intervals, finding routes from the source to a random destination node. Figure 4.4 depicts the movement of forward and backward ants in the network, as they find routes.

▷ Mobile agents (referred to as *forward agents*) are sent from each node to a random destination at regular intervals. Each agent stores information about the nodes it has visited and the time taken to visit each node.

▷ The next hop node is selected by the mobile agent based on the routing table at that node. This selection is made choosing a node proportional to the probability (*goodness*) of each neighboring node, or at random selecting any node

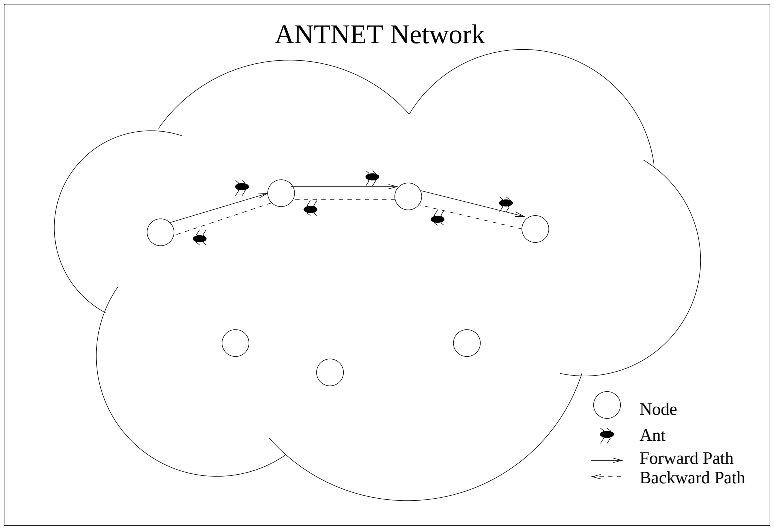


Figure 4.4: AntNet Routing, where forward and backward ants discover routes in the network

with equal probability.

- ▷ If the selected node has already been visited by the agent, the agent clears its memory of all the previous nodes it has visited, since this path is not feasible (a loop in the path is indicated).
- ▷ Upon reaching the destination node, the agent creates a backward agent, and transfers its memory to the new agent.
- ▷ The backward agent traverses the path of the forward agent in the reverse direction.

- ▷ The backward agent updates the routing table and the list of trip times for visiting a node, at each node.

The trip times at each node are important as they indicate the distance of the node from the source node. This distance is not only measured in terms of number of nodes between the source and the current node, but also in terms of the transmission capacity of the links, the processing speed of the intermediate nodes and also the traffic in the network. Thus, congested links would indicate large delays and trip times, causing the agent to decrease the probability in the routing table of that node being selected as the next hop.

The indirect communication between agents serves to provide them with global network knowledge, and facilitates network routing. While each agent handles the routing problem for only one source to one specific destination, the updates in the routing table and visitation times made by the agents help other agents also to select their routes effectively. This form of indirect stigmergy is essential to achieve global network routing information.

The AntNet system was compared with traditional algorithms such as the Bellman-Ford (BF), Open Shortest Path First (OSPF), Shortest Path First (SPF) and SPF_1F, a modification of SPF. The BF algorithm was implemented with dynamic metrics for link costs, and the OSPF algorithm was reduced to a “simple static shortest path calculation". The SPF algorithm was implemented as the prototype of link state routing, with link costs evaluation and network wide flooding. The SPF_1F algorithm was implemented similar to the SPF with flooding of the routing tables limited only to the neighboring nodes. A Daemon algorithm was also implemented, which was believed to be close to an ideal algorithm. This algorithm functioned based on the presence of a *daemon* that would be able to obtain information on the state of all the queues in the network, and accordingly calculate link costs and shortest paths at any given point. The results for traffic distribution in the network indicate that the AntNet algorithm performs the best of all algorithms, save the Daemon algorithm. AntNet maintained the system stability and gave cause to very small delays.

It is clearly evident that some form of communication between the agents to exchange routing information vastly improves the performance of the system. Routing is made much easier and more effective in the presence of adaptive routing techniques rather than with rigid algorithms. The use of mobile agents in the AntNet was logical as the agents mimic the behavior of ants closely. The updating of the routing tables is the form of stigmergy that takes place in this algorithm, to allow the agents to communicate with one another to bring about effective routing in the network. However, AntNet has been deployed in static networks, and has yet to be implemented and tested in dynamic networks.

AntNet in Data Networks

Resource: AntNet Routing Algorithm for Data Networks based on Mobile Agents, by Benjamin Baran and Ruben Sosa, [10].

This paper presents modifications to the AntNet routing system described above [34]. These modifications have improved the performance of the system, and enhanced routing in the network.

The ant chooses a neighboring node as the next hop, dependent on the probability value in the routing table. However, if this node is one that has already been visited, then its probability is reduced in the routing table, and another node is selected at random. If the second node has also been previously visited, then the agent deletes all prior information it carries about the nodes it has visited.

Further, if the agent is within a set time threshold, it goes back to the process of selecting at random a node to visit next based on the probabilities. If the time threshold has been exceeded, the agent kills itself. This ensures that there are no cycles, or infinite loops in the routing path.

This paper proposes several modifications to the AntNet algorithm presented in [34].

▷ One of the modifications is that the routing tables be initialized using previous knowledge of the network topology.

▷ The paper also suggests that the routing tables be modified intelligently in the case of link failures, or when a failed link is available again to be used.

▷ A noise factor was introduced to distribute the probabilities in a truly random manner, and enable the agents to explore new and possible better paths “by accident”.

▷ The paper suggests two different methods of choosing the next hop node, and a probability of 0.5 in selecting one or the other. The first method is to randomly select any node present in the entry for the destination node in the routing table, and the second method is to select a node with the highest probability in the routing table. This dual probability selection method is introduced in order to make the system as probabilistic as possible, and to preempt deterministic behavior.

▷ The number of agents or ants was considered to be a computational and congestion load on the network. Therefore, the paper proposes to limit this number to a maximum of four times the number of nodes in the network.

▷ A very important modification in terms of the destruction of an agent that is lost in transit, or has no means of returning to the source has been included in this paper. Any backward agent that cannot reach the source node due to a link failure is destroyed immediately, as its data is now outdated and of no value in the network.

A number of simulations of the AntNet with the modifications were conducted on different types of networks. The *instantaneous packet delay, average packet delay, instantaneous throughput* and *average throughput* of the system were measured.

Instantaneous packet delay is defined as the average packet delay for all data packets successfully routed at a given instance of time.

Average packet delay refers to the average delay of all data packets routed successfully during the length of the simulation time.

Instantaneous throughput of a network is the total number of packets successfully routed at a given instance of simulation time.

Average throughput of a network indicates the total number of packets routed successfully during the entire time of the simulation of the algorithm.

In a simple network simulation, the AntNet was found to perform much better and produce significantly better throughput than RIP and OSPF, as AntNet is capable of finding alternative routes to destinations. RIP and OSPF however used the same routes once decided upon, causing congestion, and possibly bottlenecks. On the flip side however, AntNet packets experience a slightly larger delay as some of the packets are routed along longer paths.

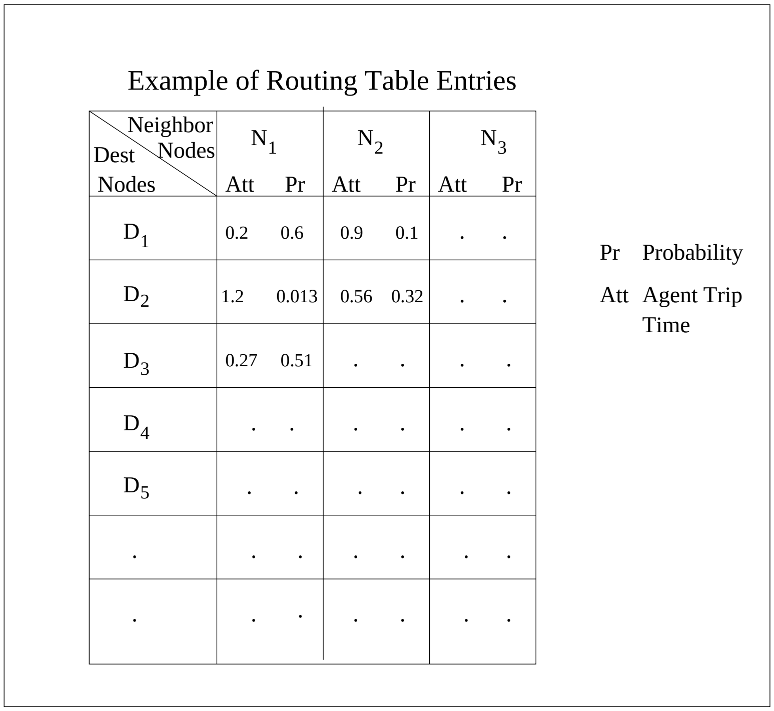
Given link failures and a changing network, AntNet with the modifications incorporated performs in a much superior manner than RIP or OSPF. The initialization of routing tables and the intelligent selection of next hop nodes lends flexibility and a high performance ability to AntNet. In the event that a link failure is observed, the throughput of the AntNet algorithm is not drastically affected, as it is robust in its ability to detect a failed link and accordingly modify its route discovery process. However RIP and OSPF were found to completely deteriorate in performance, with respect to the network throughput, when a link failed.

The AntNet algorithm implemented with the modifications suggested in this paper has proved to be robust and flexible. It has performed better than the traditional RIP and OSPF, and also the original versions of the AntNet [34, 35, 37, 36]. The modified AntNet seems to be a feasible option to be implemented in networks for the purpose of adaptive routing. An improved AntNet algorithm with provisions for congestion, flow control and admission control has been expected to be developed to facilitate its commercial implementation in large networks.

Ant-like Agents for Routing in Telecommunication Networks

Resource: Routing in Telecommunication Networks with “Smart” Ant-like Agents, by Eric Bonabeau et al, [24].

This paper presents a mobile agent system that dynamically updates routing tables at the nodes in a network. The algorithm presented is an extension to the one originally presented in [172], and very similar to the algorithms summarized above.



Example of Routing Table Entries

Neighbor Nodes Dest Nodes	N_1		N_2		N_3	
	Att	Pr	Att	Pr	Att	Pr
D_1	0.2	0.6	0.9	0.1	.	.
D_2	1.2	0.013	0.56	0.32	.	.
D_3	0.27	0.51	.	.	.	.
D_4	.	.	.	.	.	.
D_5	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.

Pr	Probability
Att	Agent Trip Time

Figure 4.5: Example of Routing Table Entries, used by ants in AntNet Routing

In the original algorithm, mobile agents follow a route from a source node to a destination node, based on routing table entries. These routing table entries are updated with the agents' trip times and the probability values of each of the neighboring nodes being selected as the next hop (see Figure 4.5). The probability entries for the neighboring nodes are updated according to the absolute time taken by the mobile agent to reach the current node.

In the dynamic algorithm presented in this paper, *smart* agents [86] are used to perform the updating of the routing tables. These agents use their relative age (relative to the time since they visited the node) and not their absolute age in order to calculate the probabilities of next hop nodes. It was found that this scheme performed better than the one in which the absolute times were considered.

Many more calls were accommodated and routed successfully in the network when the *smart* agents were deployed; these agents performed significantly better when network conditions were dynamic. The *smart* agents are thus adaptable and flexible to cope with changing conditions and enhance the routing of calls.

This paper is an early experiment in using dynamic agents to perform routing in an intelligent manner in networks. There have been encouraging results, and the agent paradigm has been proved to be useful in routing applications. However, the paper has not explored the performance of agents in realistic network models, nor the overhead of deploying agents to perform routing. Too many agents may crowd the network, utilizing resources otherwise required for calls, and too few agents, it has been recognized, would be insufficient to solve the routing problem. The optimum number of agents is yet an indefinite quantity.

Swarm Intelligence for Routing in Telecommunication Networks

Resource: Routing with Swarm Intelligence, Tony White [207].

Ant routing is considered an effective auto-catalytic algorithm to find the optimal route from a source to a destination. The term *autocatalysis* refers to the ability to accelerate reaction by dint of the reaction itself. This paper examines the performance of biologically-inspired ants to perform routing in telecommunication networks.

A point-to-point routing algorithm is implemented using mobile ant-agents to discover routes and allocate resources on them. The agents are deployed to find the shortest path from the source to the destination. The network is represented as a graph, with the nodes as vertices, and the arcs of the graph between vertices represent the physical links between nodes. The links have a pre-allocated amount of bandwidth, which is considered during route discovery.

Ant nests are created at both the source and destination nodes. Mobile agents (or ants) from the nest follow a random path trying to discover a route to the destination.

Available bandwidth is decreased from each link every time a path is discovered and a connection is established between two nodes. An ant chooses a path dependent on the available bandwidth, the cost of the link (defined in terms of the distance or number of hops), and the pheromone level on the link.

There are three kinds of agents deployed in this algorithm:

1. Explorer Agents: These agents explore and discover a path between the source and destination nodes.

2. Allocator Agents: Allocator agents reserve or allocate resources (bandwidth) on the path found by the explorer agents.

3. Deallocator Agents: Resources (or bandwidth) are released on the links of the path from the source to the destination by the deallocator agents.

The algorithm is explained as follows:

```
Each explorer agent explores a path from the nest (either the
source or the destination node), choosing links according to
the pheromone level and cost. Each node visited by an agent is
stored in its memory.
The agents have restrictions on the links they can follow - a
```

node cannot be visited twice in order to prevent the possibility of loops in the route, and there must be enough bandwidth available for the route to be chosen and the call to be established.

When an agent reaches the destination node, it travels back to the source, laying a pheromone trail at each node. The agent follows the same route it took to reach the destination, but in the reverse direction.

When the explorer agent returns, having found a unique path, it transfers the path information to the source node. An allocator agent is despatched by the source node to the destination node. The allocator agent follows the route taken by the explorer agent, and allocates bandwidth on each link that it traverses towards the destination. If bandwidth allocation fails because another agent has utilized the resources, the allocator agent returns to the source node. After the allocator agent reaches the destination, it returns to the source node to establish the connection.

An explorer agent carries with it an updated cost for the route it takes from the source to the destination. If this cost exceeds a pre-defined threshold, the explorer agent is terminated. Further, an explorer agent is terminated if it cannot find a path within a set period of time. The source node generates explorer agents with a certain frequency and continues to do so, until a connection is established with the destination node.

The basic algorithm presented above has been further modified to improve overall performance. Several enhancements have been added to the existing algorithm.

- The selection of links by agents is proportionate to the amount of pheromone on it, and to the cost of the link. Constant cost and pheromone *sensitivity* values have been added to this computation in order to find better solutions. Thus, the selection value is equal to the cost of the link raised to the cost sensitivity (β) times the amount of pheromone raised to the pheromone sensitivity (α). The selection value is multiplied by a random number to normalize this result, and select the next link for the agent to traverse.

$$selectionProbability_j = \frac{(Cost_j)^{-\beta}(Pheromone_j)^{\alpha}}{\Sigma_i(Cost_i)^{-\beta}(Pheromone_i)^{\alpha}}$$

$$selectionValue = Cost^{(CostSensitivity)} * Pheromone^{(PheromoneSensitivity)}$$

- The amount of pheromones deposited by the agents is a deciding factor in selecting the route. Therefore, this amount has been modified to be inversely proportionate to the cost of the link, so that the link with the least cost is selected. Also, the pheromones are evaporated at a constant rate, thus ensuring that outdated paths are not chosen by agents. Paths that are currently being used will always have levels of pheromones deposited by the agents.

- A connection monitor is situated at the source node, and all the agents are originated there. The explorer agents report the cost of the routes taken to reach the destination. After the connection monitor has evaluated the costs, it determines the route to the destination, and sends the allocator agent along the route to reserve resources. It is only when the allocator agent returns to the source node, that the connection is established.

- In establishing point-to-multi-point connections, the agents must reserve bandwidth on each of the routes to the various destinations. It is possible that the routes share a common path partway to the destinations. In such cases, it is optimal to send only one message till that point, before the paths separate, in order to conserve bandwidth.

- Agent species *ids* and pheromone *ids* have been added to identify the different routes and connections. This helps to conserve and reduce search times for new paths in the case of already existent routes, as pheromones of a particular agent species may already be present in the connection.

- Load balancing was achieved by modifying the amount of pheromones on the links by using a cost function inversely proportionate to the usage of the link (used bandwidth/total bandwidth). This ensures that all the available routes and resources are utilized.

- Genetic algorithms [76] have been implemented to enhance the agents. This greatly improved the performance and reduced the time taken to discover new paths.

It was found that agents improved the load balance in the network. There was little standard deviation observed in the occupancy of the links in the network, indicating that every link was utilized to approximately the same degree. However, using an agent species *id* impaired the load balancing as agents tended to follow their own pheromone trails, leaving other routes undiscovered.

Agents occupy space on the links consuming bandwidth. This bandwidth occupied by agents may change depending on the number of agents deployed in the network, adversely affecting traffic and connections if the number of agents is high. An alternative is to allocate a fixed amount of bandwidth on each link of the network specifically for agents executing the algorithm, ensuring that agents always have bandwidth and are not eating into the resources of incoming calls. The agents further continued to look for alternate routes even after having found one optimal path, thereby ensuring load balancing in the network.

This paper has examined a mobile agent system to perform routing in telecommunication networks. The algorithm performs efficiently and further enhancements to the algorithm as described above have improved the performance. The use of swarm intelligence has produced a robust, adaptable and continuous system for routing, which is yet simple. The ratio between the parameters that affect the routing of agents - link costs and pheromone levels - needs to be fine-tuned in order to achieve the maximum output.

In the paper, Connection Management using Adaptive Mobile Agents [209], mobile agents are used to typify ants in finding routes from source nodes to destination nodes. The agents adapt and control the parameters to follow routes based on the success of the previous routes they discover, similar to the technique described above. This algorithm finds and reinforces routes in the network with the least congestion, and is self-adapting to improve performance.

Mobile Agents for Distance Vector Routing

Resource: Agent-based Distance Vector Routing: A Resource Efficient and Scalable Approach to Routing in Large Communication Networks, by K. Amin and A.

Mikler [4].

This paper presents an agent-based implementation of the traditional Distance Vector algorithm to carry out routing in a communication network. The agent system affords scalability, fault tolerance and resource efficiency, as opposed to the difficulty in scaling distance vector based algorithms for routing, and controlling the number of messages exchanged.

Agents are created at different nodes, and each agent carries with it a vector of information consisting of its *id*, the *id* of the source node, the *id* of the neighboring node the agent is migrating to, routing table information pertinent to the node it is migrating to, and a migrating strategy.

Agents migrate to neighboring nodes using the migrating strategy. Upon reaching the adjacent node, an agent performs a computation to verify and/or update the routing table at that node, with the information it carries in its own routing table. When the agent has updated the routing table, it selects the pertinent information therein, and makes it its current routing table, and proceeds to migrate to the next neighboring node.

The agents select only those portions of the routing tables at nodes that could be relevant at the next neighboring node that they migrate to. This strategy ensures that information is not unnecessarily propagated in the network, using precious resources, thus rendering the system more scalable [2].

The migration strategy of agents is considered as being relevant to achieving efficiency. The migration of agents cannot be completely random as it results in degradation of performance of the system. On the other hand, if agents were to carry a history of their routes [131] and then decide on the next node to visit based on this history, their size would get unmanageable, and the overhead of implementing such a system would be considerable. The migration strategy based on the biologically inspired stigmergy of ants [34] while low on system overhead, leans towards a biased path, once found.

Therefore, the migration strategy implemented in this paper combines the advantages of both the oldest-node (history-carrying agents) and ant routing systems. The agents perform a depth-first search initially when they have no knowledge of the network topology. Subsequently, the agents deposit small quantities of pheromones on the edges that they traverse (Edge Pheromone). This pheromone, contrary to the AntNet routing systems, *repels* other agents, thereby ensuring that each agent follows a different path to discover and update routes in the network.

The population of agents is an important factor in maintaining resource efficiency of the system. In Distance Vector Routing (DVR), the number of messages exchanged by nodes is unbounded. However, using Agent-based Distance Vector Routing (ADVR), the agents themselves are the messages to be exchanged and their number can easily be controlled. While a relatively high number of agents in the network implies good routing results, it also means that there will be a large resource overhead. To maintain the agent population at an optimal level, there needs to be a dynamic control algorithm [3].

The paper uses the stigmergetic function of pheromone levels to introduce a dynamic population control algorithm. Each agent deposits a level of pheromone on each node it visits (Node Pheromone). The pheromone is volatile and its level is indicative of whether the system requires an increase or decrease in the number of agents currently populating it. If the level of pheromone is below a set level, another agent arriving at that node will clone itself. On the other hand, if the pheromone level is too high, the new agent terminates itself as it has arrived too soon after the previous agent that deposited the pheromone. No action is taken if the level of pheromone at a node falls between these two levels.

The results of experiments conducted on network simulations comparing DVR and ADVR show that the latter is more scalable as it consumes less overhead in achieving convergence. Further, ADVR maintains a controlled messaging activity in sending agents out with updates to routing tables, as opposed to the broadcasts in DVR. With a controlled agent population however, it was found that all the routes in the network cannot be discovered within a reasonable time frame. DVR functions better in this case, finding all the connected nodes in the network.

This paper effectively shows the use of intelligent mobile agents to achieve routing in a network based on the DVR algorithm. The population and the migration strategy of agents have been considered and feasible solutions are provided with no compromise on performance. The extension of distance vector routing to include mobile agents is yet unrealized completely, and only this agent system has been developed so far to incorporate the distance vector algorithm in mobile agent-based routing. This agent system too, however, borrows from ant-routing to achieve optimal performance in networks.

Routing with Mobile Agents (Non-traditional Algorithms)

In this section, some algorithms that do not conform to traditional routing algorithms are presented. These algorithm all use mobile agents to enable routing in networks. The mobile agents are represented as autonomous entities, traversing the network to perform routing functions.

Dynamic Routing with Mobile Agents

Resource: Cooperating Mobile Agents for Dynamic Network Routing, N. Minar, K. H. Kramer, P. Maes [131].

This paper introduces the concept of mobile software agents to perform routing dynamically in heterogeneous cellular peer-to-peer networks. The use of agents as against object-oriented program has been argued, and a dynamic routing scheme is presented.

In the algorithm presented here, agents traverse the network, learning about routes and edges as they do so. An agent carries with it a history of all the nodes it has visited in the past. Using this history in its memory, the agent updates a node's routing table with the best routes known to it. Agents can only write to routing tables, not read from it; the agents are therefore *dumb* and cannot effect changes in the network.

There are three main steps in this algorithm. The agent first decides where to go from the node that it is currently on. This decision is made based on the history of the agent route, or is purely random in the case of random agents. Once the agent has made this decision, it traverses a route to the neighboring node, learning all it can about the route it takes, such as the traffic, latency, available bandwidth, etc.. The agent updates the routing table at the new node it visits, using the information that it has learned on the way.

A population of mobile agents thus traverses the network, updating routing tables at nodes and providing current information about the network routes. Since the agents depend on their traversal history to make decisions about which node to visit next, this information must be kept current. Therefore, a limitation is placed on the number of items an agent can store in its history. This helps in keeping the agent size within manageable limits. Further, the number of agents also contribute to the overhead on the system, and therefore this number also must be kept within a reasonable range.

Agents can either use their history to make decisions on which node to visit next, or make a random selection. In the former case, an agent would visit a node that it has visited the earliest, or never has visited, thus improving the efficiency of routing table updates. This prevents backtracking and unnecessary frequent repetitive updations of routing tables.

The results show that the network achieves connectivity of 80% soon after the agents begin traversing the network. The system performs better when the agents with history are used as against random selection agents. Further, the overhead of the system should be balanced between the number of agents in the network, and the history size of each of these agents. A decentralized dynamic system such as the one presented here is superior to centralized systems in achieving overall routing in networks.

This algorithm presents a *dumb* mobile agent system in which agents do not have any knowledge, nor communicate with each other or the nodes. Their only function is to traverse the network, gathering information about the edges they traverse, and updating the routing tables at the nodes. There needs to be some intelligence incorporated here so that agents do not function redundantly. Further, security is an aspect that needs to be considered and implemented in the agent system to prevent malicious hacking of agents.

The algorithm however, appears to achieve network mapping [132], rather than routing. There is no information provided on what entries or information is updated in the routing tables of each of the nodes. How do the nodes decide on optimal paths to other nodes? Is congestion in the network avoided? Is the throughput of the network increased? These factors, which are so intrinsic to the routing problem, have not been addressed in this paper.

Routing with Dedicated Mobile Agents

Resource: Routing via Dedicated Intelligent Agents, by Dimitar Lakov, [110].

This paper provides implementation details of a Soft Computing Agent (SCA) to carry out the function of routing in a network. Soft Computing Technology is used to implement the SCAs and the mobile agent paradigm is incorporated to achieve mobility.

The paper views routing as two aspects - the management of traffic congestion, message transfers, queuing and delays, and priority of messages; and secondly, the Quality of Service (QoS) requirements for the transfer of messages. There are two types of SCAs presented in this paper - *information SCAs* which roam the network gathering information about topology changes, storing all the information regarding communication between nodes, and *optimization SCAs* which control the issues such as QoS, message delays and costs related to routing in order to approach optimal performance.

The SCA is constructed on two computational 'levels'. The first level takes as input the global length of message queues, global transfer delay and priority of messages, and produces the transfer time as output. The cost of the path and the bandwidth is also taken into account to calculate the performance. At the second level, the performance and transfer time are used to compute the QoS of the chosen path.

There are a number of parameters such as the cost, priority, security and delay of the message on the path, that can be specified by the user to choose the optimal path. Using these parameters, the SCA chooses the optimal from all the available paths.

This paper presents a generic software model using Soft Computing Technology and exploits the mobility of software agents to find optimal paths in network routing. No specific routing algorithm is implemented here, and the system is adaptable to execute any algorithm to find routes. This is an interesting concept that needs further development and specification, in order to be implemented and considered as a serious alternative to traditional routing methods in the network.

Mobile Agents for QoS in Routing

Resource: Multipoint-to-point Routing With QoS Guarantees Using Mobile Agents, by S. Gonzalez-Valenzuela, V. C. M. Leung and S. T. Vuong [79].

Mobile agents functioning on a WAVE platform achieve routing with QoS guarantees in this paper. The WAVE platform [167] is recognized as being inherently distributed and conducive to network application programming. It therefore provides a suitable base for mobile agents to perform the required tasks.

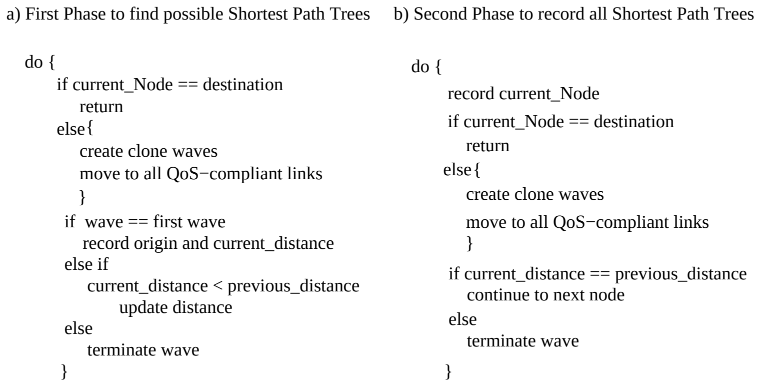
Routing is performed by mobile agents roaming the network in a wave-like movement, merging at one specific node (*egress node*). The movement of agents from one node to all other nodes in the network, before converging at the egress node is referred to as the *wave-like* movement. A knowledge network layer is formed by this wave movement, endowing the agents and nodes with knowledge about the paths taken by the agents. Thus, information is spread and shared in the network by the wave-like movement of the agents.

Information is managed by the waves in the network. The information is classified into two different types of variables - frontal and nodal. Frontal information is that which is carried by the waves, and required to perform computations. On the other hand, nodal information is local to the node it is located on and is accessible to other waves.

The static implementation of the WAVE platform to form shortest path trees for routing is further divided into three parts. In the first part, agents roam the network in a wave movement to find all the possible shortest paths to a given node. All the possible common nodes in the shortest path trees are detected, and finally, one shortest path to the destination is determined.

The agents propagate in the network discovering routes that conform to given bandwidth and number of hops requirements. When an agent arrives at a node, it ensures that no agent from the same source has previously visited the node. The agent stores its ID in the node to indicate its presence. The agent also updates the hopcount to the current node from the source node. If the agent discovers that another agent with a smaller hopcount than its own has visited the node prior to its arrival, it terminates its wave propagation in the network. Otherwise, the agent is cloned into as many agents as there are outgoing links from the current node that meet the QoS criteria. Each of these agents then continues the wave propagation for discovery of shortest paths in the network (see Figure 4.6).

This process of agents following a wave movement to flood the network is repeated to ensure that *all* the shortest paths in the network are discovered. Therefore, when the wave propagation is repeated, each agent checks whether its current hopcount from the source is equal to the one already stored at the node. If it is not, the agent or wave is terminated. Only if the distance is equal, the agent continues to the destination node (see Figure 4.6).



a) First Phase to find possible Shortest Path Trees

```
do {
    if current_Node == destination
        return
    else{
        create clone waves
        move to all QoS−compliant links
        }
     if  wave == first wave
        record origin and current_distance
     else if
         current_distance < previous_distance
                update distance
     else
         terminate wave
    }
```

b) Second Phase to record all Shortest Path Trees

```
do {
     record current_Node
     if current_Node == destination
        return
    else{
        create clone waves
        move to all QoS−compliant links
        }
     if current_distance == previous_distance
        continue to next node
     else
        terminate wave
    }
```

Figure 4.6: Finding all the Shortest Path Trees between a source-destination pair in Two Phases

The second part of the static algorithm determines all the common nodes in the shortest paths. A group of agents propagate through the network following the shortest paths discovered. Each node is marked by the agents, so that other agents recognize that the node is common to the path to the destination. Thus, common nodes are flagged by the agents for data-merging.

In order to determine one shortest route to the destination, the agents traverse all the shortest paths found, assigning weights to the paths they traverse. These weights are directly proportional to the number of common marked nodes that the agents encounter on their paths to the destination. Therefore, the path with the highest weight for a given source node is chosen as the shortest path by default, as it contains the node(s) that most of the paths traverse to reach the destination, indicating that it is indeed the path that meets the QoS requirements and does not exceed the shortest hopcount.

The dynamic implementation of this algorithm involves little change. When a node joins the network, a group of agents is propagated to find the shortest path trees for this node. All the other nodes then carry out the last two phases of the shortest path discovery process. Similarly, when a node leaves the network, a group of agents traverse the shortest paths that include the deleted node. The records of each of the nodes on these paths are deleted by the agents. The agents then perform the second and third parts of the algorithm to find the shortest paths without the deleted node.

The algorithm is efficient in finding shortest path routes that conform to QoS requirements. Further, this algorithm requires little or no computation, and is extremely distributed in nature. There are also no loops or circular paths discovered by the agents in this case, as the hopcount is compared at every node. However, the inherent nature of these wave-like agents causes significant overhead and widespread flooding in the network. This cannot be inexpensive, especially given the bandwidth constraints in networks today.

Routing Management with Mobile Agents

Resource: Routing Management Application Based on Mobile Agents on the Internet2, Angelica Reyes, Ernesto Sanchez, Antonio Barba [159].

This paper describes a routing management application based on Common Object Request Broker Architecture (CORBA) [52], and uses an intelligent agent system to make decisions on routing, depending on the resources available in the network.

Although SNMPv2 routers are used, the application presented here requires the use of a proxy between SNMP and CORBA, as SNMP routers cannot support CORBA.

The management application server informs the edge router about the best point-to-point path that packets need to follow in the network. The edge router then sends the router information, including the source and destination addresses and customer reference, to the application requesting the route.

In order to obtain the best possible path, the CORBA management application needs information regarding the QoS, priority level and Service Level Agreement (SLA) of the packets. All this information is encapsulated in the General Inter ORB Protocol (GIOP). The management application also requires a routing-reference sequence of the core routers to hand over to the edge routers. The edge routers then use this information for routing packets. Routing is based on (Multi Protocol Label Switching (MPLS) [138] and the routers receive updated routing tables based on QoS criteria from the management system.

The mobile agents roam the network collecting information about the monitored parameters for routing, from the static agents resident on the routers. The parameters include packet-loss ratio, bandwidth available, delay average, queue management, times of birth and death of requests, etc.. The management application uses this information to make the appropriate routing decisions. The mobile agents also maintain updated information about the routing tables, so that correct routing tables can be generated for new connections.

The mobile agent system is implemented in the Grasshopper platform [82]. The Grasshopper platform v2.0 is an OMG's CORBA based Distributed Agent Environment. The mobile agent system is used to manage congestion and the traffic in the network, and thereby aid routing decisions. The system contains both static and mobile agents. Static agents contain and maintain the information such as the status of the queues, throughput of the links, delays, packet-loss rate, etc. in the routers. They are therefore used to monitor information, and also to facilitate routing per QoS requirements. The static and mobile agents communicate with each other to establish traffic patterns and to control the congestion in the network. QoS requirements are also tested for by the agents.

The system was implemented with an aim to increase throughput and to aid routing decisions with the use of agents. Thus, the management application was considered an on-off source depending on the existence of mobile agents. The routers were considered as buffers with a processor on-off for agents. The transmission and propagation delays of agents through the links were also considered. The routers send out *stop* agents to indicate an overflow in their queues, and *start* agents if their queue sizes fall below a certain threshold.

The use of mobile agents and the Grasshopper platform ensure that there is an improvement in the performance in the distributed network environment. Further, MASIF [130] and FIPA [66] standards have been maintained to provide a secure and flexible management system. The CORBA application system ensure that the QoS is maintained according to the SLA of the customer. The exact parameters and results of analyses however, have not been presented in this paper. This paper makes use of a number of different application systems, but does not present accurate output parameters. There could be a more detailed explanation of each of the algorithms implemented and how it affects the overall performance in the network as regards routing decisions.

Summary

Routing has traditionally been implemented in networks using routing protocols and algorithms like the Routing Information Protocol, Distance Vector Algorithm or Link State Algorithm. These algorithms have been proved to perform sufficiently well to be the default protocols implemented in networks today. However, there is a distinct lack of scalability and the increasing necessity to improve network throughput to a significant extent, given the rapid growth of networks. This motivates the development of routing algorithms that can accommodate rapidly growing networks, and dynamic route discovery.

Mobile agents have been used in various network applications due to the flexibility in their being applied to different types of applications. The efficacy of mobile agents in routing applications is evident in the inherent distributed nature of the problem. Mobile agents can roam the network and discover and update routes in the nodes in the network to facilitate routing. However, there is a need for a uniform routing protocol to be used, as mobile agents cannot adhere to various different protocols implemented in the nodes of the networks.

The resources consumed by agents in the network, are a trade-off for the benefits of using such a system. The agents ensure a distributed control system that maintain updated information about the traffic and links in the network. In the event of a link failure or reduced bandwidth availability in the network, the agents are always aware of these changes and the routing tables are accordingly updated. Thus, the agents perform efficiently to reduce redundant or ineffectual attempts at routing. Added security to the agent systems would further enhance reliable decision making for the network management system as regards routing.

CHAPTER 5

FAULT TOLERANCE AND SECURITY BY MOBILE AGENTS

Introduction

This chapter examines security and fault tolerance issues in networks, and ways to achieve both with the use of mobile agents. While considering security and fault-tolerance by mobile agents, there is a distinction between the security provided *by* the agents and *to* the agents; similarly, fault-tolerance is achieved for the execution of agents in the network, while also achieving a stable fault-tolerant network environment.

Agents need to be protected during their execution, and the network needs to be secured from malicious hosts, and agents. This level of security towards both agents and the network is achieved by implementing a variety of authentication schemes. Further, the network needs to be stable, i.e., not subject to frequent faults and/or reconfigurations, in order to provide an accurate picture to the agents, in order to aid their execution. Also, in the event of any failure, both the network and the mobile agents need to be able to recover their states in order to continue functioning without any major interruptions in service. This fault-tolerance is achieved by constant monitoring and by implementing back-up schemes.

Fault Tolerance in Networks

Networks are require a certain degree of fault-tolerance for the provision of uninterrupted services. Fault tolerance has been defined as *the use of protective redundancy to permit continued correct operation of a system after the occurrence of specified faults* [141]. Protective redundancy is *the use of extra hardware, software, information, or time to mask faults or to reconfigure a fault system* [141].

Faults in networks can cause a number of problems, causing deteriorated performance and decreased throughput of the various nodes in the network. Further, if faults are not detected and rectified, there may be incorrect information and updates passed to the nodes of the network, causing confusion and errors. Therefore, it is necessary to have a system wherein faults are detected early and properly dealt with so as to maintain the working status of the network.

Further, the issue of fault-tolerance is divided into fault-tolerance in the network, and fault-tolerance for mobile agents. The former is brought about by the use of mobile agents roaming the network and detecting faults in the networks. Fault tolerance can only be achieved with the location of faults and thereafter by correcting them. Mobile agents, due to their mobility, make it relatively easy to detect faults and thereby facilitate the almost-continuous provision of services in the network.

Fault-tolerance for mobile agents is the maintaining of the state of the mobile agents, so that they continue to function normally to carry out whatever tasks they are performing in the network. In providing fault-tolerance for agents, the network is also rendered fault-tolerant, as agents carry with them the state of some parts of the network. Further, cloning of agents provides the required redundancy of information in achieving fault-tolerant mobile agents and networks.

The faults that could occur in networks or agents have been identified as:

- Node failures, wherein a node either temporarily or permanently goes down, thus causing a loss of all information
- Agent components, such as the communicative system between agents, or the agent directories could fail, causing reduced or incorrect functionality of agents
- The mobile agents themselves could completely fail owing to computational or network errors
- The network could fail either in part or as a whole
- There could be faulty transmission of messages in the network, causing incorrect or loss of messages.

There are several different models for fault-tolerance in systems, and these models may be used in agents too to promote fault tolerance. Among these are primary back-up, exactly-once and sliding window models. Each of these models is used to prevent the collapse of operations in a network, and to achieve a certain degree of fault tolerance [210].

In the primary back-up model, one server is elected as the primary server, and the back-up servers maintain the exact same information as the primary. When the primary server goes down to some reason, one of the back-up servers can immediately take up as the primary without causing too much inconvenience and delay in the network.

Agents in the exactly-once model communicate with the nodes and are transferred from node to node in the network exactly once to prevent duplication of information exchange. This is achieved by encapsulating the agent and its code and data, and placing it on a message queue to be transferred to the next node. In the event of a failure, all the information is lost, and the system returns to the state prior to the collection of information by the agent.

In the sliding window model, agents maintain a backup of the data on a limited number of each of the hosts that they have recently visited (see Figure 5.1). When an agent moves to the next host, the oldest backup at the first visited host is deleted. The agents thereby maintain a limited number of backups. Moreover, the information in the backups is recent. Because the backups are maintained along the path of the

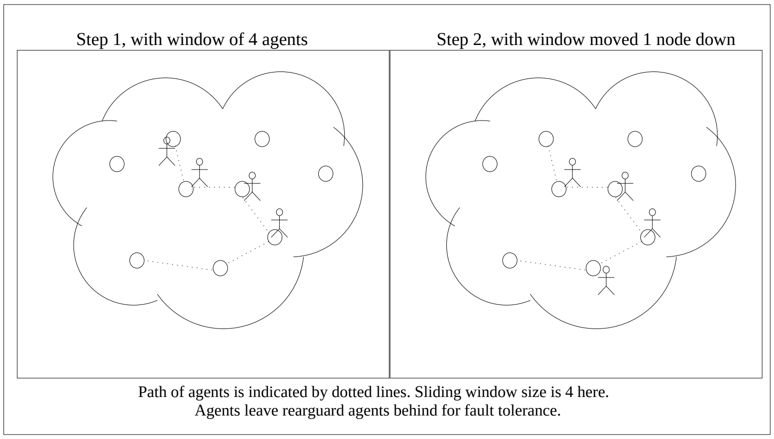

Step 1, with window of 4 agents
Step 2, with window moved 1 node down
Path of agents is indicated by dotted lines. Sliding window size is 4 here.
Agents leave rearguard agents behind for fault tolerance.

Figure 5.1: Sliding Window Model for Fault Tolerance

agents execution, this is referred to as a sliding-window model. The back-ups help to maintain the current state and promote fault tolerance of the system.

Fault Tolerance by and for Mobile Agents in Networks

This section presents a number of papers and work that has been carried out to achieve fault tolerance in networks. The network itself is rendered fault-tolerant to a certain degree by the aid of mobile agents providing backups, or alternately, the mobile agents are provided with various mechanisms to preserve their state in case of failures.

Fault Tolerance in Mobile Agents

Resource: Fault Tolerance Issues in Mobile Agents, by Qiao Xin, Yang Yu, Yu Xu and Zhanhai Qin [210].

This paper looks at the different models of fault tolerance explained above, before propounding its own model for providing fault tolerance to mobile agents in the network.

The problem has been defined as that of a mobile agent visiting several hosts in the network, accessing different services, and then returning to the source with the results. There needs to be a solution in the event of the failure of a host. Further, the mobile agent must be able to continue with its tasks in accordance with the specifications of the originating requesting host. There therefore needs to be a fault-tolerant model in order for the agent to carry out its tasks uninterrupted and for the requesting home node to receive its results as per the requirement.

The first model proposed is called the *naive home-based model.* Every time that an agent visits a host and accesses a service, it updates its own state or memory contents. Further it sends a copy of its updated state back to its home. The obvious drawback to this model is the necessity of both the agent and the home node to be online constantly and in communication with each other. The home node may also become a potential bottleneck in the event of frequent updates from the mobile agent.

The first model is slightly modified to provide an *agent-based model.* In this model, one machine or node is designated as the proxy home node. The agent then communicates with this machine, and sends its updates to this node instead of to the home node. While the home node remains relatively free and has the flexibility to go offline in this model, there still needs to be a fault-tolerant solution for the proxy home node.

The third model proposed in this paper is that of an active mobile agent with sliding window. A group of machines is designated as the *mothership*, wherein the primary backup model is used to provide secure backup for the mobile agent. Thus, the mobile agent only communicates with the mothership upon accessing a service, and the mothership maintains a copy of the updated state of the agent. In the event that the mothership is not notified by the agent after a service has been accessed, it assumes that the agent has encountered a fault, and therefore another agent needs to be re-created.

Since this model implies an increase in traffic in the network, there could be an added specification of the time interval between the agent sending updates to the mothership. The traffic in the network could further be reduced by introducing a sliding window model in the agent's route. The agent leaves *rearguard* agents behind upon accessing a service, and before proceeding to the next host. This ensures that even if the agent encounters a problem, the rearguard agents have the updated state, and can continue execution from that point. Information that the rearguard agents have is also conveyed to the mothership to maintain a synchronized state.

This model has proved to be the most fault-tolerant by far, as the mothership is an abstraction to the mobile agent, thus leaving the agent free to execute its tasks, rather than focus on fault-tolerance issues. The model is also quite flexible as the interval values and the sliding window sizes can be specified as required. Lastly, the fault-tolerance aspect is addressed quite effectively as the mothership can launch a new mobile agent to continue execution from the point of failure of the previous agent, with little or no delay.

This paper has addressed the issue of achieving fault-tolerance in the execution of mobile agents in the network quite effectively. Backups have been provided in the event of failure, and almost every possibility of failure has been prempted with a counter mechanism. However, the overhead in providing so many backups in both the mothership and the sliding window rearguards has not been considered at all. In an already over-burdened network, it may be impractical to provide for many rearguards. The number of machines in the mothership to provide for backups needs to be limited too. Further the time interval of the agents reporting back to the mothership with the updated states may not be sufficiently controlled. If this interval is too large, there may not be enough of fault tolerance. At the same time, if this interval is too small, it would mean that the agents increase the traffic in the network by almost constantly updating the mothership.

This paper would have definitely been more significant if it had provided overhead estimations and results of performance. However, it has provided extremely useful insights into the fault tolerance issues of agents in the network.

Fault Tolerance for Mobile Agents

Resource: FANTOMAS : Fault Tolerance for Mobile Agents in Clusters, by Holger Pals, Stefan Petri and Claus Grewe [150]

The model presented in this paper aims at providing fault tolerance to mobile agents. The paper provides a background to the existing fault tolerant mobile agent systems, and highlights the deficits in each. New methods are proposed to overcome the inherent vulnerabilities in existing systems and to create a more fault tolerant mobile agent system. background on the existing fault tolerant mobile agents systems, also denounces them as insufficient and introduces its own approach.

The concept of rearguard agents introduced in the TACOMA [102] project does not facilitate network partitioning. In this project, an agent leaves behind another agent with its current state, which can take over in the event of the failure of the original agent. In Minsky's model [133], an itinerary is maintained, and agents and their results are sent to each of the next-step nodes. Thus, if an agent and its result do not arrive at a particular node according to the itinerary, it may be concluded that the agent has failed in its execution, and another agent could be introduced into the system. However, this model can only function with a fixed itinerary, and is not flexible; further, the agents cannot communicate with other agents, and thus distributed computing is not supported. A voting protocol is introduced in the Mole system [186] by Strasser and Rothermel, in which the itinerary can be changed by agents; however, this model has the added uncertainty of the agents' location in order to provide for fault tolerance.

This model therefore recognizes a need to correct node or agent failures, and provide support for agent communication. The goals that the model aims at achieving are support for agent communication, autonomy, user transparency, efficiency, portability and reusability. The model provides fault tolerance as an option, and as little change to the existing hardware and operating system as possible.

Recovery of systems can be achieved with either forward or backward recovery protocols. Forward recovery of systems avoids the re-execution of computations, and therefore can be faster than backward recovery. However, it involves considerable overhead to ensure fault-free execution, and hence this paper uses backward recovery to implement fault-tolerance. The Domino Effect [155][1] also must be avoided, and independent checkpointing of the states of the mobile agents is preferred as opposed to coordinated checkpointing to prevent this effect.

Messages need to be logged so that they can be regenerated if required in the event of a fault. It is also considered more efficient to log messages upon receiving them, rather then when sending them, as it results in faster recovery of messages. Agents can recover and/or prune the message logs autonomously without interaction with other agents.

This paper presents a fault-tolerance model for agent based systems, and uses independent checkpointing techniques with receiver-based logging of messages to implement fault tolerance. Agents are used to store message logs and the checkpointed state of the system. Each mobile agent, also called *user agent*, for which fault tolerance is required, has a *logger agent* associated with it. The *user agent* and the *logger agent* form an agent pair that perform the required tasks and enhance fault tolerance in the network. The *logger agent* is not an active computational model, and requires little CPU power. The *logger agent* and the *user agent* must never reside on the same node at any given point of time, so as not to be susceptible to the same faults. The *user agent* communicates its state and messages to the *logger agent* by exchanging its

[1] The Domino Effect is observed when the backward recovery of checkpoints is propagated to the initial state of the system due to inter-dependence of checkpoints.

communication unit with it, before conveying the same to the local node. Thereby, the agents monitor each other constantly, and in the event of a fault, one of this pair of agents can rebuild the other from its own state.

The *logger agents* are created by serializing the state of the *user agent* similar to the migration technique. Further, checkpointing is done at appropriate intervals, most conveniently at migration times, as this involves a serialization of the agent state. This approach ensures a stable system in the event of node or even agent failures, and is not dependent on a predetermined itinerary of the agents.

The analytic evaluation of the FANTOMAS (Fault Tolerant Approach for Mobile Agents) based on a Markov model is provided [165]. A cluster consists of n nodes, each of which contains an agent. The agent can detect a failure of the system and itself fail in the event of physical failure of the node, or failure of the runtime environment. When a failure occurs, all affected user and logger agents will be recovered using the message logs, and checkpoints previously stored. The system is considered to be in an insecure state during the recovery period. If there are additional failures during this time period, the system cannot be sustained. When the system is fully recovered, it is considered secure again, and functions as normal.

The recovery of the system involves only the recovery of the agent states. It does not signify the physical repair of the nodes, or of the runtime environment. Mobile agents are regenerated with the stored checkpointed states, and it is only the mobile agent system that is used as a fault tolerant approach to the failed system. It was found that the FANTOMAS approach tends to provide a more reliable system, in proportion to the size of the system, as compared to a simple system.

Fault Location in Networks using Mobile Agents

Resource: Distributed Fault Location in Networks using Mobile Agents, by Tony White, Andrzej Bieszczad, Bernard Pagurek [208].

This paper explores fault tolerance in networks using swarms of mobile agents, and the coordinated communication between these agents to detect faults. A biologically inspired model of agents and agent communication is used to achieve the required level of efficient communication between agents, and to locate the faults in the networks using this inter-agent communication.

The paper presents a Mobile Code Environment (MCE) for the execution of mobile agents, and for agent-based management of networks (this agent model has already been described in Chapter 2, and is briefly re-visited here). The MCE consists of a Mobile Code Daemon (MCD), a Migration Facility (MF), an Interface to Managed Resources (IMR), a Communication Facility (CF), and a Security Facility (SF) [16].

Further, there are three principal types of agents introduced in this paper -

servlets, deglets and netlets. Servlets are resident on servers and migrate to the servers only to provide extensions or upgrades to the servers. Once they migrate to the server and are installed, they remain on that server, and migrate no further. Mobile agents that are delegated to perform a specific task and migrate to certain localized areas of the network to perform that task are known as deglets. Agents that circulate in the network and perform predefined tasks on a permanent basis are known as netlets.

The agents in this paper sense their environment and act accordingly, on the basis of stigmergy [81]. The agents themselves are not innately intelligent; their intelligent behavior is a consequence of their indirect communication with one another, upon sensing changes in the environment. Sign-based stigmergy is the principle of agent communication in this paper, wherein agents leave traces of their visits on the paths they follow, thereby enabling other agents to detect their presence and/or changes in the environment. This form of foraging behavior of ant-like agents has been previously documented and utilized in other network routing applications too [34] (this form of routing was considered in Chapter 4).

Agents in this paper have been described using a tuple consisting of five components - emitters, receptors, chemistry, a migration decision function, and memory. An agent's receptor enables it to sense the environment, and the emitter allows it to leave behind changes to the local chemical environment. The agents must follow some rules associated with the receptors and emitters and also compare the local states of their memories before attempting to change the environment. Thus, a series of chemical reactions is initiated before any change can take place in the environment.

An agent may migrate randomly or in accordance with a chemical reaction brought about by the emitters of other agents. The migration decision function, a part of each agent, is used to make the decision of migration either probabilistically, or deterministically, in the direction of increasing concentrations of other agent emitters. Random migration is allowed in order to allow the network to balance concentrations of agent populations.

The detection of faults in the network is carried out by the agents communicating with each other. There are different classes of agents identified, each of which performs a pre-defined task.

Condition Sensor Agents (CSA) are agents that specifically test and measure parameters associated with certain components, and determine whether a specific condition, for example, the utilization of links or a node, holds true or not. They are examples of netlets, and perform only these testing and measuring functions in the network. CSAs have the ability to adapt to the network based on the values that they ascertain from the network components. For example, it is more likely that a

CSA would visit a component where the condition, such as node under-utilization implying a problem, evaluated would be true, and avoid one where there is no information of interest available. CSAs can also leave chemical traces behind on the components where they detect faults, in order to allow other agents to make the necessary diagnostic changes.

Service Monitoring Agents (SMA) and Service Change Agents (SCA) have been defined to monitor and mark the services that require attention respectively. The SMAs are static and reside on sources of service providers in the network, for example at the source of a Private Virtual Circuit (PVC). When the SMA detects significant changes in the monitored service provided by a PVC, a Service Change Agent is sent to those resources in the network on which the service depends, to mark them with a chemical message. The concentration of this chemical message indicates the significance of the change in the service, and if the change is beneficial, a negative concentration of the chemical is left behind. This negative concentration of the chemical indicates a positive change, whereas positive traces of the chemical indicate detrimental changes in the service. Therefore reinforced or new traces of the chemical indicate deteriorating services.

Problem Identification Agents (PIA) are netlets that roam the network and determine the location of faults in the network with the help of the chemical traces left by the SCAs. These agents determine the exact location of the faults and initiate diagnostic activity so that the network is not incapacitated due to these faults. There can be several different types of problem agents in order to tackle the network load. For example, Chronic Failure Problem Agents and Overload Problem Agents identify those services in the network (due to inordinately high concentrations of chemicals) that experience multiple faults in a short period of time or that are over-utilized respectively. The Overload Problem Agents can also be used to aid in the re-configuration of the network so as to decrease the load on these services.

In the place of using the traditional alarm systems, this paper has identified and classified different types of agents to aid in the detection of service failures, and determination of fault locations in the network. The communication of agents is based on a successful biological model and significant overhead has been reduced by rendering the agents unintelligent. The agents can only act intelligently due to their stigmergetic communication. The paper however, elaborates on the necessity of having a network model in real time using agents as described here, to provide accurate results. An intelligent network model capable of interaction with the netlets and deglets would enable scalable solutions to issues in network management.

Security by and for Mobile Agents in Networks

Providing security in networks is a delicate issue, as it means not only detecting malicious elements in the network, but also pre-empting attacks and preventing intrusion and eavesdropping. Both intentional and unintentional attacks need to be countered, and the system rendered secure.

Work on Security in Networks by and for Mobile Agents

There has been considerable attention devoted to the issue of security in networks. Given the widespread use of networks and the increase in the number of users, security has become more of a pressing concern of late, and it is imperative that both the users and the network be rendered secure against malicious attacks. Cryptography and the use of public and private keys have been explored as options to provide foolproof systems, and there is still ongoing work to ensure complete privacy and attack-free environments in networks.

The use of mobile agents has also made the security factor more important, as it involves the introduction of what is viewed as foreign code in the execution environment. The networks therefore, need to be able to protect themselves from sundry attacks and be available to only trusted mobile agents. At the same time, mobile agents also need to be provided the guarantee of risk-free execution on the network hosts that they settle on. Thus, mobile agents need to be secured against malicious hosts and other potentially malicious agents; network hosts also need security against mobile agents that could be malicious. Therefore, security provided by the mobile agents to networks is distinguished from the security provided to the mobile agents themselves. Further, the security provided in the networks needs to be constantly tested in order to ensure that the security mechanisms have not been corrupted. The papers presented in the following sections present research concepts in solving the security issue in networks that use mobile agents.

Network Security Testing Using Mobile Agents

Resource: Network Security Testing Using Mobile Agents, by T. Karygiannis [106].

This paper presents security testing mechanisms in the network using mobile agents, as opposed to the traditional central security testing mechanisms in place. The security provided in the networks is checked at frequent intervals to ensure that security mechanisms are functioning as they are supposed to. Mobile agents have been used to promote distributed security testing in the network. With a centralized security testing controller, there is bound to be error, as the controller node could fail, be corrupted or receive corrupted information from a compromised node, thereby disseminating incorrect information to the rest of the network. If the central controlling security node fails, then the network security on the whole is at risk.

Security checks are conducted after regular intervals of time. These checks could be conducted by host-based monitors, or by the centralized security testing controller. Host-based monitors also pose their own risks in terms of security. If one host-based security node is corrupted, and sends out incorrect information to the central security manager, this incorrect information is broadcast to other hosts and managers in the network. Further, if the central manager has also failed, then the information from the host-based security monitor is never received, nor broadcast in the network. This form of security control is not very secure, nor fault-tolerant. It also is not easily scalable, as an increase in the number of nodes, or in the amount of data directly implies an increase in the interval of time between security checks conducted by the monitors, thereby exposing the network to the risk of attacks for longer periods of time.

This paper has used mobile agents to enforce security in the network, given the limitations of the centralized and host-based security monitors. Mobile agents are suited to autonomously travel through the network, and also hide information from potential intruders. Since there are a number of agents in the network, the network is not susceptible to one single point of failure. Agents can clone themselves to accommodate the addition of new nodes to an increasing network.

Agents are used to traverse the network, migrate from node to node, and test the security of the nodes. If an agent detects a problem with a host, it can report the problem to another host, and with the cooperation of other agents, help to reconfigure the network, in order to render the compromised host secure again. Upon reconfiguration, the network hosts are brought into a secure state to protect their information. Network services may be refused to an insecure host to prevent further corruption in the network, until it is deemed safe again by the agents.

Agents have an itinerary of hosts to visit and check. The agents communicate with each other and with *meta-agents*. The *meta-agents* are responsible for the generation and dispatching of agents to the hosts. The agents transmit alarms in the event that they discover a host that is potentially a security hazard. Agents communicate via proxies in order to render their own communication secure. Intruders are thus prevented from entering the network, by the constant vigil and inter-communication between mobile agents.

Agents need to be protected from malicious hosts, and other malicious agents. The network hosts also need to be protected from malicious agents. However, this issue is not covered in this paper. This paper solely discusses the protection of the network nodes through the use of mobile agents. It has introduced mechanisms to do so, and the inherent mobile and autonomous communicative quality of agents is exploited to perform constant security checks and preventive maintenance on the network nodes.

The paper outlines the advantages of using mobile agents to ensure security in the network as:

Fault tolerance is improved with the use of mobile agents, as the network is not vulnerable to a single point of failure. There are several mobile agents in the network that can provide backups in the event of a security breach. Disabling security tests and monitors is not an easy task when they are distributed through the network. Mobile agents help circumvent this problem as they can traverse the network and detect and correct problems in security. Mobile agents improve the parallelism of security testing by distributing the tests over the network.

The use of mobile agents renders the security testing more scalable as agents can be cloned or terminated as required. Security test upgrades can also be performed quite easily with the use of mobile agents. Mobile agents can not only be used to test nodes for intrusion and errors, but also to reconfigure and recover the state of the system. Thus they are flexible. Agents can be used on heterogeneous computer systems, due to their inter-agent communication system, and still perform well.

There are relatively few disadvantages of using mobile agents, and these are mostly due to the recent nature of the agent paradigm. Mobile agent systems may yet have flaws that could render them inefficient and unsuited for use in large networks. The security testing tools are quite large and add considerable overhead to the mobile agents. The platforms for the execution of mobile agent code must be installed on each of the nodes before the agents can perform their tasks. Further, it is the very mobility of agents which, although quite appealing in its advantages, may however pose a security threat by exposing the agents to wandering hazards.

This paper has effectively outlined the use of mobile agents in providing security testing features in network nodes. While fully exploiting the intrinsic qualities of mobile agents, the paper has addressed the probable drawbacks of using the mobile agent paradigm. There is a recognized need for added security to the agents themselves, to aid the provision of security in the network.

Agents for Intrusion Detection

Resource: Intelligent Agents for Intrusion Detection, by Guy G. Helmer, Johnny S. K. Wong, Vasant Honavar and Les Miller [90].

This paper introduces an intelligent agent system for intrusion detection and countermeasures in a network. The work carried out here is based in part on previous experiments conducted on security and intrusion detection in networks [185, 56, 69, 139]. Intelligent agents are employed to apply a data mining approach to the network activities in order to detect intrusions in the network.

In previous work, the Distributed Intrusion Detection System [139] uses multiple local area and host monitors to monitor the system and network activities. The activities are then reported to a central director that aggregates all this information to determine whether there is an attempt at intrusion. The Computer Immunology project [69] developed a notion of *sense* in computers to detect intrusions foreign to the normal state of the system. The Java Agents for Meta-Learning (JAM) Project [185] uses distributed intelligent Java agents to learn and detect models of fraudulent behavior in the network.

The agent model developed here is a distributed and intelligent model similar to the JAM model. This model detects attacks depending on previous system call traces stored in memory. Thus, if the system call traces are found to differ significantly, an intrusion is detected. A system model of agents is developed to implement the prototype intrusion detecting system [54]. The goals of the system are defined as:

- Using individual agents in subsystems to learn to detect intrusions on hosts and in networks.

- Using mobile agents and agent technology to intelligently process data at sources to detect intrusions.

- Using inter-agent communication to share information and thereby determine when to make the system more secure or when to relax extra vigilance.

- Using agents to apply data mining techniques to the data gathered in order to identify and react to attacks on systems.

This model based on data mining is especially significant as it can identify multi-stage attacks on the entire system, initiate countermeasures to the attacks, and also provide the necessary documentary evidence for the system administrators to initiate action against the attackers.

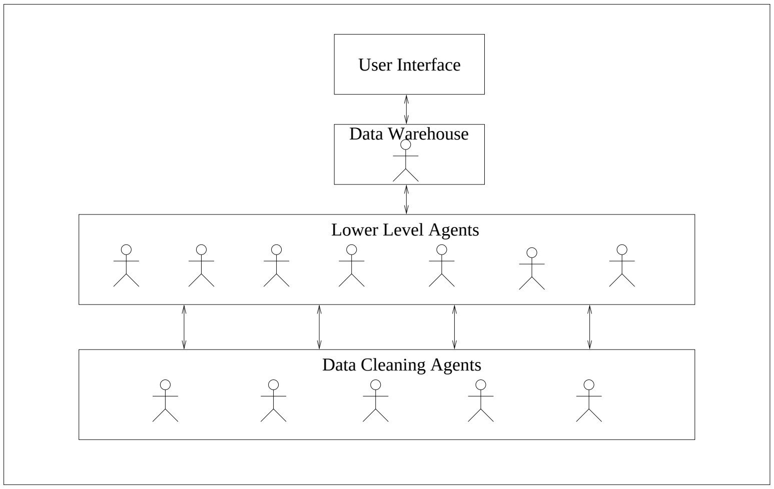


Figure 5.2: Architecture of an Agent-based Intrusion Detection System, adapted from [90]

Data cleaning agents process and store information at the lowest level. A higher level of agents classify this data and determine whether the activity associated with the data is normal or not (see Figure 5.2). The agents here use a variety of data classification algorithms depending on the data to be processed. At the top level, agents combine this data with some knowledge of network activities, and apply data mining algorithms to determine whether there has been suspicious activity, suspicious associations, and suspect frequent events or patterns. This level of agents provides a global view of the system and aids the system administrators in not only defending the system from attacks, but also in developing protective strategies to identify attacks.

The prototype intelligent agent used in this paper was created using the system call traces and data provided by the University of New Mexico. System call traces can be used to identify the use of privileged programs and to determine whether there is any anomalous use of these programs against a system [69]. This is the method that has been employed in this paper to determine whether or not an attack is being mounted against a system. Intelligent agents are used to gather and store system call traces and this data is combined with the knowledge in data mining algorithms to identify attacks.

While this paper has introduced an intelligent agent approach to intrusion detection in networks, it has borrowed extensively from other work previously conducted and has lent the agent approach to these models. The agent paradigm has been effectively used to detect security breaches in the network, and the inherent distributed and mobile nature of intelligent agents has been exploited to promote security. This model has been based on existing data mining and comparative algorithms in order to make it simple to implement. However, the overhead associated with the implementation of this model needs to be examined.

Distributed Intrusion Detection

Resource: A Framework for Distributed Intrusion Detection using Interest Driven Cooperating Agents, by Rajeev Gopalakrishna, Eugene H. Spafford [80].

This paper presents a framework for distributed intrusion detection in networks by agents that communicate with each other, purely on the basis of expressed interest. There is no hierarchy of analysis, and the agents communicate with each other using the framework developed.

Hierarchical intrusion detection systems were found to have considerable disadvantages, in that there was a hierarchy involved in data analysis, which requires that data would be analyzed at all levels, and all the modules would require changes in the event of an attack. Further, data refinement, and the communication of the filtered data to all levels in the system is purely circumstantial and it is difficult to determine what information is to be conveyed to higher or lower levels in the system. If analysis is to take place at all levels, this implies the presence of large and bulky analysis modules at all levels of the system, causing not inconsiderable overhead in terms of

CPU and memory usage, and limited fault tolerance. The communication between the levels is passive and static at best - there is no analysis-based querying, or further intelligent communication between the levels.

This paper introduces agents that are communicative and dynamic, and that request events and alerts from other agents based on the system state. The agents are therefore dynamic and respond to the system changes, instead of blindly assessing and reporting all system changes at all times. This reduces the overhead of the system as the agents can actively request information only about those events that interest them, instead of accessing all information at all times. Agents can hence be used for either host-related or network-related security.

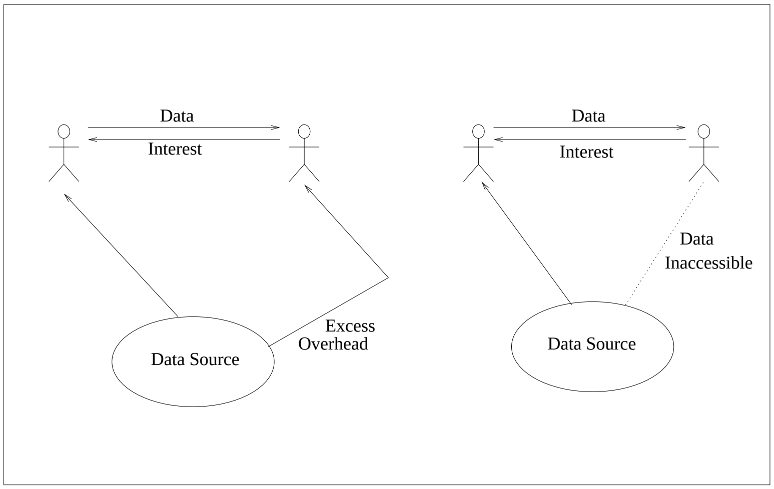


Figure 5.3: Data transferred between agents upon expressing interest due to a) more overhead and b) no access to data, adapted from [80]

In this paper, the *interest* of an agent has been defined as *"a specification of data that an agent is interested in, but is not available to the agent because of the locality of data collection or because the agent was not primarily intended to observe those data"*. Agents may express interest due to a locality of data, in the event that obtaining the data from the source involves more overhead than obtaining the same data from another agent that already has access to it. An agent may also evince interest in data in the event that it does not have access to some data and requires that data in order to determine intrusion (see Figure 5.3). Providing agents with data only as and when required is a significant advantage of the system proposed in this paper, as agents are not inundated with unnecessary information at all times. Data is defined as either events or alerts; an event is data that has not been analyzed, and an alert is the result of an analysis of events that indicates a potential security hazard.

Agents are required to communicate interest in data; however, an agent cannot know whether that data is available to another agent or not. Therefore, a hierarchy of propagation of interest in the network is introduced to facilitate inter-agent communication. This hierarchy is divided into local, domain and enterprise levels. Further, interests may be classified as directed or propagated interest depending on whether or not the agent knows the specific host or domain from which it requires information respectively. Agents may evince interest in data restricted to a specific level - local, domain or enterprise only. Interests may be classified based on the duration for which the agent requires the data. Agents could specify permanent interest in data for the duration of their existence, or a temporal interest in data for only a specific period of time.

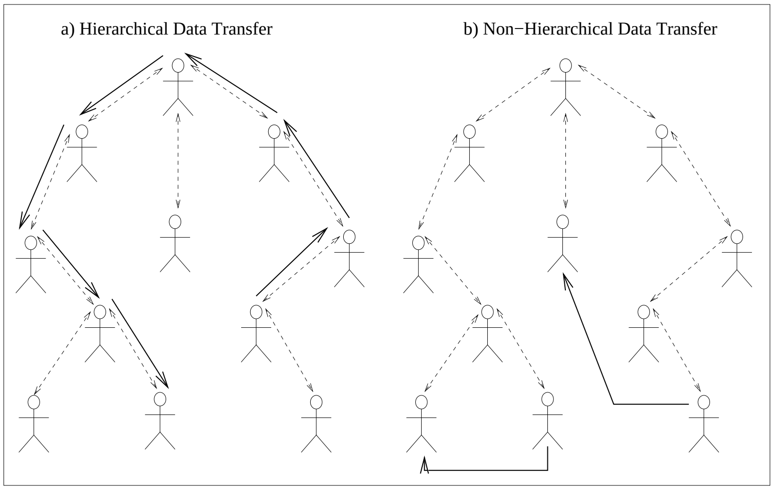

a) Hierarchical Data Transfer
b) Non−Hierarchical Data Transfer

Figure 5.4: Data transferred between agents in a) a hierarchy b) a non-hierarchy, adapted from [80]

The delivery of data requested by agents could either be carried out in a hierarchical or non-hierarchical (direct) manner (see Figure 5.4). The hierarchical transfer of data could mean that in the event of failure of modules, data is not available to agents requesting them. However, a distinct advantage of using the hierarchical communication model is that duplicate data are not sent and re-sent to different agents requesting them along the same paths. In general, the hierarchical model is more scalable in terms of agent communication links maintained in the network, as opposed to several individual agent-to-agent links all over the network in the non-hierarchical model, since there are fewer agent-agent links that need to be maintained.

The main components of the architecture proposed in this paper are an agent registry, an interest registry, and a propagator. Agent registries are present in each host that can execute agents in the network. The registries maintain information about the agents, the events that the agents collect, and the alerts generated by the agents. The interest registry on each host is used to maintain a record of all the interests that not only originate but are also currently being serviced by the agents on that host. The propagators are used to transfer interests and data between the different hierarchical interest levels. Propagators are connected to all entities (propagators and agents) in the level below them, and exactly one propagator in the level above.

The communication between agents has been explained and the model for data propagation and detection of intrusion has been discussed in detail. Security concerns for the agents themselves however, have not been addressed. The duplication of propagators on different levels to introduce a degree of fault tolerance has been suggested. The paper, while explaining that agents can evince interest in, and therefore gain access to data and analyze the same, needs further elaboration on what processing of data takes place before an agent sends out an alert. The algorithms used for the analysis of data, and for the actual detection of intrusion have not been put forth. There is also a lack of performance analysis of this model on the network and on the overhead involved. However, the model proposed in this paper presents a reasonable method of evaluating data and only conveying data of interest to the network components.

Distributed Transactions for Mobile Agent Security and Fault Tolerance

Resource: An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions, by H. Vogler, T. Kunkelmann and M.-L. Moschgath [203].

This paper presents an architecture to provide security for mobile agents and for the hosts in a network, while guaranteeing a certain degree of fault tolerance for the agent system too.

The issue of fault tolerance for an agent has been equated to that of a guaranteed and correct agent transfer problem. An agent must be transferred to its destination correctly and must be initiated at the new host with the same state. Further the agent must not be duplicated at any point of time, thereby causing no redundancy of information. This requires a certain transfer semantic, and the *exactly-once* semantic has been guaranteed to deliver the agent state correctly exactly once with no duplications, and also provides for recovery after a host crash. This is possible with persistent storage of the agent's state, and a protocol for distributed transactions.

Transactions have long been accepted as the solution for the management of large quantities of data. Properties of transactions have been identified as Atomicity, Consistency, Isolation of parallel data access and Durability (ACID). Access of distributed resources within a transaction is referred to as a distributed transaction. A two-phase commit protocol is employed in a distributed transaction. This form of distributed transactions has been implemented in this paper alongwith additional security to provide a fault tolerant mobile agent system.

Trust between the agent and the host has to be established in order for the system to function efficiently and smoothly. A third party with complete knowledge and access to the system, has been introduced to aid the trust process between the agents and hosts. This concept is similar to that introduced with the Kerberos model [142]. The trust service also logs data about the agents and the hosts in the network, such as the IDs of the agents and the hosts, and the time interval between agents' visits to the hosts. This ensures consistent data and also accountability in case of a breach of trust.

The protocol for agent migration, and therefore agent security has been outlined with a few assumptions. Each agent is assumed to possess a certified public key [105], and every host and agent in the system is assumed to be registered with the trust service. When a user requests a secure service, the agent system comes into play in the following manner:

- A new agent is created and introduced to the trust service, with a unique ID. The agent's ID is used in conjunction with the transaction that the user requests, and the communication between the trust service, user and agent is protected by public key encryption.

- A newly initiated agent must find another target host to migrate to. A special trader for agents contains information concerning resources and services and conditions for agents at various hosts. This information is used by the agent to locate a suitable new target host.

- Once a contract has been established between the new host and the agent, a copy of the agent is sent to the new host. Distributed transaction guarantees that the agent's state is preserved and consistent. Encryption of the agent state provides the required security. The protocol for secure transfer of agents is given as (see Figure 5.5):

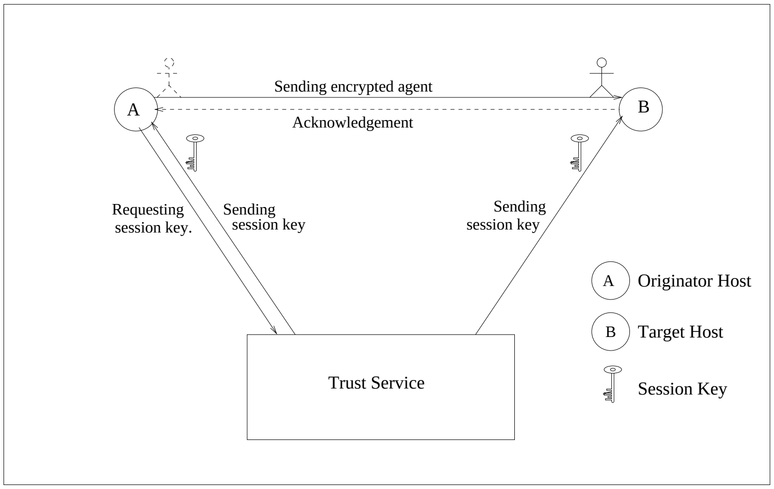


Figure 5.5: Protocol for the Secure Transfer of Agents between hosts

1. The originator host identifies the beginning of a transaction between the trust service and the target host.

2. The originator host requests a session key to enable the secure transfer of the agent to the target host.

3. A session key is generated by the trust service and sent to the originator and to the target host using the public key protocol.

4. The agent is encrypted using the session key and is then transferred to the target host by the originator.

5. The target host decrypts and initializes the agent, and acknowledges receipt of the agent copy.

6. The originator host concludes the two-phase commit protocol, and deletes the old agent, updates the trust service with the data regarding the new agent, and also invalidates the current session key.

- Once the agent is successfully transferred, it can be involved in transactions with the user and the secure facility.
- An agent is terminated upon conclusion of the user service, and the appropriate information is logged in the trust service.

This model ensures that both the agent and the host remain secure through all the transactions that are conducted in the network between different entities and users. All relevant information is logged and no agent can be introduced in an untrusted host. Encryption ensures that only trusted entities have access to the agent and therefore it is deemed secured against any attack. The distributed transaction two-phase commit protocol ensures fault tolerant transfer of agents. The trust service provides a control and management mechanism to monitor agents in the system, and the persistent logs ensure that no data is ever lost, and all attacks can be traced eventually.

The paper presents a highly detailed secure and fault tolerant agent model for use in networks. Encryption and session keys have been used to ensure security of both the hosts and the agents. There are however, no specific results presented in terms of performance. However, the paper proposes an extremely viable model for agent security in networks.

Security for Network Management Mobile Agents

Resource: Security Requirements for Management Systems using Mobile Agents, H. Reiser, G. Vogt, Proceedings of the Fifth IEEE Symposium on Computers and Communications: ISCC 2000, July 2000 [158].

This paper considers aspects of security for mobile agents that are deployed for network management. Management systems use mobile agents for more distributed functionality. This presents the problem of security as it can be dangerous to execute any piece of code arbitrarily. Therefore, any action of agents is made possible only on an agent platform. Further, access to data by agents should also be controlled and coordinated. Only trusted agents should be able to access secure data.

The security aspects considered here are those concerning the agents themselves, the agent systems, the managers, and the managed resources in the network. Each of these is a potential source and destination of attack. Further, the relationship between each of these entities is also a target for attack. Any two entities exchanging messages are subject to a breach in security. Entities may form an execution relation with each other, where an entity, such as a manager or agent system, executes another entity, such as an agent system or agent respectively. Calling relations also exist between entities, where an agent calls certain interfaces in order to interact with the agent system.

Attacks are usually defined where the attacker attempts to 'masquerade' as a valid entity to gain secure access to the data. Attackers may also eavesdrop on messages and use the information obtained to attack at a later date, or convey this information maliciously where it is not supposed to be conveyed. Denial-of-service is another form of attack, where an agent or an agent system, is denied service or execution. Attackers may use several different methods to attack the message communication between different entities. They may alter messages, misuse the resources of a management system, stop messages from being delivered and also trace an agent and prevent it from executing.

Several methods may be implemented to incorporate security in agent systems. However, most of these are application-specific, and need continuous upgrading to cope with new attacks. Therefore, an architecture needs to be developed and adapted to suit the security requirements of a mobile agent system. This architecture needs to provide authentication of agents, authorization and access to agents, confidentiality of data, integrity of agents and agent systems, non-repudiation of services, resource constraints to prevent misuse and provide a protected environment (*sandboxing*) to control flow of resources and agents.

In the agent architecture developed by the authors, the manager signs an agent when it initiates it and supplies the agent with the necessary rights. An agent system is responsible for the decryption and authentication of agents. The agent system checks the signature in the agent and authenticates the sender. The agent system can determine the level of security required for the agent based on the level of encryption used. Only after an agent is authenticated, it is authorized by the agent system. The authorization process is dependent on the previous path of the agent and its authentication. If an agent visited any known malicious hosts on its path, its rights are removed. The agent is executed in a sandbox only after it is authorized by the agent system. Frequent checks are performed when the agent is executing to ensure that the agent is running only valid processes. After execution, the agent is packed with its current state, and signed by the agent system. If additional rights are required, they are assigned and the agent system sends the signed and possibly encrypted agent to the next link.

The paper presents a secure mobile agent framework to perform network management functions. Requirements for agent security are first defined, and then a suitable architecture is designed based on these requirements. Existing agent architectures are evaluated and improved on to provide the security features discussed in this paper. While this paper is incomplete in presenting a successful implementation of a secure framework for mobile agent systems, it is definitely a systematic and well-thought out approach to the ever-existent problem of providing authentication to mobile agents in a mobile agent system. More work is required to make this architecture completely fool-proof and available to be used in wide-spread mobile agent network management systems.

The MAMAS (Mobile Agents for the Management of Applications and Systems) [11] provides a secure environment for the management of networks, services and systems, by providing for the integrity and authentication of agents and hosts. Authentication of agent signatures, and encryptions are used to provide security. MAMAS introduces the concept of a 'closing property' in networks, which is defined as the property by which a network can be closed off to outside possibly malicious, attacks. The network is thus restricted, and any outside entity can be easily identified, and tagged as either malicious or not, by the implementation of certain security policies. An 'opening property' is also defined, by which the network can be allowed to access outside components, and external usage of resources is permitted, provided by interoperability features.

The SOMA (Secure and Open Mobile Agents) [5] architecture is built on a similar model, wherein a hierarchy of abstractions is provided to facilitate mobile agent security. There are security and authentication procedures required of agents from an unreliable source, and constant monitoring ensures that the data carried by the mobile agents for network management functions are not altered, while they migrate in an untrusted domain.

Summary

Mobile agents have been long since accepted as the alternate solution to a variety of problems in distributed systems. While different agent architectures have been developed and implemented, they are all open to further extensions to include added features. Among these extensions is fault tolerance and security for both the agents and the hosts of the networks. Mobile agents can be considered for use in a wider range and number of applications only with the inclusion of security and fault tolerant features [203].

It has been recognized that there are no guarantees regarding agent delivery or transfer of agent data in a network. A network host could fail, the agent could itself fail, or the network as a whole could be error-prone. Under such circumstances, there are no reliable recovery mechanisms in place for agents to cut their losses and continue execution. Agents could also be attacked by malicious network hosts, or turn malicious themselves to attack the network hosts or other agents. There need to be sufficient security mechanisms in place to recognize such rogue agents or hosts, and take corrective or preventive action.

There has been considerable effort devoted in recent years to the issues of fault tolerance and security for and by agents in networks. Various protocols and ideas have been explored and implemented, and there is still work in progress to achieve secure agent models that are fault tolerant, with the minimal overhead and cost outlays. While security for networks has been an overriding concern for many years, of late, security for agents and agent systems has also become an important consideration. Several papers focus on the issue of providing secure mobile agent systems in networks [211, 166, 186, 93]. Security for agents is important, and coupled with fault tolerant models, agent systems will become more foolproof for use in the various applications that have been detailed in the previous chapters.

This chapter focuses but briefly on these issues of security and fault tolerance, as extensive research and study is required to do justice to them. They are a standalone area of research and merit not just one chapter, but an entire thesis work by themselves. However, this work would have been incomplete without a mention of these concerns regarding mobile agent security and fault tolerance.

CHAPTER 6

CONCLUSIONS

Introduction

Mobile computing has become popular, both in the client-server and the mobile agent paradigms. Mobile agent systems have advanced tremendously in the past decade to gain acceptance in a wide number of applications. Agent systems prevalent today provide sophisticated routing and resource management functionalities, in addition to secure interfaces for communication between entities, etc. The client-server paradigm has all but given way to mobile agent systems, owing to the latter's flexible, communicative attributes.

Mobile Agents Vs Client-Server Paradigms

The client-server paradigm relies on the transfer of data necessary for computation between the server and the client. In the mobile agent scenario, the code itself is transferred to the data, thereby curtailing bandwidth requirements and alleviating latency problems in the network that large amounts of information could cause. As a consequence, mobile agent systems perform relatively faster and more efficiently in computing results and transmitting the same. It would not be unreasonable to deduce that mobile agent systems would perform significantly better than client-server computations only when the amounts of data required to be transferred are considerably large. The two mobile computing systems otherwise perform in a comparable manner for small quantities of data to be computed, as it would not make much of a difference whether the bandwidth requirement is say, 1500 bytes of data from the server to the client, or by 1500 bytes of mobile agent code to be transferred.

Mobile agent systems have however, found considerable use and popularity in many applications these days. They present an intelligent alternative to the mere transfer of data to be processed. The agents are capable of interacting with not only the computing environment but also other agents, so as to determine the requirements of the system and dynamically perform operations to achieve goals. Agent systems endowed with the ability to migrate from place to place have been acknowledged as powerful forces in performing various tasks and achieving targeted goals.

In spite of the growing popularity of mobile agent systems, researchers have found it necessary to compare and contrast the existing paradigms with that of mobile agent systems to justify their use. The IBM report, 'Mobile Agents: Are They A Good Idea?' [45] presents a comprehensive picture of the mobile agent paradigm, and a comparative analysis of the benefits of using mobile agents as against existing protocols. Danny Lange and Mitsuru Oshima [111] briefly enumerate 'Seven Good Reasons for Mobile Agents', stating that mobile agents reduce network load and latency, can adapt dynamically, execute autonomously and encapsulate protocols, among other attributes. Ismail and Hagimont [96] have evaluated the performance of mobile agent systems against the client-server model and found that despite the slightly increased cost of implementing the mobile agent systems, they evinced improved performance over the client-server models. The advent of mobile agents in telecommunications has been analyzed by Magedanz, et al. [118], and it has been established that while mobile agents add significantly to the performance of systems, they need to be incorporated alongwith the existing computing systems in order to be considered improvements rather than afterthoughts.

With many protocols already established in networks today, it will be a while before mobile agent systems are accepted wholeheartedly as the standard and deployed network-wide. Researchers have developed new and improved models of agent systems; however, the acceptance of these systems as the primary medium for functions in networks is yet unrealized. Change is always slow in coming, and while mobile agent systems have made a significant breakthrough in systems today, their widespread acceptance and use is yet a distant possibility.

The use of mobile agent systems in different applications in networks was considered and their performance analyzed in this thesis. The purpose of this review is to help establish the very real power of mobile agent systems, and the possibility of their far-reaching application in networks.

Work in this Thesis

In the previous chapters, the use of mobile agents has been reviewed in various network-centric applications, specifically those of network management (Chapter 2), resource management (Chapter 3), routing (Chapter 4), and fault tolerance and security (Chapter 5). Each of these areas requires complex procedures and computations in order to maintain the smooth functioning of the network, and to prevent undue congestion, latency and to circumvent link- or other failures. Mobile agent extensions to algorithms already in place in networks have proved to satisfactorily achieve the functionalities mentioned above, and indeed, have improved the performance of networks considerably.

Network Management

In Chapter 2, several mobile agent systems for network management were considered. Each of these systems focused on the computation of Management Information Base (MIB) variables at the network elements themselves, rather than transmitting these values to the network manager node. The delegation of gathering the information from the network elements, performing the required computations, and transferring only the absolutely requisite information to the network manager is adeptly carried out by mobile agents.

Mobile agent platforms have been developed for the creation, execution, communication and migration of agents for network management [154]. Agents can implement policies and adapt applications dynamically in the network to provide better service. Different agent communication languages have been developed and used, and ontologies have been defined for these mobile agent systems. Researchers in the mobile agent field have tried to exploit every area possible, from adapting the network components, to data repositories, to network links. They have taken every factor into consideration and fine-tuned the mobile agent systems in order to improve their performance in network management.

Resource Management

Resource management presents a slightly more difficult problem for mobile agent systems to solve. Agent systems have been deployed for the management of resources and the scheduling of tasks in distributed systems. Chapter 3 presents several agent models that have been developed to achieve resource management in selected applications. However, these agent models rely on static agents that are resident on the network components, and do not migrate in the network. Further, the agent systems have been implemented in distributed computing environments, and not at large in networks as yet.

Mobile agents, by dint of their attributes, would be immensely useful in the area of resource management. They can be ubiquitously used to provide QoS, and also to locate and allocate resources in the network. Resources may sometimes need to be redistributed and/or re-allocated for uninterrupted and superior service, which agents can perform easily. Agents provide flexible resource management by implementing the required service policies rather than adhering rigidly to policies that are embedded in the network components. In addition, mobile agents provide fault tolerance (see Chapter 5) in being able to react immediately to sudden events in the network, and re-route or re-allocate resources to accommodate as many connections as possible. There is a steady inroad into the use of mobile agents in resource redistribution and management, and these projects need to be researched and rendered viable to be considered feasible in networks today.

Routing

Routing has presented the most fundamental of all functions and related problems in networks. Without an efficient routing algorithm, networks would be redundant. Network nodes must obtain not only the shortest route from a source to a destination, but also one that has sufficient resources, and little congestion. With the increasing heterogeneity and growing size of networks, route discovery and maintenance of the routing tables in the network components present a considerable challenge to the existing routing protocols. It is evident that dynamic algorithms are required to meet the routing desideratum in networks.

Mobile agent solutions have been proposed to discover and maintain routes in networks. In Chapter 4, some mobile agent models for routing in networks have been presented. Most of these models are based on the biologically-inspired ant routing mechanism, and have been found to perform effectively. Mobile agents are a natural implementation of ants roaming the network discovering routes and updating the routing tables in the various network components. However, no algorithm has been proposed yet for the extension of the link-state routing algorithm to mobile agent system model, and there is only one algorithm that extends the distance-vector routing algorithm to mobile agent based routing.

The efficacy of mobile agents in routing applications is evident in the inherent distributed nature of the problem. Mobile agents can roam the network and discover and update routes in the network nodes to facilitate routing. However, there is a need for a uniform routing protocol to be used, as mobile agents cannot adhere to various different protocols implemented in the nodes of the networks.

Fault Tolerance and Security

In each of the functions discussed above, it is implicitly imperative that the system continues to perform reasonably well even in the event of failure. Faults or failures in networks could be the lack of resources at a location, the loss of a link, congestion, delay, etc. The effect of each of these conditions on the overall performance of the network must be controlled. Systems in the network that achieve various tasks such as routing, must be able to provide alternate solutions in the face of shortcomings of existing solutions. Such fault tolerance is difficult to achieve as it would involve the storage of iterative information, thereby occupying precious memory in the system.

Another pressing concern in networks is the provision of secure interaction between entities. Networks are inundated with requests from different users from numerous locations, and the network components need to be able to accede to each request with full faith. It is highly probable that malicious entities roaming the network could attack or corrupt the network components and the information stored therein, thereby further corrupting the chain of ongoing communication with other entities. Security therefore, is an exigent condition to enable foolproof functioning and communication of all concerned parties in networks.

Mobile agents have been considered as the apt means to promote fault tolerant and secure networks. Due to their mobility and ability to store state, mobile agents can be used to checkpoint the state of the network, and to restart from wherever the fault occurred. Further, due to the agents' ability to clone themselves, several copies of their memory contents are available at different locations of the network. However, it has been recognized that there are no guarantees regarding agent delivery or transfer of agent data in a network. A network host could fail, the agent itself could fail, or the network as a whole may be error-prone. Under such circumstances, there are no reliable recovery mechanisms in place for agents to *cut their losses* and continue execution.

Agents could be attacked by malicious network hosts, or turn malicious themselves to attack the network hosts or other agents. There need to be sufficient security mechanisms in place to recognize corrupted agents or hosts, and take corrective or preventive action. Several agent systems focus on the issue of facilitating secure mobile computations in networks. Some of these systems that provide fault tolerance and security have been presented in Chapter 5. Various protocols and ideas have been explored and implemented, and there is still work in progress to achieve secure agent models that are fault tolerant, with minimal overhead and cost outlays.

Additional Applications of Mobile Agents

In all the systems that have been presented thus far, mobile agent systems have been adapted for use in networks, and to aid network functionalities. There are many other applications of mobile agents, which are as necessary to the current day computing world as agents in networks.

Mobile agents have been used in monitoring GRID environments [196], for parallel processing [178] and in agent-based modeling systems [147]. Even a brief overview of each of these topics and the applications of mobile agent systems in them would comprise yet another chapter. There are endless possibilities to the uses and applications of agent technology, and the distributed nature of mobile agent systems makes them extremely conducive to be used in any and every field.

Conclusions

Mobile agent systems are efficient, provide the perfect delegated work force to perform tasks, and significantly improve performance. However, they are yet limited in use for several reasons. The mobile agent paradigm is a relatively new concept, and is therefore not widely accepted very easily. Nevertheless, given the functionality of networks as they are today, and the rapid pace at which the networks are evolving, it will not be very long before there is a widespread demand for such mobile agent systems, and not just a preference for them. Further, mobile agent systems require the necessary frameworks/platforms to execute on. These differ for different agent systems, and it is therefore difficult to support different agent systems on network components. There needs to be a standardization [66, 130] of agent technology and its corresponding agent model before it is widely implemented.

This is by no means a conclusion to the applications of mobile agents. The area is vast enough to comprise another such thesis with ease, and yet have some topics uncovered. The work presented in this thesis has been a modest attempt to put together and present some important mobile agent systems currently in use in networks. With this work, voluminous though it is, the surface of mobile agent applications has barely been scratched, and there are yet unconsidered and unexplored possibilities to the use of this paradigm in every aspect of computer science today.

BIBLIOGRAPHY

[1] Anurag Acharya, M. Ranganathan, and Joel Saltz. Sumatra: A Language for Resource-aware Mobile Programs. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet*, volume 1222, pages 111–130. Springer-Verlag: Heidelberg, Germany, 1997.

[2] K. Amin and Armin Mikler. Towards Resource Efficient and Scalable Routing: An Agent-based Approach. Technical report, Department of Computr Science, University of North Texas, 2001.

[3] Kaizar Amin and A. Mikler. Dynamic Agent Population in Agent-Based Distance Vector Routing. In *ISDA2002: Second International Workshop on Intelligent Systems Design and Applications, Atlanta, USA, August*, 2002.

[4] Kaizar Amin and Armin Mikler. Agent-based Distance Vector Routing: A Resource Efficient and Scalable approach to Routing in Large Communication Networks. Accepted in The Journal of Systems and Software, Elsevier Publications.

[5] Paolo Bellavista Antonio. Protection and Interoperability for Mobile Agents: a Secure and Open Programming Environment, 2000. IEICE Transactions on

Communications, Special Issue on Autonomous Decentralized Systems, Vol. E83-B, No. 5.

[6] Aglets Software Development Kit (ASDK); url: www.http://sourceforge.net/projects/aglets/.

[7] Isaac Asimov. *I, Robot.* Doubleday Publishers, 1963.

[8] ATM Forum; url: http://www.atmforum.com/aboutatm/guide.html.

[9] Mario Baldi and Gian Pietro Picco. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In R. Kemmerer, editor, *Proceedings of the 20th International Conference on Software Engineering*, pages 146–155. IEEE CS Press, 1998.

[10] Benjamin Baran and Ruben Sosa. AntNet Routing Algorithm for Data Networks based on Mobile Agents, 2001. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No. 12 (2001), pp 75-84. ISSN: 1137-3601.

[11] P. Bellavista, A. Corradi, and C. Stefanelli. An Open Secure Mobile Agent Framework for Systems Management, 1999.

[12] R. Bellman. On a Routing Problem, 1958. Quart. Appl. Mat/. 16, 87-90.

[13] D.P. Bertsekas and R. G. Gallager. Data Networks, 1957. Englewood Cliffs, NJ: Prentice-Hall, 1957.

[14] Lubomir Bic, Munehiro Fukuda, and Michael B. Dillencourt. Distributed Computing Using Autonomous Objects. *IEEE Computer*, 29(8):55–61, 1996.

[15] A. Bieszczad, T. White, and B. Pagurek. Mobile Agents for Network Management. *IEEE Communications Surveys*, 1998.

[16] Andrzej Bieszczad, Bernard Pagurek, Tony White, Cheryl Schramm, and Gatot Susilo. Programmable Networks and Their Management Using Mobile Code, 1998. submitted to IEEE Networks.

[17] A. D. Birrell and B. J. Nelson. Implementing Remote Procedure Calls, 1983. Tech. Rep. CSL-83-7, Xerox Palo Alto Research Center, October.

[18] S. Blake, D. Black, and et al. An Architecture for Differentiated Services, 1998. IETF RFC-2475.

[19] E. L. Bodanese and L. Cuthbert. A Multi-Agent Scheme for Resource Allocation in Mobile Networks, 2000. ITC Specialist Seminar on Mobile Systems and Mobility, pp 349-360, Lillehammer, Norway, 22-24 March 2000.

[20] E. L. Bodanese and L. G. Cuthbert. A Multi-Agent Based Scheme for Channel Allocation in Mobile Networks, 2000. World Multiconference on Systemics, Cybernetics and Informatics, Orlando, USA 23-26/07/2000 Volume 7, Computer Science and Engineering Part 1.

[21] E. L. Bodanese and Laurie Cuthbert. A Multi-Agent Channel Allocation Scheme for Cellular Mobile Networks, 2000. 4th International Conference on Multi-Agent Systems (ICMAS 2000), pp. 63-70, Boston, Massachusetts, USA, 7-12 July 2000.

[22] Eliane L. Bodanese and L. Cuthbert. Intelligent Agents for Resource Allocation in Mobile Networks, 2000. XVII World Telecommunications Congress, Birmingham, UK, 7-12 May 2000.

[23] C. Bohoris, A. Liotta, and G. Pavlou. Evaluation of Constrained Mobility for Programmability in Network Management.

[24] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in Telecommunications Networks with “Smart” Ant-Like Agents. In *Intelligent Agents for Telecommunications Applications ’98 (IATA’98)*, 1998.

[25] M. Breugst and T. Magedanz. Mobile Agents - Enabling Technology for Active Intelligent Network Implementation, 1998. Network Magazine, Vol. 12, No. 3, Special Issue on Active and Programmable Networks.

[26] R. A. Brooks. Intelligence Without Representation. In *Artificial Intelligence Journal*, pages 139–159, 1991.

[27] Rodney A. Brooks. Intelligence Without Reason. In John Myopoulos and Ray Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[28] Jose Carlos Brustoloni. Autonomous Agents: Characterization and Requirements. Technical report, Technical Report CMU-CS-91-204, School of Computer Science, Carnegie Mellon University, November, 1991.

[29] P. Buckle, M. Dinis, and L. Cuthbert. Distributed Intelligent Control and Management for 3G Networks, 2000. published at SCI2000, Orlando, Florida, July 2000.

[30] S. Cammarata, D. McArthur, and R. Steeb. Strategies of Cooperation in Distributed Problem Solving. In *In Proceedings of the International Joint Conference On Artificial Intelligence, Karlsruhe*, pages 767–770, 1983.

[31] J. Cao, D. J. Kerbyson, and G. R. Nudd. Use of Agent-based Service Discovery for Resource Management in Metacomputing Environment. In *7th Int. Euro-Par Conf., Manchester, UK, LNCS 2150, Springer Verlag, 882-886*, 2001.

[32] Karel Capek. R.U.R.: Rossum's Universal Robots, 1920. A Czech play.

[33] Luca Cardelli. Obliq A Language with Distributed Scope. Technical Report 122, Digital, Systems Research Center, 1994.

[34] G. Di Caro and M. Dorigo. AntNet: A Mobile Agents Approach to Adaptive Routing, 1997. G. Di Caro and M. Dorigo, AntNet: a mobile agents approach to adaptive routing, Tech. Rep. IRIDIA/97-12, Universit Libre de Bruxelles, Belgium.

[35] G. Di Caro and M. Dorigo. Mobile Agents for Adaptive Routing. In *31st Hawaii International Conference on System Science*, Big Island of Hawaii, 1998.

[36] G. Di Caro and Marco Dorigo. An adaptive multi-agent routing algorithm inspired by ants behavior, 1998.

[37] Gianni Di Caro and Marco Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[38] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing Distributed Applications with a Mobile Code Paradigm. In *Proceedings of the 19th International Conference on Software Engineering*, Boston, MA, USA, 1997.

[39] A. Castillo, M. Kawaguchi, N. Paciorek, and D. Wong. Concordia as enabling technology for cooperative information gathering. In *Proc. of Japanese Society for Articial Intelligence Conference, June*, 1998.

[40] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, London, UK, 1996. Practical Application Company.

[41] Anthony Chavez, Alexandros Moukas, and Pattie Maes. Challenger: A Multi-agent System for Distributed Resource Allocation. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 323–331, New York, 5–8, 1997. ACM Press.

[42] Morsy M. Cheikhrouhou, Pierre Conti, and Jacques Labetoulle. Intelligent Agents in Network Management, a State-of-the-Art. *Networking and Information Systems*, 1(1):9–38, 1998.

[43] Larry T. Chen. AgentOS: The Agent-based Distributed Operating System for Mobile Networks, 1998. ACM Crossroads Student Magazine, Networks and Distributed Systems, Winter.

[44] Cherkassky, Goldberg, and Radzik. Shortest Paths Algorithms: Theory and Experimental Evaluation. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1994.

[45] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile Agents: Are They a Good Idea? Technical Report RC 19887 (December 21, 1994 - Declassified March 16, 1995), IBM, Yorktown Heights, New York, 1994.

[46] Vivek Chokhani and Armin Mikler. Agent Based Wave Computation Towards Controlling the Resource Demand, 2001. Proceedings of the International Workshop IICS (Innovative Internet Computing Systems) 2001, Ilmenau, Germany, June 2001.

[47] S. H. Clearwater, R. Costanza, M. Dixon, and B. Schroeder. Saving Energy Using Market-Based Control, 1996. In Clearwater S. H., editor. Market-Based Control: A Paradigm for Distributed Resource Allocation. Singapore: World Scientific.

[48] Network Knowledge - CMIP; url: http://www.net-know.co.uk/netman/cmip.html.

[49] M. Coen. SodaBot: A Software Agent Construction System, 1995.

[50] Martin Collier. Mobile Agents and Active Networks: Complementary or Competing Technologies? Technical report, Broadband Switching and Systems Laboratory, 1998.

[51] Compaq Management Agents, url: http://www.compaq.com/products/servers/description.html.

[52] CORBA FAQ Page. http://www.omg.org/gettingstarted/corbafaq.htm.

[53] Daniel Corkill and Victor Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *In Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, 1983.

[54] M. Crosbie and G. Spafford. Defending a Computer System using Autonomous Agents. In *8th National Information Systems Security Conference*, 1995.

[55] Gianpaolo Cugola, Carlo Ghezzi, Gian Pietro Picco, and Giovanni Vigna. A Characterization of Mobility and State Distribution in Mobile Code Languages. In *2nd ECOOP Workshop on Mobile Object Systems*, pages 10–19, Linz, Austria, 1996.

[56] D. E. Denning. An Intrusion-Detection Model, 1987. IEEE Trans. Engineering, Vol. SE-13. No. 2, pp.222-232.

[57] E.W. Dijkstra. A note on two problems in connection with graphs, 1959. Numerische Mathematik, Vol. 1, 269-271.

[58] A. Clematis G. Dodero and V. Gianuzzi. A Resource Management Tool for Heterogeneous Networks, 1999. 7th Euromicro Workshop on Parallel and Distributed Processing, Madeira, Portugal, February 3-5 1999, 367-373.

[59] Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A Scalable Comparison-Shopping Agent for the World-Wide Web. In W. Lewis Johnson

and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 39–48, Marina del Rey, CA, USA, 1997. ACM Press.

[60] Cameron Ross Dunne. Using Mobile Agents for Network Resource Discovery in Peer-to-Peer Networks, 2001.

[61] E. H. Durfee. *Coordination of Distributed Problem Solvers.* Kluwer Academic Publishers, Boston, MA, 1988.

[62] E. H. Durfee and V. R. Lesser. Negotiating task decomposition and allocation using partial global planning, 1989. Distributed Artificial Intelligence 2.

[63] George Eleftheriou and Alex Galis. Mobile Intelligent Agents for Network Management Systems, 2000. In the 2000 London Communications Symposium.

[64] Oren Etzioni and Daniel P. Weld. A Softbot-Based Interface to the Internet. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 77–81. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[65] G. Feher, K. Nemeth, and et al. Boomerang - A Simple Protocol for Resource Reservation in IP Networks, 1999. In IEEE Workshop on QoS Support for Real-Time Internet Applications, Vancouver, Canada, June 1999.

[66] Foundation for Intelligent Physical Agents; url: www.fipa.org.

[67] Leonard N. Foner. What's An Agent, Anyway? A Sociological Case Study. Technical report, MIT Media Lab, 1993.

[68] L. R. Ford and D. R. Fulkerson. Flows in Networks, 1962. Princeton Univ. Press, Princeton, NJ.

[69] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A Sense of Self for Unix Processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[70] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service*, 1999.

[71] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)*, volume 1193, Berlin, Germany, 1996. Springer-Verlag.

[72] D. Gavalas, D. Greenwood, M. Ghanbari, and M. O'Mahony. Advanced network monitoring applications based on mobile/intelligent agent technology, 2000.

[73] Guidelines for the Definition of Managed Objects, 1992. ISO10165-4 Open Systems Interconnection, Structure of Management Information, ISO/ IEC 10165-4, ITU-T Recommendation X.721.

[74] Michael R. Genesereth and Steven P. Ketchpel. Software Agents. *Communications of the ACM*, 37(7), 1997.

[75] M.P. Georgeff. Communication and interaction in Multi-Agent Planning. In *In Proc. of the third National Conference on Artificial Intelligence (AAAI-83)*, 1983.

[76] D. Goldberg. *Genetic Algorithms in Search.* AddisonWesley, 1989. Optimization and Machine Learning.

[77] G. Goldszmidt. Network Management Views using Delegated Agents, 1996.

[78] G. Goldszmidt, S. Yemini, and Y. Yemini. Network Management by Delegation - the MAD Approach, 1991.

[79] Sergio Gonzalez-Valenzuela, Victor C. M. Leung, and Son T. Vuong. Multipoint-to-Point Routing with QoS Guarantees Using Mobile Agents. In *MATA*, pages 63–72, 2001.

[80] R. Gopalakrishna. A Framework for Distributed Intrusion Detection using Interest Driven Cooperating Agents, 2001.

[81] P. P. Grasse'. La reconstruction du nid et lescoordinations inter-individuelles chez Bellicoitermes, 1959. Solve Foraging Problems for Ant Colonies in Hughes R.N.(ed.) NATO ASI Series, Vol.G20, Behavioural Mechanisms of Food Selection,Springer Verlag,Berlin.

[82] Grasshopper Mobile Agent Platform. Center for Communications System Research, Electronic Engineering, Surrey, UK, url:http://www.ee.surrey.ac.uk/CCSR/ACTS/Miami/grasshopper.html.

[83] Robert S. Gray. Agent Tcl: A transportable agent system. In *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95)*, Baltimore, Maryland, December 1995.

[84] Shaw Green, Leon Hurst, Brenda Nangle, Dr. Pdraig Cunningham, Fergal Somers, and Dr. Richard Evans. Software Agents: A review, 1997. A review published by the Intelligent Agents Group, a collaboration of Broadcom Ireland (Broadcom ireann Research Ltd) and the Trinity College, Dublin.

[85] David Griffin, George Pavlou, and Panos Georgatsos. Providing Customisable Network Management Services Through Mobile Agents. In *IS&N*, pages 209–226, 2000.

[86] S. Guerin. Optimisation multi-agents en environnement dynamique: application au routage dans les rseaux de tlcommunications, 1997. DEA report.

[87] B. Hayes-Roth. A blackboard architecture for control, 1985. Artificial Intelligence, 26(3):251–321.

[88] Alex L. G. Hayzelden and John Bigham. Software Agents in Communications Network Management: An Overview. Technical report, Queen Mary and Westfield College, University of London, 1998.

[89] C. Hedrick. Routing Information Protocol, 1988. Internet Request for Comments 1058, June 1988.

[90] G. Helmer, J. Wong, V. Honavar, and L. Miller. Intelligent agents for intrusion detection, 1998.

[91] C. Hewitt, P. Bishop, and R. Steiger. A Universal Modular Actor Formalism for Artificial Intelligence. In *Proc IJCAI '73, Stanford, California, pp235– 245*, 1973.

[92] Carl Hewitt. Viewing Control Structures as Patterns of Passing Messages, 1977. Journal of Artificial Intelligence, Volume 8 Number 3.

[93] Fritz Hohl. A Framework to Protect Mobile Agents by using Reference States. In *International Conference on Distributed Computing Systems*, pages 410–417, 2000.

[94] J. Huang, N. R. Jennings, and J. Fox. An Agent-based Approach to Health Care Management. *Applied Artificial Intelligence: An International Journal*, 9(4):401–420, 1995.

[95] Michael N. Huhns, Uttam Mukhopadhyay, Larry M. Stephens, and Ronald D. Bonnell. *DAI for document retrieval: The MINDS project*, volume In Michael N. Huhns, editor, Distributed Artificial Intelligence, pages 249–283. Pitman/Morgan Kaufmann, London, 1987.

[96] Leila Ismail and Daniel Hagimont. A performance evaluation of the mobile agent paradigm. In *Conference on Object-Oriented*, pages 306–313, 1999.

[97] Prasanna Venkatesan Iyengar. Dynamic Resource Management for RSVP-Controlled Unicast Networks. Master's thesis, University of North Texas, Denton, Texas, December 2001.

[98] Brendan Jennings et al. FIPA-compliant agents for real-time control of Intelligent Network traffic. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(19):2017–2036, 1999.

[99] N. R. Jennings, J. M. Corera, L. Laresgoiti, E. H. Mamdani, F. Perriollat, P. Skarek, and L. Z. Varga. Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control. *IEEE Expert*, 1995.

[100] N. R. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[101] J.Harju, T.Karttunen, and O.Martikainen. *Intelligent Networks.* Chapmen & Hall, 1994.

[102] D. Johansen, R. van Renesse, and F. B. Schneider. An Introduction to the TACOMA Distributed System—Version 1.0. Technical Report 95-23, Institute of Mathematical and Physical Sciences, University of Tromsø, Tromsø, Norway, June 1995.

[103] Michael B. Jones. Interposition Agents: Transparently Interposing User Code at the System Interface. In *Symposium on Operating Systems Principles*, pages 80–93, 1993.

[104] Kyungkoo Jun, Ladislau Bni, Krzysztof Palacz, and Dan C. Marinescu. Agent-Based Resource Discovery. In *Heterogeneous Computing Workshop*, pages 43–52, 2000.

[105] Burton S. Kaliski, Jr. Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services. Technical Report 1424, RSA Laboratories, 1993.

[106] T. Karygiannis. Network Security Testing Using Mobile Agents. National Institute of Standards and Technology, Computer Resource Security Center, Publication in Practical Application of Intelligent Agents and Multi-Agents, March 1998.

[107] Alan Kay. Computer Software, 1984. Scientific American 251.

[108] E. Kovacs, K. Rohrle, and M. Reich. Integrating Mobile Agents into the Mobile Middleware. In *Proc. Mobile Agents International Workshop (MA98), Springer-Verlag*, pages 124–135, 1998.

[109] B. Krulwich. The BargainFinder agent: comparison price shopping on the Internet, 1996. in Bots and Other Internet Beasties. Sams Publishing.

[110] Dimitar Lakov. Routing via Dedicated Intelligent Agents. In *Proceedings of European Symposium on Intelligent Techniques, ESIT 2000, Germany*, 2000. Soft Computing in Telecommunication Networks.

[111] D.B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. Communciations of the ACM, 45(3):88–89, March 1999.

[112] Guy W. Lecky-Thompson. Mobile Agents and Framework Models, 1997. url: http://www.geocities.com/CapeCanaveral/Lab/1888/fw.htm.

[113] Antonio Liotta, Graham Knight, and George Pavlou. On the Performance and Scalability of Decentralized Monitoring Using Mobile Agents. In *DSOM*, pages 3–18, 1999.

[114] Antonio Liotta, George Pavlou, and Graham Knight. A Self-adaptable Agent System for Efficient Information Gathering. In *MATA*, pages 139–152, 2001.

[115] P. Maes. Agents that reduce work and information overload, 1994. Communications of the ACM., 37 (7).

[116] Pattie Maes. Modeling Adaptive Autonomous Agents. *Artificial Life, I*, (1&2)(9), 1994.

[117] Maes P. Artificial Life Meets Entertainment: Lifelike Autonomous Agents, 1995. published in Communication of the Association for Computing Machinery, V38, No. 11, November.

[118] T. Magedanz, K. Rothermel, and S. Krause. Intelligent Agents: An Emerging Technology for Next Generation Telecommunications? In *INFOCOM'96*, San Francisco, CA, USA, 24–28 1996.

[119] G. S. Malkin and M. E. Steenstrup. Distance Vector Routing, 1995. In M. E. Steenstrup, editor, Routing in Communications Networks, pages 83-98, Prentice Hall, 1995.

[120] T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard. Enterprise: A Market-like Task Scheduler for Distributed Computing Environments, 1988. In

B. A. Huberman, editor, The Ecology of Computation, pages 177–205. North-Holland, Amsterdam, 1988.

[121] A. D. Marshall. Department of Computer Science, Cardiff University, Wales, Remote Procedure Calls; url: http://www.cs.cf.ac.uk/Dave/C/node33.html.

[122] Jean-Philippe Martin-Flatin and et al. Annotated Typology of Distributed Network Management Paradigms. Technical report, Swiss Federal Institute of Technology, Communication Systems Division, Lausanne, Switzerland, 1997.

[123] F. Matthes, G. Schroder, and J. W. Schmidt. *Tycoon: A Scalable and Interoperable Persistent System Environment.* M.P.Atkinson (ed.): Fully Integrated Data Environments. Springer-Verlag, 1999.

[124] Management by Delegation, German Goldzsmidt; url: http://www.cs.columbia.edu/g̃erman/mbd.html.

[125] J. McCarthy. Ascribing Mental Qualities to Machines, 1979. reprinted in Formalizing Common Sense in 1990.

[126] John McCarthy. Programs with Common Sense. In *Teddington Conference on the Mechanization of Thought Processes*, 1958.

[127] H. De Meer, A. Corte, A. Puliafito, and O. Tomarchio. Programmable Agents for Flexible QoS Management in IP Networks, 2000.

[128] MIAMI Project Web Site, http://www.fokus.gmd.de/research/cc/ima/miami.

[129] Management Information Base for Network Management of TCP/IP-based internets: MIB-II. RFC 1213, Network Working Group, March 1991.

[130] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF, The OMG Mobile Agent System Interoperability Facility. In *In Kurt Rothermel and F. Hohl, editors, Mobile Agents, Proceedings of the Second International Workshop, MA'98, LNCS 1477, Springer Verlag*, pages 14–15, 1998.

[131] N. Minar, K. H. Kramer, and P. Maes. Cooperating Mobile Agents for Dynamic Network Routing, 1999. Software Agents for Future Communications Systems, Springer-Verlag, 1999.

[132] Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes. Cooperating Mobile Agents for Mapping Networks. In *Proceedings of the First Hungarian National Conference on Agent Based Computation*, 1999.

[133] Yaron Minsky, Robbert van Renesse, Fred B. Schneider, and Scott D. Stoller. Cryptographic Support for Fault-Tolerant Distributed Computing. In *Proceedings of the Seventh ACM SIGOPS European Workshop*, pages 109–114, Connemara, Ireland, 1996.

[134] Vayias Soldatos Mitrou. Open and Flexible Control and Resource Management for ATM Networks using Intelligent Agents. In *in Proc. of 6th International Conference on Intelligence in Networks (ICIN2000)*, pages 205–209, 2000.

[135] Home of the Mole - Mobile Agents' List, url: http://mole.informatik.uni-stuttgart.de/mal/preview/preview.html.

[136] E. F. Moore. Shortest Path Through a Maze. In *in Kluwer Proceedings of the International Symposium on Switching Circuits*, 1959.

[137] Patricia Morreale. Agents on the Move, 1998. in IEEE Spectrum, April.

[138] Multi Protocol Label Switching; url: http://www.mplsforum.org/.

[139] B. Mukherjee, L.T. Heberlein, and K.N. Levitt. Network intrusion detection. IEEE Network, 8(3):26–41, May-June 1994.

[140] Klara Nahrstedt and Jonathan M. Smith. The QOS Broker. *IEEE MultiMedia*, 2(1):53–67, Spring 1995.

[141] Victor P. Nelson and Bill D. Carroll. *Tutorial: Fault-Tolerant Computing.* The Computer Society of the IEEE, 1987.

[142] B. C. Neuman and T. Tsó. Kerberos: An Authentication Service for Computer Networks. IEEE Communications Magazine, 32(9):33-38, September 1994.

[143] Jan Nicklisch, Jürgen Quittek, Andreas Kind, and Shinya Arao. INCA: An Agent-Based Network Control Architecture. In S. Albayrak and F. J. Garijo, editors, *Intelligent Agents for Telecommunication Applications — Proceedings of the Second International Workshop on Intelligent Agents for Telecommunication (IATA'98)*, volume 1437. Springer-Verlag: Heidelberg, Germany, 1998.

[144] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. PACE — A toolset for the performance prediction of parallel and distributed systems. *The International Journal of High Performance Computing Applications*, 14(3):228–251, Fall 2000.

[145] H. S. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 11(2):205–244, 1995.

[146] Hyacinth Nwana and Divine Ndumu. An Introduction to Agent Technology, 1996. Intelligent Systems Research, Applied Research and Technology, BT Labs,

BT Technology Journal Vol 14 No 4 October.

[147] D. O'Sullivan, H. Mordechay, M. Thurstain-Goodwin, and T. Schelhorn. STREETS: An Agent Based Pedestrian Model. presented in the 6ht International Conference on Computers in Urban Planning & Urban Management (CUPUM), Venice, Spetember 1999.

[148] Marcelo Rubinstein Otto. Scalability of a Network Management Application Based on Mobile Agents. Technical report, GTA (Grupo de Teleinformatica e Automacao), 2002.

[149] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison Wesley, 1994.

[150] Holger Pals, Stefan Petri, and Claus Grewe. FANTOMAS: Fault Tolerance for Mobile Agents in Clusters, 2000.

[151] H. Van Dyke Parunak. *Manufacturing Experience with the Contract Net.* In M. Huhn, editor of Distributed Artificial Intelligence. Pitman Publishing, 1987.

[152] A. Puliafito and O. Tomarchio. Using Mobile Agents to implement flexible Network Management strategies, 2000.

[153] A. Puliafito, O. Tomarchio, and L. Vita. MAP: Design and Implementation of a Mobile Agents Platform. Technical Report TR-CT-9712, Journal of Systems Architecture: the EUROMICRO Journal, 1997.

[154] Antonio Puliafito and Orazio Tomarchio. Advanced Network Management Functionalities through the Use of Mobile Software Agents. In *IATA*, pages 33–45, 1999.

[155] B. Randell. System Structure for Software Fault Tolerance, 1975. IEEE Transactions on Software Engineering, Vol. SE-1.

[156] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[157] David Reilly. Mobile Agents - Process Migration and its Implications, 1998. project for post-graduate degree, Bond University, Queensland, url: http://www.davidreilly.com/topics/software_agents/mobile_agents/.

[158] H. Reiser and G. Vogt. Security Requirements for Management Systems using Mobile Agents. In *Proceedings of the Fifth IEEE Symposium on Computers & Communications*, pages 160–165, July 2000.

[159] Angelica Reyes, Ernesto Sanchez, and Antonio Barba. Routing Management Application Based on Mobile Agents on the Internet2. Innovative Internet Applications, Sixth EUNICE Open European Summer School, September 13 - September 15, 2000, University of Twente, Enschede, the Netherlands.

[160] Cisco Systems - Remote Monitoring, url: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/rmon.htm.

[161] Nicolas Rouhana and Eric Horlait. Dynamic Congestion Avoidance Using Multi-agent Systems. In *Mobile Agents for Telecommunication Applications, MATA 2001: 1-10*, 2001.

[162] Marcelo G. Rubinstein, Otto Carlos M. B. Duarte, and Guy Pujolle. Evaluating the Performance of a Network Management Application Based on Mobile Agents, 2002. to be published in Lectures Notes in Computer Science, Networking, Pisa, Italy.

[163] Marcelo Gonalves Rubinstein and Otto Carlos Muniz Bandeira Duarte. Evaluating Tradeoffs of Mobile Agents in Network Management. *Networking and Information Systems Journal*, 2(2):237–252, 1999. HERMES Science Publications.

[164] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Series in Artificial Intelligence, 1995.

[165] R. A. Sahner, K. S. Trivedi, and A. Puliafito. Performance and Reliability Analysis of Computer Systems, 1996. Kluwer Academic Publishers, Boston.

[166] Tomas Sander and Christian F. Tschudin. Protecting Mobile Agents Against Malicious Hosts. *Lecture Notes in Computer Science*, 1419:44–??, 1998.

[167] P. S. Sapaty. The WAVE paradigm. Technical report, Technical Report 17/92, Dept. of Informatics, Univ. of Karlsruhe, Germany, July, 1992.

[168] O. Schelen, A. Nilsson, J. Norrgard, and S. Pink. Performance of QoS Agents for Provisioning Network Resources. In *in Proceedings of the IFIP 7th international IWQoS conference*, London, UK, June 1999.

[169] O. Schelen and S. Pink. Resource Reservation Agents in the Internet. In *Proceedings of the 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98), Cambridge, United Kingdom, July 1998.*, 1998.

[170] O. Schelen and Stephen Pink. Sharing Resources through Advance Reservation Agents. In *Proceedings of IFIP Fifth International Workshop on Quality of Service (IWQoS'97), New York*, May 1997.

[171] Olov Schelen and Stephen Pink. An Agent-based Architecture for Advance Reservations. In *Proceedings of IEEE 22nd Annual Conference on Computer Networks (LCN'97), Minneapolis*, November 1997.

[172] Ruud Schoonderwoerd, Owen E. Holland, Janet L. Bruten, and Leon J. M. Rothkrantz. Ant-Based Load Balancing in Telecommunications Networks. *Adaptive Behavior*, (2):169–207, 1996.

[173] R. Schrooten and W. Van de Velde. Software agent foundation for dynamic interactive electronic catalogs, 1997. Applied Artificial Intelligence, 11:459–481.

[174] U. M. Schwuttke and A. G. Quan. Enhancing Performance of Cooperating Agents in Realtime Diagnostic Systems. In *In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 332–337, 1993. Chambery, France.

[175] Oliver G. Selfridge and Ulric Neisser. Pattern Recognition by Machines, 1960. Scientific American, 203:60-68.

[176] S. Shenker, C. Partridge, and R. Guerin. Specification of Guaranteed Quality of Service, 1997. RFC 2212.

[177] Luis Moura Silva, Paulo Simões, Guilherme Soares, Paulo Martins, Victor Batista, Carlos Renato, Leonor Almeida, and Norbert Stohr. JAMES: A Platform of Mobile Agents for the Management of Telecommunication Networks. In S. Albayrak, editor, *Intelligent Agents for Telecommunication Applications — Proceedings of the Third International Workshop on Intelligent Agents for Telecommunication (IATA'99)*, volume 1699, pages 76–95. Springer-Verlag: Heidelberg, Germany, 1999.

[178] Lus Moura Silva, Victor Batista, Paulo Martins, and Guilherme Soares. Using Mobile Agents for Parallel Processing. International Symposium on Distributed Objects and Applications, September 05 - 07, 1999, Edinburgh, United Kingdom.

[179] Lus Moura Silva, Guilherme Soares, Paulo Martins, Victor Batista, and Lus Santos. The Performance of Mobile Agent Platforms, 1999. First International Symposium on Agent Systems and Applications Third International Symposium on Mobile Agents, Palm Springs, California, October.

[180] P. Simes, R. Reis, L. Silva, and F. Boavida. Enabling Mobile Agent Technology for Legacy Network Management Frameworks, 1999.

[181] Paulo Simes, Luis Moura Silva, and Fernando Boavida Fernandes. Integrating SNMP into a Mobile Agent Infrastructure. In *Proceedings of DSOM'99 - Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Springer-Verlag LNCS 1700, Zurich, Switzerland*, 1999.

[182] David Canfield Smith, Allen Cypher, and Jim Spohrer. KidSim: programming agents without a programming language. *Communications of the ACM*, 37(7):54–67, 1994.

[183] SNMP World : http://silver.he.net/ rrg/snmpworld.htm.

[184] SNMP RFC 1157, J. Case, M. Fedor, M. Schoffstall, J. Davin, 1990.

[185] Salvatore J. Stolfo, Andreas L. Prodromidis, Shelley Tselepis, Wenke Lee, Dave W. Fan, and Philip K. Chan. JAM: Java Agents for Meta-Learning over Distributed Databases. In *Knowledge Discovery and Data Mining*, pages 74–81, 1997.

[186] Markus Straer and Kurt Rothermel. Reliability Concepts For Mobile Agents, 1998. International Journal of Cooperative Information Systems (IJCIS), Volume 7, Number 4, World Scientific Publishing Company.

[187] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.

[188] Robert Szabo, Tamas Henk, Vlora Rexhepi, and Georgios Karagiannis. Resource Management in Differentiated Services (RMD) IP Networks, 2001. Papers published as part of The Internet NG Project.

[189] K. Takahashi, Y. Nishibe, I. Morihara, and F. Hattori. Intelligent Pages: Collecting Shop and Service Information with Software Agents, 1997. Applied Artificial Intelligence Intl. J., Vol. 11, No. 6, Taylor and Francis.

[190] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.

[191] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[192] Warsaw March This. J'ozef Winkowski A MULTI-AGENT SYSTEM FOR DISTRIBUTED JOB AND RESOURCE MANAGEMENT, 1999. Institute of Computer Science at the Polish Academy of Sciences.

[193] B. Thomsen, L. Leth, S. Prasad, T.-M. Kuo, A. Kramer, F. Knabe, and A. Giacalone. Facile Antigua Release Programming Guide. Technical report, European Computer-Industry Research Centre, ECRC-93-20, 1993.

[194] K. R. Thorisson. *Communicative Humanoids: A Computational Model of Psychosocial Dialogue Skills*. PhD thesis, Program in Media Arts & Sciences, School of Architecture & Planning, MIT, 1996.

[195] Manish Tiwari. Active Networks, 1998-1999. Student Report on Networking Issues, IIT, Department of Computer Science and Engineering, Kanpur.

[196] Orazio Tomarchio, Lorenzo Vita, and et al. Active Monitoring In Grid Environments Using Mobile Agent Technology. In 2nd Workshop on Active Middleware Services (AMS'00) in HPDC-9, Pittsburgh (Pennsylvania (USA)), August 2000.

[197] C. Tschudin. An Introduction to the M0 Messenger Language, 1994. University of Geneva, Switzerland.

[198] M. Tsvetovatyy, M. Gini, B. Mobasher, and Z. Wieckowski. MAGMA: An agent-based virtual market for electronic commerce, 1997.

[199] Mark van Assem. Limitations of Agent Classifications: Subjective Objectivity, 1998. Project for Artificial Intelligence, Vrije Universiteit Amsterdam, Department of Artificial Intelligence, Netherlands.

[200] L. Z. Varga, N. R. Jennings, and D. Cockburn. Integrating Intelligent Systems into a Cooperating Community for Electricity Distribution Management. *Int Journal of Expert Systems with Applications*, 7(4):563–579, 1994.

[201] Evangelos Vayias, John Soldatos, and Nikolas Mitrou. An Approach to Network Control and Resource Management Based on Intelligent Agents. In *IS&N*, pages 191–207, 2000.

[202] B. Venners. Under the Hood: The architecture of Aglets. JavaWorld: IDG's magazine for the Java community, 2(4), April 1997.

[203] H. Vogler, T. Kunkelmann, and M.-L. Moschgath. An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions, 1997. Proc. 1997 Int'l Conference on Parallel and Distributed Systems (ICPADS'97), Seoul, Korea, December, IEEE Computer Society, ISBN 0-8186-8227-2.

[204] Voyager, url: http://www.objectspace.com/products/voyager/.

[205] H. Wang and C. Wang. Intelligent Agents in the Nuclear Industry, 1997. IEEE Computer, 30(11):28-34.

[206] Jim White. Mobile Agents White Paper, 1996. Telescript Mobile Agent System, General Magic.

[207] T. White. Routing with Swarm Intelligence. Technical Report SCE-97-15, Systems and Computer Engineering Department, Carleton University, 1997.

[208] T. White, A. Bieszczad, and B. Pagurek. Distributed Fault Location in Networks Using Mobile Agents. In S. Albayrak and F. J. Garijo, editors, *Intelligent Agents for Telecommunication Applications — Proceedings of the Second International Workshop on Intelligent Agents for Telecommunication (IATA'98)*, volume 1437. Springer-Verlag: Heidelberg, Germany, 1998.

[209] T. White, B. Pagurek, and F. Oppacher. Connection Management using Adaptive Agents. In *Proceedings of 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, 1998.

[210] Qiao Xin, Yang Yu, Yu Xu, and Zhanhai Qin. Fault Tolerance Issues in Mobile Agents. Project Report, University of California, San Diego, June 2000.

[211] Bennet S. Yee. A Sanctuary for Mobile Agents. In *Secure Internet Programming*, pages 261–273, 1999.