Journal of Computer Science 2 (8): 646-659, 2006
ISSN 1549-3636

A Network Management Framework Using Mobile Agents

Jonathan Lefebvre, Steven Chamberland and Samuel Pierre
Department of Computer Engineering, École Polytechnique de Montréal, C.P. 6079
Succ. Centre-Ville, Montréal (Québec), Canada H3C 3A7

Abstract: Network management of heterogeneous networks is still hard to achieve automatically and efficiently. In this study, we present a framework that has the ability to perform network management tasks on heterogeneous networks using mobile agents. This framework handles the inability of many network devices to run mobile agents. While the main focus of the project is the framework, we present an example of mobile agents that are able to locate a fixed set of network failures and detect the possible causes accurately. Experimental results show that some network management tasks can be more easily executed by mobile agents. In particular, search and diagnostic mobile agents are able to find more precisely a cause of a network failure by finding alternate paths to gather more data about the failure.

Key words: Network management, mobile agents, network diagnosis, fault location

INTRODUCTION

Network management of heterogeneous networks is still hard to achieve automatically and efficiently. Indeed, since one network management system is typically used per technology and per vendor, it is difficult to perform simple network management tasks such as fault location and service provisioning.

Considering the flexibility and scalability limitations of the centralized network management[1,2], in the last decade, a lot of research has been done for using mobile agents efficiently for network management. Mobile agents decentralize management tasks and distribute load over the network. They also provide a fast and flexible way to create solutions for fast-evolving environments and enable the possibility to automate management tasks efficiently. For a survey of the potential uses of the mobile agents in network management, see[3].

Several network management solutions using mobile agents have been proposed in the network management literature. The MAMAS (Mobile Agents for the Management of Applications and Systems) environment, proposed in[4], is a secure and open mobile agent environment for the management of networks, services and systems. An implementation is proposed. However, no performance analysis is done. The objective of the JAMES project[5] is to create an efficient mobile agent platform mainly used for network management. It provides many optimizations in comparison to commercial mobile agent platforms that are mandatory to achieve efficiency and high performance in the domain of network management. The Network Management and Artificial Intelligence Laboratory of Carleton University[6] is doing research on many applications and models to bring mobile agents to network management. They have studied many aspects of the subject such as the need for a uniform management interface, the use of the simple network management protocol (SNMP)[7] and mobile agents[8].

Novel architectures inspired by simple life organisms have been proposed. One such architecture is called ECOMOBILE[9]. It uses mobile agents to execute task objectives, but these agents are not by themselves network tasks. They have a life, compete with each other, exchange and take or leave task objectives at any time. This architecture offers an interesting way to regulate its mobile agent population while achieving network management tasks. ANTNET[10] is another architecture in that field. It was introduced at first to use mobile agents for adaptive routing. However, it has inspired a lot of research[8,11,12]. The common point of this research is the accomplishment of complex objectives using simple mobile agents. On the concept of proximity, a research group[1] has studied efficient ways to place mobile agents on a network combining both mobility and remote monitoring. Monitoring is one part of network management and may be used for fault management as well. Recently in[13], a performance management system based on mobile agents for virtual home environment has been proposed. However, this system is limited to performance management.

New approaches using active nodes and lightweight agents, such as Weaver[14] and Chameleon[15], have been proposed in the literature. These systems are highly-scalable but, in general, they can not be utilized on existing networks since the

Corresponding Author: Samuel Pierre, Department of Computer Engineering, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-Ville, Montréal (Québec), Canada H3C 3A7
Tel: +1 (514) 340-4711, Fax: +1 (514) 340-4658

majority of commercial routers do not permit the execution of user-supplied code. However, workarounds can be used, for instance, by attaching to each router a single-board computer[14] or by using shared proxies.

The first objective of this study is to create a network management framework using mobile agents to investigate the utility of them in real networks. Even though the concept of using mobile agents for network management has been considered before, it is the first time that such framework is implemented and tested using a heterogeneous network (containing, among other equipments, internet protocol (IP) routers, asynchronous transfer mode (ATM) switches and several management stations) in various environments. The proposed framework can also be used to manage elements that cannot receive mobile agents. The second objective of the study is that the framework can be utilized on existing networks.

THE FRAMEWORK

The proposed model suggests using one or two agents per network management task. This allows us to limit inter-agent communications that can be costly if the framework tends to use small and inefficient agents. By inefficient, we mean agents that cannot act autonomously. In[16] and[2], it was shown that using only one mobile agent for one task increases the task response time, but lowers the total traffic on the network. A study presenting a real configuration task[17] provides a performance evaluation of using one mobile agent against using parallel mobile agents for one task. In fact, communication and synchronization between multiple agents slows the whole task to a point where a single agent performs better in both areas. It is therefore impossible to give an optimal choice for every topology, network size and management task. Our choice to use few mobile agents for one task is based on[17] and on our motivation to limit the network load and agent building complexity. For tasks where there are few dependencies between each device, we suggest using multiple instances of the same mobile agent to divide the task.

Network management is done using network tools already available. The advantage of using existing protocols such as SNMP is pointed out by[18]. They are already widely supported and implemented in network devices and limit the effort involved in building a network management framework. Also, mobile agents tend to use these tools more efficiently.

Management table: The management table is the only knowledge of the network that the framework provides. Any other knowledge is taken from the network as needed by mobile agents. This table keeps links between network elements and network management stations. One important utility of this table is to manage elements that cannot receive mobile agents. For our experiments, associations in this table were static, meaning that one management station was bind permanently to one or many elements. The association is based on proximity. Proximity may be determined by a wide selection of factors. In our case, it is simply the number of hops between the station and the element. Static associations do not mean that an element is always managed by the same station. It only tells the mobile agent the preferred management station for a particular element. The framework could use a dynamic update for this table and a way to adapt to network modifications made on topology. One research study[1] offers interesting ideas on how this aspect could be improved.

For optimization purposes, these tables are installed on each management station, freeing the mobile agent's memory to save bandwidth. This also allows local optimizations when it is not clear whether a central element must be managed by one station or another depending on the point of view.

Network Management Interface: Uniform interfaces are key parts of many mobile agent systems[19-22]. However, our framework is not tied to uniform interfaces. This lets us introduce two kinds of mobile agents, general agents and specialized agents. The general agent will mostly use uniform interfaces, managing the network with limited functionality. The specialized agent is able to do a lot more tasks and use specialized features.

Stationary agent: Stationary agents are used to implement network management code that has to be dynamic, but may be totally inefficient to move with mobile agents. By dynamic, we mean that they could keep a state, be modified easily; keep local information in cache for fast and efficient retrieving. This code is moved once and stays permanently on the station.

Stationary agents implement a set of uniform management interfaces. The management table that keeps references between management stations and elements also keeps a set of network management abilities for an element. Such abilities could be an operating system application programming interface (API), a protocol like SNMP or any other way to manage an element. These stationary agents look like interface agents found in[23], but fill a wider range of functionalities and utilities.

Intelligence: Mobile agents need a great load of intelligence to be able to manage networks of heterogeneous devices. Although the framework uses uniform interfaces, it is still difficult to give agents sufficient intelligence to let them manage these networks confidently. Some research tends to use artificial intelligence or collective intelligence[8,10-12]. Our focus was to use an expert system, but the

framework is not limited to a specific form of intelligence. The network tasks implemented using our framework aim to use proven procedural instructions that are best implemented by an expert system.

Security and fault-tolerance: For networks where agents should move on user stations, the framework suggests, but does not yet implement, letting only the approved mobile agent's code to get back from user stations. For confidentiality purposes, mobile agents should give up sensible information from the network before entering a user station. Quotas may also be used to counter flooding.

Since the framework is on top of any network management system, it does not interfere with these systems and is not needed for management. Fault management mobile agents described later are able to tell if the network management system is faulty, but they cannot recover completely from such a failure.

Global view of the framework: Two logical networks are present: the management network and the normal network. The management network is essentially the management stations and the links between them. The normal network is the part assigned to useful applications. Both logical networks may be the same, have some devices and links shared or be completely different networks. In Fig. 1, we see a general view with a network device that can accept a mobile agent (active node) and a device that cannot (passive node). Each passive node must have an associated management station (management node) to be managed. Mobile agents on the network are depicted as a person icon. We also see the composition of a management mobile agent which is mainly its data, its execution state, its intelligence and its abilities. Each management station runs a mobile agent platform and installs basic elements and stationary agents used by mobile agents. Mobile agents can migrate in selected private networks and are not allowed to migrate on public networks unless it is in a strictly controlled manner. This requirement serves the minimal security model explained earlier. Management stations are detailed in Fig. 2.

Each management station of the framework runs a mobile agent platform that can receive and launch mobile agents. A station contains a network management mobile agent bank that stores each mobile agent that may be needed to accomplish a task. These mobile agents, through stationary agents, may access local operating system functions and any ability installed on this station.

IMPLEMENTATION

We first describe all technologies used in the framework and in mobile agents. We have already presented key elements of the framework earlier. Implementation of management tables, uniform interfaces, stationary agents and communication between agents are explained here. Then, we present in detail mobile agents that were used to validate the framework. We have implemented two mobile agents that are able to find a set of network failures in a network. These mobile agents never claim to be able to find all possible errors. The first mobile agent, called Diagnostic, tries to go as far as possible in a network to find a failure cause. The diagnostic capability of this agent could be reproduced by a stationary agent. The second agent, named Search, is used to pinpoint more precisely the cause of a network failure.

Technology used: To implement the framework, we used both Java and C++ programming languages. Java is used for almost all aspects of the framework. The C++ language is used to implement some advanced functions that are accessed using JNI (Java Native Interface). It also demonstrates the framework's ability to use multiple technologies and therefore use virtually any management functionalities in a heterogeneous network. Each mobile agent is implemented only in Java and interacts with Java modules. Mobile agents are built using the Grasshopper platform and API (Application Program Interface)[24]. Grasshopper was a natural choice because of its simplicity, maturity and supported operating systems and implemented standards. To access elements using SNMP, we used AdventNet easy-to-use classes[25] and Java Beans. The management information base of type II (MIB-II) has been used.

Implementation of the management tables: Management tables are placed on each network management station. The current implementation does not require that a management table be installed on each station, but strongly suggests it. Mobile agents find and access these tables by creating a proxy to the table. We give more information on this type of communication later. The management table implementation uses a hash map to link the management station to network devices. The key of the map is a unique identifier (Table 3), therefore allowing a management station to manage any devices and restrict a device to having only one management station assigned per table.

Communication, interfaces and stationary agents: The main communication medium between agents in our framework is done using remote procedure calls (RPCs) instead of KQML (Knowledge Query Manipulation Language) or ACL (Agent Communication Language), to have a better control of the communication mechanisms. We use the Grasshopper proxy communication mechanism to enable communication between agents. The framework favors local communications between mobile agents

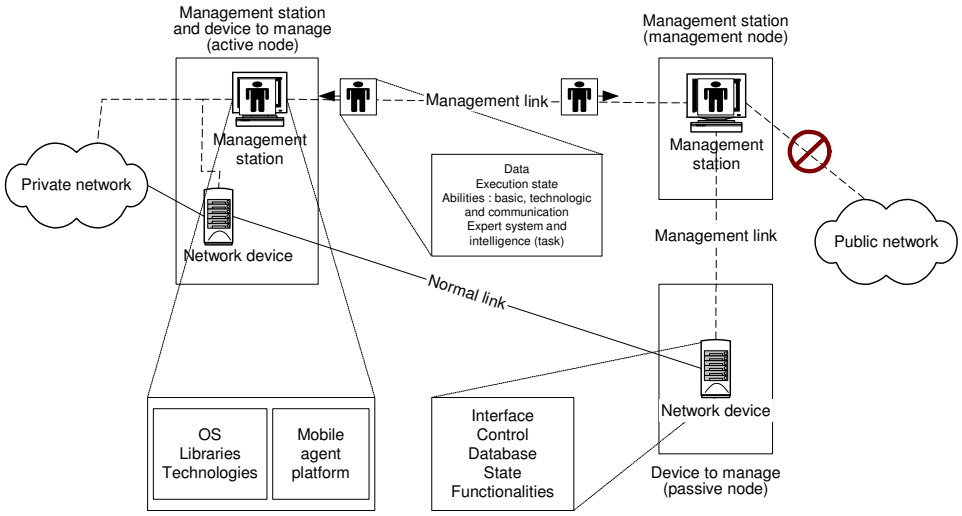

Fig. 1: Global view of the framework

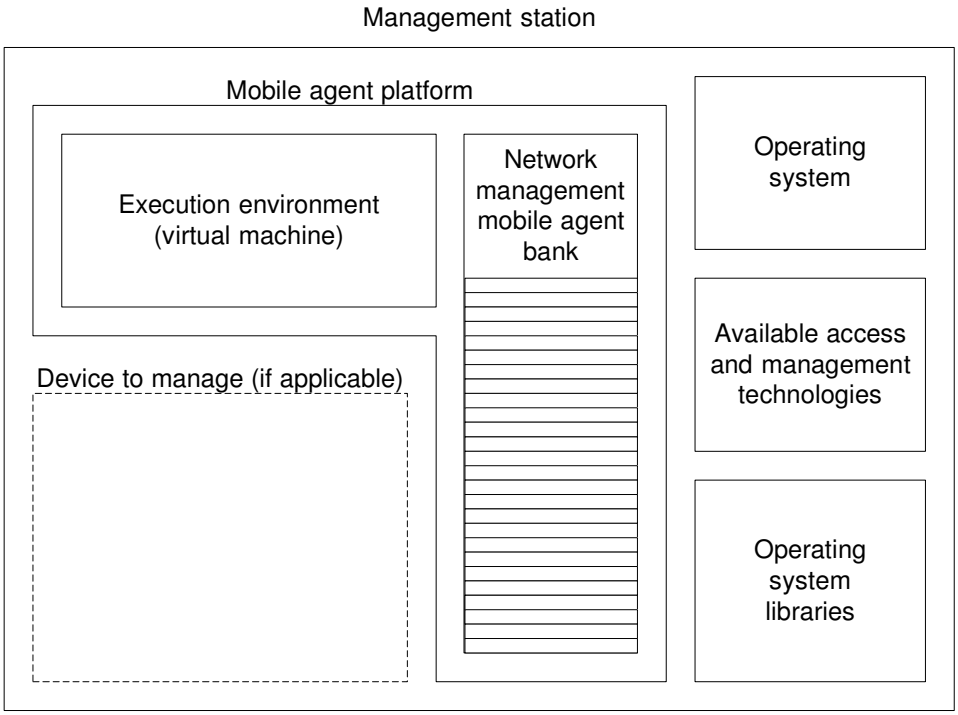

Fig. 2: Detailed view of a management station

and stationary agents on the same place to use as few network resources as possible.

Mobile agents that have to manage the network with a global view use proxy communications to access technologies and functionalities that are hidden inside the stationary agents. They therefore lower their need to carry technology dependent code. The framework favors an installation of these agents on each relevant management station. To see how proxy communications are implemented for the framework, we refer the reader to Fig. 3.

Access to a stationary agent works like a lookup mechanism. For example, let's say that a mobile agent has to manage a device named A. It also knows the interfaces needed for this network management task. The mobile agent then tries to find a stationary agent that implements the interface and has the ability to manage the device A. To do so, it uses a basic set of tests on each stationary agent installed locally. One test allows the mobile agent to test if a stationary agent is able to manage the device. A management station should at least provide one implementation of each interface for each device it has to manage. Otherwise, some devices may not be manageable. Interfaces and functions offered by the framework are given in Table 1. This table is not exhaustive, but is a good snapshot of the abilities of the framework.

Table 1: Interfaces and functions of the framework

Parent ability for all interfaces	
Functions	Description
ManagementStationAvailable	Tell if a component management system is active and may be managed using a given technology

Base	
Functions	Description
Ping	Ping a network address
VerifyService	Verify that a given service is available

Performance	
Functions	Description
GetCongestion	Returns the congestion rate
GetUtilization	Returns the utilization rate

Routing	
Functions	Description
NextComponents	Return the next components physically connected to the device
NextComponent	Return the next component physically connected to the device to join a given destination

Interrogation	
Functions	Description
GetInterfaceAddresses	Returns all addresses of an interface
GetInterfaceInformation	Returns useful information about the interface and its state
GetInterfaceIndex	Returns the interface number given a network address
GetValue	Returns the value of a variable or a state
IsDeviceAnswering	Try to reach the device and return true if it's a success

Table 2: Stationary agents of the framework and implemented interfaces

Stationary management agents	Implemented interfaces
SSnmp	Interrogation
	Performance
	Routing
SBase	Base
SWindows	Query
	Performance
	Routing

Table 3: Management table used for all tests

Device	Management station
Router Montreal	Montreal
Router Vancouver	Vancouver
Router Boston	Boston
Switch Fidji	Montreal
Management station Montréal	Montreal
Management station Vancouver	Vancouver
Management station Boston	Boston

We used three stationary agents in our framework. These agents are listed in Table 2.

A stationary agent SBase implements server-side functions that are not dependent on the device management technologies. SSnmp is used for network devices and SWindows is used to manage Windows workstations that run mobile agent platforms and are considered as a part of the network to manage.

Diagnostic Agent: One fact is that a network failure could cause many alarms and cause many direct or indirect failures. The diagnostic mobile agent is used when a failure occurs between a source and a destination. It is informed of these two parameters, as well as the port used and nothing more.

The diagnostic agent never stops on the first failure. For this reason, its first task is not to diagnose, but accumulate a series of proofs containing facts and places where these proofs are found. Then, at the end of the proof finding phase, it can establish a diagnostic. The proof finding phase ends when the diagnostic agent is unable to move further has moved on or near the destination or has no clues on how to continue (management system down or no route to host). Before terminating, it may try to launch a search agent that returns with an alternate path to the next element. If this agent is slow, a timeout tells the diagnostic agent to continue without waiting longer. The last phase is called the diagnostic phase.

Journal of Computer Science 2 (8): 646-659, 2006
ISSN 1549-3636

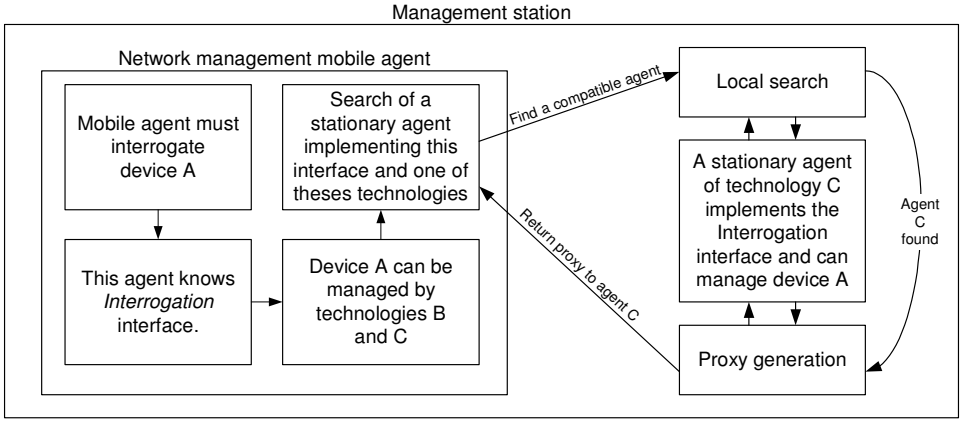

Fig. 3: Lookup mechanism to find a stationary agent

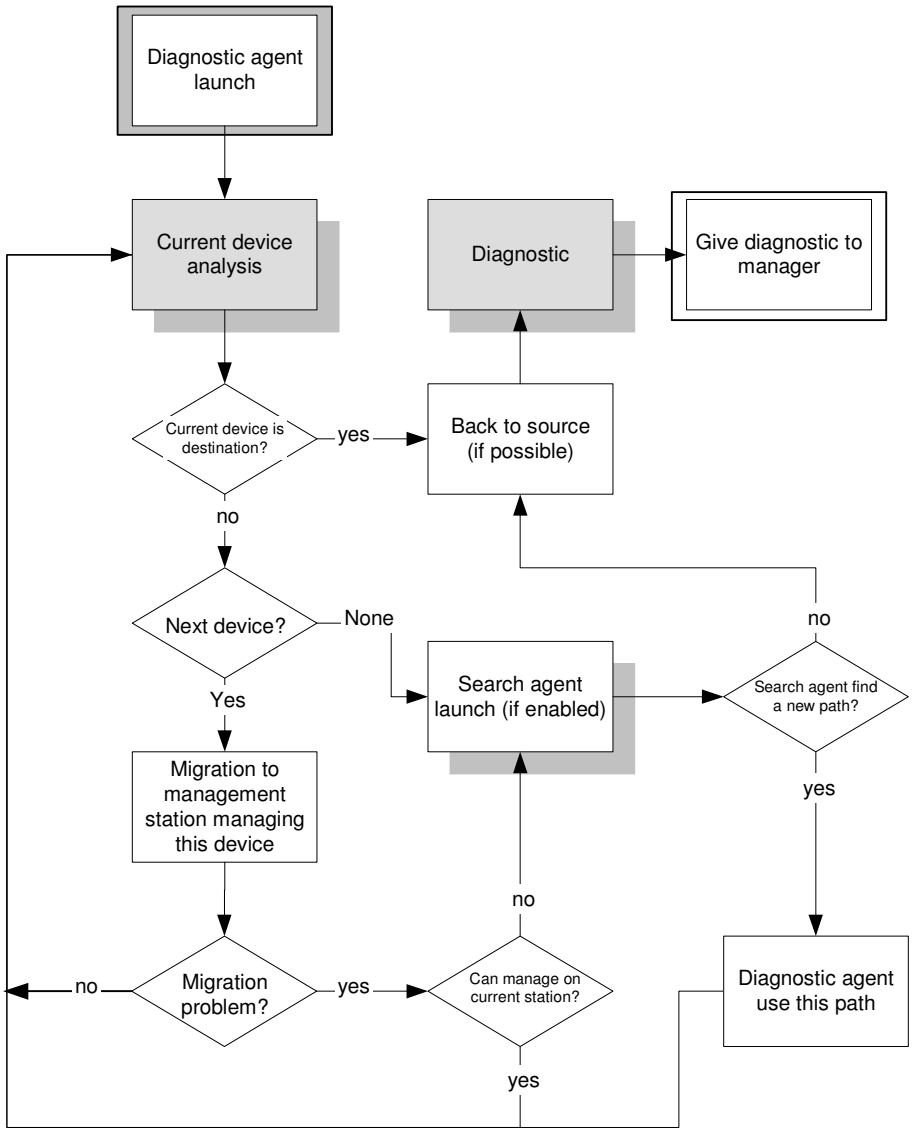

Fig. 4: Diagnostic agent global algorithm

Corresponding Author: Samuel Pierre, Department of Computer Engineering, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-Ville, Montréal (Québec), Canada H3C 3A7
Tel: +1 (514) 340-4711, Fax: +1 (514) 340-4658

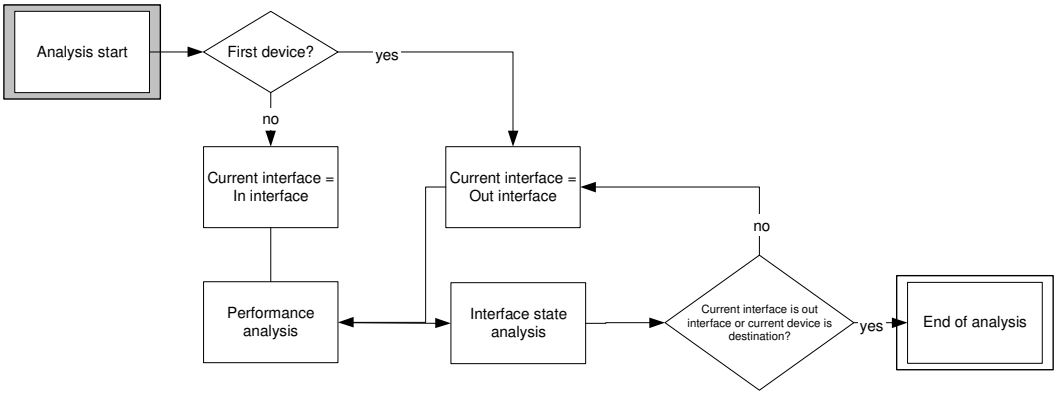

Fig. 5: Detailed analysis phase

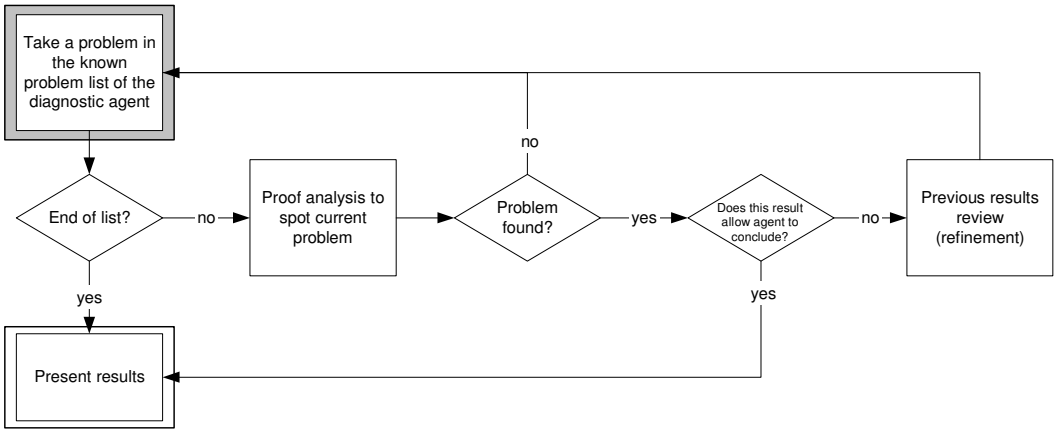

Fig. 6: Detailed diagnostic phase

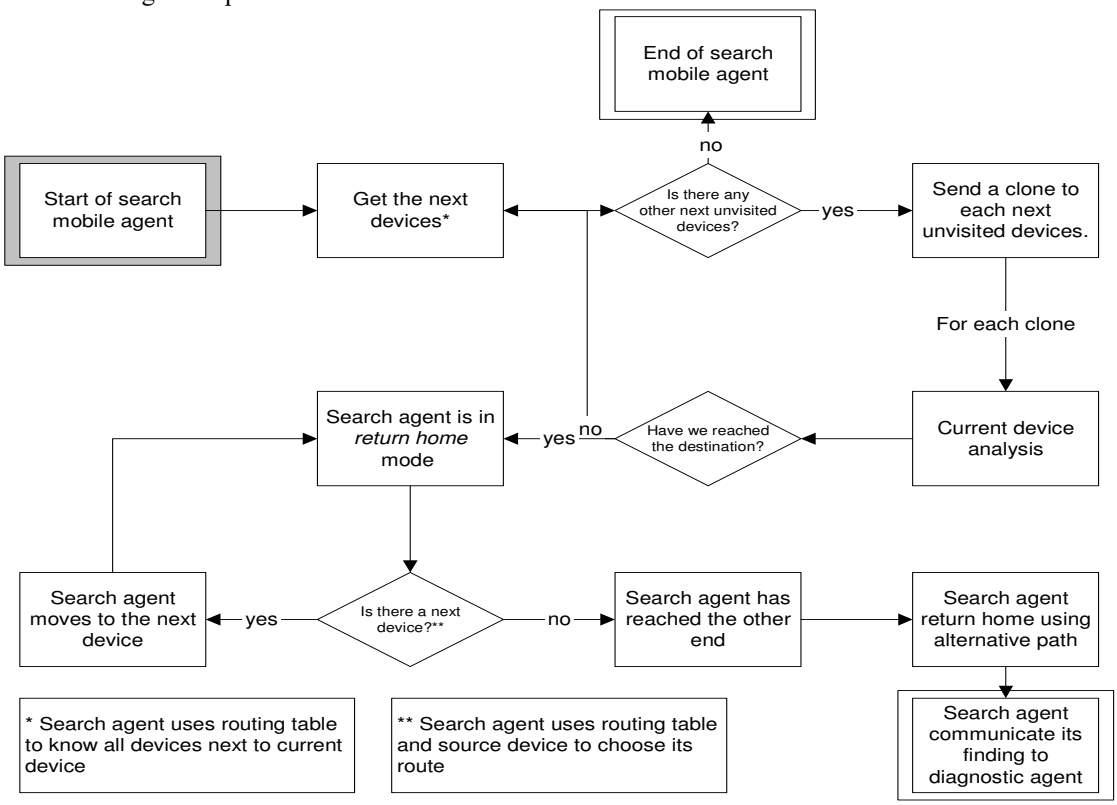

Fig. 7: Search agent algorithm

Table 4: Specification of main element of test networks

Description	Manageable	Operating system	Interfaces	Mobile agent platform
Intel Pentium 4 1.7Ghz 256meg	yes	Windows 2000 SP3	1x100 Mbit/s	GrassHopper 2.2.4b JRE 1.3.1
Intel Pentium 4 1.7Ghz 256meg	yes	Windows 2000 SP3	1x100 Mbit/s	GrassHopper 2.2.4b JRE 1.3.1
Intel Pentium 4 1.7Ghz 256meg	yes	Windows 2000 SP3	1x100 Mbit/s	GrassHopper 2.2.4b JRE 1.3.1
Router Cisco 3640	yes	IOS 12.2	1x100 Mbit/s 1x10 Mbit/s 1xATM155Mbit/s	not applicable
Router Cisco 3640	yes	IOS 12.1	1x100 Mbit/s 2x10 Mbit/s	not applicable
Router Cisco 3640	yes	IOS 12.2	1x100 Mbit/s 1x10 Mbit/s 1xATM155Mbit/s	not applicable
Switch Cisco Catalyst 8500	yes	IOS 12.0	2xATM 155Mbit/s	not applicable
Hub 4 ports	no	not applicable	4x100 Mbit/s	not applicable

Table 5: Results of Test 1 for all three cases

Session	Management station		Network failure	Failure cause			Solution	Response time (s)		
	Source	Destination		unique	identified	near	rating	1	2	3
1	Mtl	Bos	Link ATM0/0/1	**1**	1,2,3		**best**	70.87	**2.26**	3.26
2	Mtl	Bos	Link ATM0/0/0	**1**	1,2,3		**best**	66.74	63.45	**41.17**
3	Mtl	Bos	Interface ATM0/0/1 on Fidji (admin down)	**1**	1,2,3		**best**	68.89	3.13	**2.14**
4	Mtl	Bos	Interface ATM0/0/0 on Fidji (admin down)	1,2,3	1,2,3		same	69.06	64.09	**40.62**
5	Mtl	Bos	Interface ATM3/0.1 on Bos (admin down)	**1**	**1**	2.3	**best**	60.80	63.48	**40.46**
6	Mtl	Bos	Link Bos station to Hub		1,2,3		same	85.87	37.97	**18.84**
7	Mtl	Bos	Interface ATM3/0.1 on Mtl (admin down)	1,2,3	1,2,3		same	67.97	3.61	**2.36**
8	Mtl	Van	Link Van router to Mtl	1	1,2,3		**best**	26.94	3.45	**2.38**
9	Mtl	Van	Interface E0/0 on Mtl (admin down)	1,2,3	1,2,3		same	50.30	2.41	**2.09**
10	Mtl	Van	Interface E0/1 on Van (admin down)	**1**	**1**	2.3	**best**	67.32	61.69	**39.11**
11	Mtl	Van	Router Van crashed		1,2,3		**best***	14.73	**2.28**	3.16
12	Mtl	Van	Service to reach not started on Van station	1,2,3	1,2,3		same	9.37	7.52	**6.96**
13	Mtl	Van	Van station crashed		1,2,3		same	141.18	39.23	**14.58**
14	Mtl	Van	Link Van station to Van router		1,2,3		same	69.45	29.80	**7.22**
15	Van	Mtl	Interface E0/0 (admin down)	**1**	1,2,3		**best**	70.03	62.30	**39.94**
16	Van	Mtl	Crash		**1**		**best**	51.45	67.21	**40.65**
17	Van	Bos	Interface E0/0 on Van (admin down)	1,2,3	1,2,3		same	55.62	5.34	**2.06**
18	Van	Bos	Interface F2/0 on Bos (admin down)	1,2,3	1,2,3		same	96.29	30.07	**6.02**
19	Bos	Van	Interface E0/0 on Bos (admin down)	1,2,3	1,2,3		same	57.65	2.66	**2.44**
20	Bos	Mtl	Link Mtl station to Mtl router		1,2,3		same	87.56	28.88	**6.09**

These phases are detailed in Fig. 4, where a white box inside a gray box indicates the start of the algorithm while a white box inside a white one indicates a possible end. Grayed boxes indicate algorithm portions that are detailed later in Fig. 5 and 6.

The proof finding phase starts when the algorithm starts, it ends when the Diagnostic phase is reached and it may be suspended when the mobile agent uses an alternate path given by the search agent. The first thing that the diagnostic agent does is a full analysis of the current device. It then tries to know which device is next on the path between source and destination. By asking the management table, the mobile agent can know which management station is responsible for this device and it tries to migrate on that station. If it is a success, the mobile agent restarts its analysis on the current device and the new management station. If it's not, it tries to manage the device from its current management station. If successful, the algorithm restarts to perform analysis. If not, this tells the diagnostic agent that it has reached a point where it cannot obtain more information. It then has the option of establishing a diagnostic or launching a mobile agent to help find an alternate route (search agent will be explained later). The analysis step is described in Fig. 5. This phase is dependent on which network failures we want to be able to find. The mobile agent does a series of tests without trying to diagnose. A possible optimization here would be to limit the mobile agents to run superfluous tests. For now, this phase has no intelligence.

The analysis phase examines a series of facts. These facts are collected in a proof list which is inserted in a path list. The path list is built by collecting information on each interface on the real path between the source and destination. The next detailed phase is the diagnostic phase presented in Fig. 6.

The mobile agent intelligence is mostly concentrated in this phase. It tries, using refinement and testing known cause with collected proofs, to know the best location and possible cause that fit current facts about the network. This diagnostic is usually more precise if more facts are found about the problem. It never assumes that the last fact collected is the more relevant for the location or the problem.

Search agent: The search agent clones itself on each route it finds on a given node. Its goal is to find the destination using another path in the network that routing tables may not contain. When it finds the destination, it then tries to come back to the source using routing tables. When it finds a point where it cannot move using these tables, this lets it know that this may be the other end that the diagnostic agent was trying to reach. It then reuses the alternate path to come back to the place where the diagnostic agent is, to give it the extra information. The diagnostic agent then suspends its proof finding phase to move to the element found by the search agent using the alternate path. Arrived at destination, it restarts its proof finding phase. This is a summary of the complete algorithm found in Fig. 7.

In case the search agent never returns, the diagnostic agent is still able to give a good estimation of the problem just like a remote management solution. The Search agent is an addition that takes advantage of multi-path networks. To limit its spawn, a hop counter is implemented to terminate itself after too many jumps. This maximum hop value should be set carefully according to network scale and desired precision and performance.

Interactions: To clearly see how the search and diagnostic mobile agents works together, let's look at a brief example illustrated in Fig. 8.

In this case, the link between devices A and B is broken. Normally, this will cause one network interface on each of these devices to be automatically deactivated (the operational down state). If the diagnostic mobile agent is used alone, it will see only one deactivated interface. Knowing that the other side is also automatically deactivated will help conclude that something between these two interfaces has gone wrong. If that other interface has been deactivated manually, it becomes apparent that only this interface is the problem. In our case, being able to find an alternate path enables the diagnostic agent to collect more facts about the failure and gives a better diagnostic. This path finding is handled by the search agent. In Fig. 8, the diagnostic agent is stopped at device A. It launches a search agent that finds an alternative route using devices D and E. The search agent comes back to inform the mobile agent of this alternate path. The diagnostic agent may then use this alternate path to pursue its analysis phase on the other end of the failure. It is important to note that on this alternate path, the diagnostic agent does not collect information about devices D and E.

Fine tuning: Our framework leaves mobile agent code on each management station. The mobile agent code is implemented in a class. The real mobile agents that move on the network inherit this class without implementing any new functions. This way, we can tell Grasshopper to only move that lighter inheriting class and install the real code on each management station as core classes. Grasshopper never moves core elements such as its own platform classes and java native classes. What is then moved is only data and execution state and this increases the responsiveness of the system and limits traffic. One drawback is that mobile agent code cannot be updated dynamically. This technique was inspired by the JAMES[5] architecture which uses a more complex system. It uses version checking and only downloads mobile agent's code as needed.

Another important technique that we used was to make sure to drop useless data before each movement. This practice is strongly suggested. By useless data, we mean information that is not used anymore, redundant or easy to get at a later time. For instance, after a correlation of alarms, some of those can be typically dropped.

EXPERIMENTAL RESULTS

Tests: In our preliminary tests, it became apparent that using the diagnostic agent in conjunction with the search agent was an improvement in diagnostic precision, but had two serious side-effects: high response time and high total traffic on the network. We then chose to use two test networks to run our tests. The first one was used to show how easily the combination of the search and the diagnostic agent could locate and diagnose network fault. The easiness was based on the ability of mobile agents to enhance diagnostic precision over stationary mobile agents even if it comes at a high price. This first test network, shown in Fig. 9, offers alternate paths.

This test network also has a second goal: evaluate qualitatively the advantages of using management mobile agents in real networks. It unravels areas where mobile agents are better suited than remote management: path finding and searching. This evaluation will be part of our analysis.

On this test network, we made a first test (Test 1) involving twenty random single faults. These twenty faults were simulated for the three following cases: diagnostic and search mobile agents (Case 1), diagnostic mobile agent alone (Case 2) and stationary diagnostic agent alone (Case 3). In each case, we evaluated the precision of the diagnostic and we measured the response times. We also ran a test (Test 2)

Journal of Computer Science 2 (8): 646-659, 2006
ISSN 1549-3636

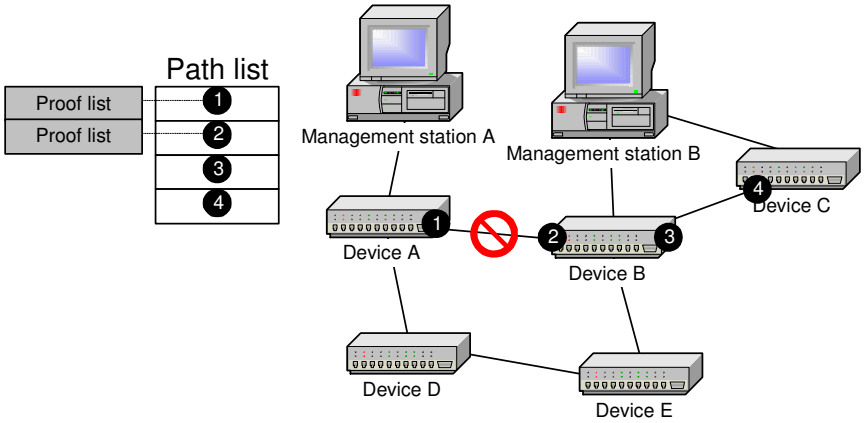

Fig. 8: Simple example using search and diagnostic agents

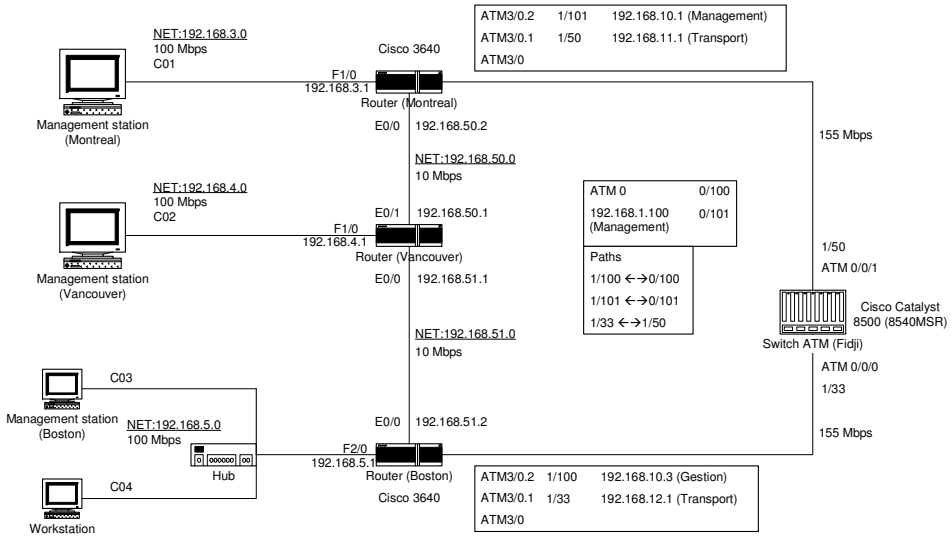

Fig. 9: First test setup

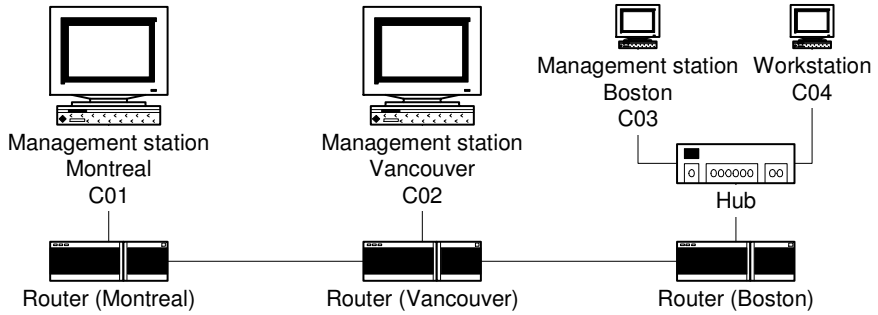

Fig. 10: Second test setup

to demonstrate that the diagnostic agent (mobile or not) was able to know that the management system was in failure. This test is important to be able to discriminate between a network failure involving a loss of service and a management system failure. Another simple test (Test 3) was run to ensure that the diagnostic agent does not mistakenly report network failure in a fully operational network.

The second test network's goal was to evaluate mobile agents in terms of raw performance. While this test has already been done for various applications including network management,

Corresponding Author: Samuel Pierre, Department of Computer Engineering, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-Ville, Montréal (Québec), Canada H3C 3A7
Tel: +1 (514) 340-4711, Fax: +1 (514) 340-4658

Table 6: Special case experiments (Test 2 and Test 3)

Session	Management Station		Network failure	Response time (s)	Cause found
	Source	Destination			
1	Mtl	Bos	None	12.09	None
2	Mtl	Van	None	7.86	None
3	Van	Mtl	None	12.40	None
4	Van	Bos	None	6.69	None
5	Bos	Mtl	None	10.02	None
6	Bos	Van	None	8.34	None
7	Mtl	Bos	Fidji management system disabled	189.24	Yes
8	Mtl	Van	Mtl management system disabled	194.02	Yes
9	Van	Bos	Bos management system disabled	195.08	Yes
10	Bos	Mtl	Bos management system disabled	195.56	Yes
11	Bos	Van	Van management system disabled	203.43	Yes

it was important to know how our framework rates against an equivalent remote approach. To achieve this, we built a test setup that enabled the mobile agent to be the closest possible to the device to manage. It is obvious that being closer to devices should lower the total traffic on the network while load balancing the charge on many devices. What is less obvious is calculating the penalty of moving network management code from one place to another[20]. The second test network uses a restrained version of the first test network. The result of this subset is a network with only one route from one host to another. The mobile agent can manage each device from the closest management station. This test network makes it easier to test the performance of mobile agents against stationary agents. The last test (Test 4) was limited to failures that imply at least one migration for the mobile diagnostic agent. It does not use the search agent to provide a fair comparison. The mobile agent returns to the source to show its diagnostic, even if it has the ability to do its diagnostic at the destination. The stationary agent uses the same diagnostic algorithm, but is limited to no mobility at all.

All tests were made with static routing and single failure scenario. These choices were made to lower the complexity of algorithms and mobile agents. Therefore, mobile agents presented in this study are built for this type of scenario only. It is a limitation for our test, but it still shows possibilities of mobile agents for more typical scenarios. By single failure scenario, we mean that the tests are conducted with only one failure, but this failure may cause more than one alarm and more than one consequence on the network.

The content of the management table described is shown in Table 3 and the specifications of the main elements of test networks are shown in Table 4.

By reviewing the elements in Table 4, it appears clearly that the focus was on using different types of devices and different transport and management technologies. Our test network can be classified as a heterogeneous network.

Results: The results of the first test are shown in Table 5. The results are for the three cases already stated which are: diagnostic and search mobile agents (Case 1), diagnostic mobile agent alone (Case 2) and stationary diagnostic agent alone (Case 3).

Here, each table shows the best results for each session in bold. The solution rating column is based on a comparison of Case 1 against the two other cases. It is interesting to note that Case 2 and Case 3 gave the same precision but not the same response times. Session 11 gave the same precision for each case, but we still rate the solution of Case 1 as best because it returned more relevant information about the problem than other cases. The failure cause found is separated in three columns. The first one, denoted unique, indicates that the exact cause of the failure was found and presented as the unique cause. This is the ideal diagnostic for a network administrator. The second one, noted identified, indicates that the cause was identified but lies among a series of other relevant but not exact causes. It may also indicate that the cause was not found precisely, but the right device was found. The last one, called near, indicates that the diagnostic was wrong, but near the cause of the failure. By using the search and diagnostic mobile agents, we always got a better or equivalent precision against the diagnostic agent alone, mobile or not. However, by using the search mobile agent, we significantly increased the response time and the total traffic on the network. The total traffic was not measured for each session but tends to be, on average, eight times greater when the search agent is used.

The next two tests are shown in Table 6. As we can see, the diagnostic mobile agent behaved like expected. The diagnostic agent is able to see a difference between a network failure and a management system failure. Also, it behaved as expected in sessions without any network failure.

All results we have shown until now on were to evaluate mobile agent technology advantages over remote solutions and stationary agents. The next results

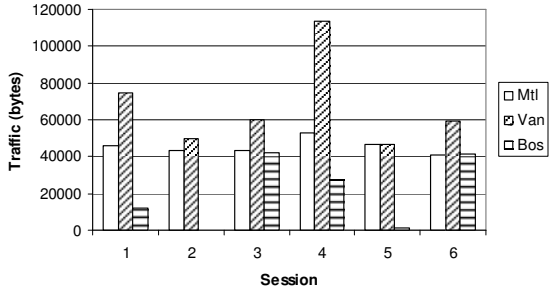

Fig. 11: Traffic measurements for diagnostic mobile agent (Test 4)

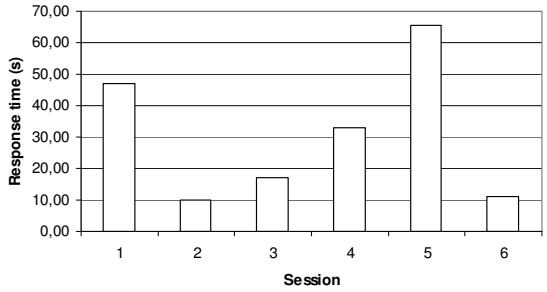

Fig. 12: Response time for diagnostic mobile agent (Test 4)

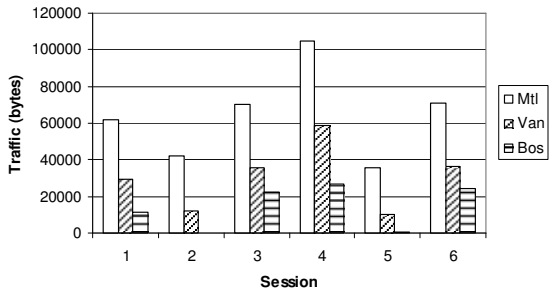

Fig. 13: Traffic measurements for diagnostic stationary agent (Test 4)

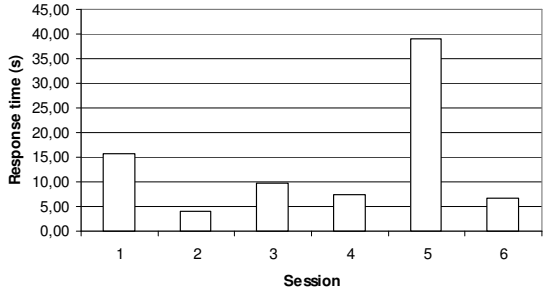

Fig. 14: Response time for diagnostic stationary agent (Test 4)

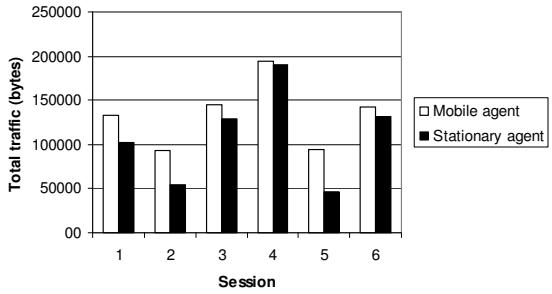

Fig. 15: Total traffic for each session (Test 4)

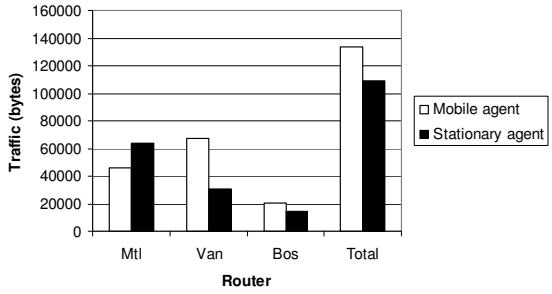

Fig. 16: Average traffic for each router and overall (Test 4)

give a better idea of the load imposed on networks by mobile agents against stationary agents.

The traffic measurements are shown in Fig. 11 and 12 for the diagnostic mobile agent and in Fig. 13 and 14 for the diagnostic stationary agent. Both agents provide the same identification precision.

Figures 11 to 14 show that the stationary agent always got a better response time. It also generates less total traffic in each case and less traffic around almost all routers.

Fig. 15 shows total traffic value for mobile and stationary agents for each session and Fig. 16 compares each traffic value for each router. To measure traffic values around router, we used out bytes value on each interface of a given router.

Fig. 15 still shows a performance advantage for the stationary agent, but with one session being almost equal (Session 4 with less than 2% of variation) and two sessions with small variation (sessions 3 and 6). Fig. 16 shows interesting statistics when we look at average values. The stationary agent only produces 18.5% less traffic on average for all sessions. Also, on router Montreal, the diagnostic mobile agent has a clear advantage over the stationary agent that is not seen in other routers.

ANALYSIS AND DISCUSSION

At first glance, the stationary agent seems to be a better solution if we only consider performance. But, we still think that mobile agents will be a better solution even in this area of comparison considering the small test network used. We base our statement on the following findings: the total traffic value for hardest network tasks is almost the same for diagnostic stationary and mobile agents (Fig. 15) and the average

network load on the first router is high for the stationary agent (Fig. 16). By hardest network tasks, we mean tasks that involve lots of information querying. Sessions 3, 4 and 6 are good examples. In these cases, the diagnostic agent needs more information about the network and the cost of movement is attenuated by the heavier network management traffic. By contrast, Session 5 of Test 4 (Fig. 15) involves few information querying. It is also the worst performance of the diagnostic mobile agent against the stationary agent because the cost of movement is high compared to the network management traffic involved. In a small network, the cost of remote management is not high enough to switch to mobile agent management. But our results let us think that mobile agents will unveil a better performance on bigger networks with most demanding and many simultaneous network tasks[26,27]. Another point of interest is the network load imposed on equipment near the remote management station. Fig. 16 shows that if we would like to execute more than one management task at the same time, the first router may become a bottleneck. It is now a fact, that mobile agents have a higher response time and may create more traffic overall. However, one main interest about them is their potential of repartition of the network load on the whole network. One bad point about our results is the high traffic variation of 54.8% imposed by mobile agents on the Vancouver router. This is mainly due to the diagnostic agent having to return home and may be optimized further by only sending a report to the source if the result is needed at the source.

From here, we did a performance analysis. We are also interested, in the mobile agent domain of research, to find tasks that are enabled only by mobile agents. So far, it seems that there is no such task. However, what we saw in our experiments are tasks that are more easily executed by mobile agents. For example, we have demonstrated that the search and diagnostic mobile agents were able to find more precisely a cause of a network failure by finding alternate paths to gather more data about the failure (Table 5). In seven of the twenty random failures, they were able to find a better solution. In these seven cases, mobile agents were able to gain access easily to interesting information about the failure that remote management was not able to see. This precision seems to be related to the number of alternate paths a network offers. Also, another point of interest is that diagnostic mobile agents can still work efficiently in an unreliable network whereas its remote counterpart may have a hard time doing the same task. Finally, they offer ways to use existing network management utilities and facilities by their ability to access local resources directly and efficiently.

CONCLUSION

In this study, we proposed, after a brief introduction, a network management framework using mobile agents and we have presented important parts of that framework. Next, we have presented implementation remarks and finally, experimental results have been performed. Experimental results show that some network management tasks can be more easily executed by mobile agents. In particular, search and diagnostic mobile agents are able to find more precisely a cause of a network failure by finding alternate paths to gather more data about the failure. Moreover, the two objectives presented in the introduction were reached. The first one is to create a network management framework using mobile agents in order to investigate the utility of them in real networks and applications and the second one is that the created framework can be utilized on existing networks.

Our framework has many limitations that must be overtaken before using mobile agents for real applications. Security and fault-tolerance are probably the greatest issues that mobile agent systems must face. The framework is also still limited in its management capabilities. It only offers a small set of management functions that should be improved before it could be ready to be used by mobile agents for any network task. The projected performance improvement is not yet proven for mobile agents of our framework. The diagnostic mobile agent is also limited in the kind of network and errors it can handle.

Limitations stated in the last paragraph can all constitute future ways to improve the current framework. In[28], Borselius offers an overview of security issues and measures for mobile agent systems. The set of management functionalities that it can handle could be enhanced. By enhancing this set, it would become possible to build mobile agents that execute network management tasks other than network fault localization and diagnostic. Mobile agents presented in this study could still be improved in many ways. For example, the diagnostic could handle much more failure causes and could be adapted for dynamic routing protocols. The diagnostic algorithm could learn from past errors instead of being tied to a deterministic expert system. Even the expert system can be improved using better expert system techniques, scripts and rules. Network fault location and diagnostic are two tasks that network administrators would want to be automated efficiently and confidently. Since mobile agents we presented are limited, there is still a lot of work to do only in that area.

REFERENCES

1. Liotta, A., G. Pavlou and G. Knight, 2002. Exploiting agent mobility for large-scale network monitoring. IEEE Networks, 16: 7-15.

2. Rubinstein, M.G., O. Duarte and G. Pujolle, 2002. Scalability of a network management application based on mobile agents. J. Communications and Networks, 5: 240-248.
3. Bieszczad, A., B. Pagurek and T. White, 1998. Mobile agents for network management. IEEE Communications Surveys, 1: 1.
4. Bellavista, P., A. Corradi and C. Stefanelli, 2003. An open secure mobile agent framework for systems management. J. Network and System Management, 7: 323-339.
5. Silva, L.M., P. Simoes, G. Soares, P. Martins, V. Batista, C. Renato, L. Almeida and N. Stohr, 1999. JAMES: A platform of mobile agents for the management of telecommunication networks. Proc. Intl. Workshop on Intelligent Agents for Telecommunications Applications, IATA'99, Springer-Verlag, pp: 77-95.
6. Carleton University, The Network Management and AI Laboratory. <http://www.sce.carleton.ca>.
7. Pagurek, B., Y. Wang and T. White, 2000. Integration of mobile agents with SNMP: Why and how. Proc. IEEE/IFIP Network Operations and Management Symposium, NOMS'00, pp: 609-622.
8. White, T. and B. Pagurek, 1998. Towards multi-swarm problem solving in networks. <http://dsp.jpl.nasa.gov/members/payman/swarm/white98-icmas.pdf>Proc. Intl. Conf. on Multi-Agent Systems, ICMAS '98, pp: 333-340.
9. Rossier, D. and R. Scheurer, 2002. An ecosystem-inspired mobile agent middleware for active network management. Proc. Intl. Workshop on Mobile Agents for Telecommunication Applications, MATA'02, pp: 73-82.
10. Di Caro, G. and M. Dorigo, 1998. Ant colonies for adaptive routing in packet-switched communications networks. Proc. Parallel Problem Solving from Nature, PPSN'98, pp: 673-682.
11. Baràn, B. and R. Sosa, 2000. A new approach for antnet routing. Proc. Intl. Conf. on Computer Communications and Networks, ICCCN'00, pp: 303-308.
12. Schoonderwoerd, R., O. Holland and J. Bruten, 1997. Ant-Like agent for load balancing in telecommunication networks. Proc. Intl. Conf. on Autonomous Agents, Agent'97, pp: 209-216.
13. Bohoris, C., G. Pavlou and A. Liotta, 2003. Mobile agent-based performance management for the virtual home environment. J. Network and System Management, 11: 133-149.
14. Koon-Seng, L. and R. Stadler, 2003. Weaver: Realizing a scalable management paradigm on commodity routers. Proc. IFIP/IEEE Intl. Symp. on Integrated Network Management, IM'03, pp: 409-424.
15. Bossardt, M., L. Ruf, B. Plattner and R. Stadler, 2000. Service deployment on high performance networks nodes. Proc. IEEE/IFIP Network Operations and Management Symposium, NOMS'02, pp: 915-917.
16. Rubinstein, M.G., O. Duarte and G. Pujolle, 2000. Reducing the response time in network management by using multiple mobile agents. Proc. Third Intl. Conf. on Management of Multimedia Networks and Services, Kluwer Academic Publishers, pp: 253-265.
17. Boyer, J., B. Pagurek and T. White, 1999. Methodologies for PVC configuration in heterogeneous ATM environments using intelligent mobile agents. ftp://ftp.sce.carleton.ca/pub/netmanage/jb-mata99.ps.gz. Proc. of Mobile Agents for Telecommunications Applications, MATA'99, pp: 211-228.
18. Zapf, M., K. Herrmann and K. Geihs, 1999. Decentralized SNMP management with mobile agents. Proc. of the Sixth IFIP/IEEE Intl. Symp. on Distributed Management for the Networked Millennium, pp: 623-635.
19. Gavalas, D., D. Greenwood, M. Ghanbari and M. O'Mahony, 1999. An infrastructure for distributed and dynamic network management based on mobile agent technology. Proc. IEEE Intl. Conf. on Communications, ICC'99, pp: 1362-1366.
20. Gavalas, D., D. Greenwood, M. Ghanbari and M. O'Mahony, 2002. Hierarchical network management: A scalable and dynamic mobile agent-based approach. Computer Networks, 38: 693-711.
21. Putzolu, D., S. Bakshi, S. Yadav and R. Yavatkar, 2000. The phoenix framework: A practical architecture for programmable networks. IEEE Commun. Mag., 38: 160-165.
22. White, T., B. Pagurek and A. Bieszczad, 1999. Network modeling for management applications using intelligent mobile agents. J. Network and Systems Management, 7: 295-321.
23. Timon, C. Du, Eldon Y. Li and An-Pin Chang, 2003. Mobile agents in distributed network management. Communications of the ACM, 46: 127-137.
24. IKV++, "Grasshopper - The Agent Platform," http://www.grasshopper.de, 2004.
25. AdventNet, Inc., "AdventNet SNMP API," http://www.adventnet.com, 2004.
26. Baldi, M., S. Gai and G.P. Picco, 1997. Exploiting code mobility in decentralized and flexible network management. Proc. Intl. Workshop on Mobile Agents, MA'97, pp: 13-26.
27. Puliafito, A. and O. Tomarchio, 1999. Advanced network management functionalities through the use of mobile software agents. Proc. Intl. Workshop on Intelligent Agents for Telecommunications Applications, IATA'99, Springer-Verlag, pp: 33-45.
28. Borselius, N., 2002. Mobile agent security. Electronics & Communication Engineering J., 14: 211-218.