Distributed Network Monitoring using Mobile Agents Paradigm

Farhad Kamangar, David Levine, Gergely V. Záruba, and Navakiran Chitturi

Department of Computer Science and Engineering
The University of Texas at Arlington, Texas
{kamangar,levine,zaruba,chitturi}@cse.uta.edu

Abstract. Traditionally network monitoring and management has been done using predominantly centralized techniques. Mobile agents have been proposed as an alternative to this centralized approach. In this paper we propose a novel approach for distributed and dynamic network monitoring, using mobile agents. We use the IBM Aglets system and show how a Java-based distributed network monitoring application can use this paradigm for efficient data collection and analysis and adapt to variations in network characteristics.

Keywords: Network Monitoring, Network Management, Mobile Agent, Java, Distributed, Dynamic, Network Modeling, Fault-tolerance and Delegation.

1. Introduction

Network monitoring and management has become necessary due to the proliferation of computers and the immense growth of the Internet. Different paradigms have been proposed for network management using mobile agents [1]-[4]. Sahai and Morin [1] proposed an architecture comprising of managers, servers and management agents. The network management load is equally distributed between managers and servers. The servers have replicated databases. The framework uses MAGENTA environment to provide the capability of sending, receiving and storing mobile agents. Gavalas [4] suggests a framework for network monitoring applications using four components: Manager application, Mobile Agent Server (MAS), Mobile Agent Generator (MAG) and Mobile Agents (MA). The manager application co-ordinates monitoring of network elements and has a GUI. The MAS receives, instantiates, executes and dispatches mobile agents. The MAG constructs customized MAs in response to service requirements. Puliafito [2] uses MAP, a platform for development and management for mobile agents, for network management. MAP is MASIF (Mobile Agent System Interoperability Facility) standard compliant. By using MAP services a management application for basic operations of various agents is developed. These agents perform simple tasks but their combination could be used to perform complex management actions. Kona [3] proposes the MAN (Mobile Agent Based Network Management) framework that supports a number of areas like code mobility infrastructure, network simulator and network management tools. The Mobile Code Daemon (MCD) is a process that runs inside a Java Virtual Machine (JVM) on every network element and listens on a UDP or TCP port for requests to receive Java code. In this paper, we propose a new network management framework, which enables us to develop a distributed, dynamic, scalable and fault-tolerant management tool.

2. Proposed Distributed Network Monitoring Architecture

We propose an architecture that is highly modular, distributed and fault-tolerant in nature. It includes a GUI Module (GM), Web Access Module (WAM), Manager or Monitor Module (MM) and various Dedicated Mobile Agents (DMA) for different monitoring or management purposes. All these are built to work together in an agent execution environment. We use Aglets, the IBM mobile agent system, for the agent environment. The above modules are built in order to blend into the Tahiti server (Aglet Server – AS), which is the Aglets execution environment. We use replicated databases and make the MM itself network-aware thereby increasing flexibility and fault-tolerance. The language used is Java [5] because it has some very useful features like platform independence, secure execution, dynamic class loading, multithread programming, and object serialization.

2.1. How the architecture works

The user has access to the monitoring system through the web using the GUI Module (GM). All the queries and management tasks are in the form of HTTP requests [6] and go to the WAM of an agent server (Tahiti). This WAM has been designed to process the HTTP requests and give out appropriate html pages or java applets and also interpret the various monitoring and management requests. The MM has the primary responsibility of monitoring the network. The DMAs are either made to reside on the same server or move around to reside on another host with a Tahiti server according to user preference and proximity to the monitored nodes. The DMAs are mobile and can be retracted from or dispatched to other hosts, deactivated or activated and disposed according to the MM's discretion. Special DMAs are used to move through different machines to collect summary information or do some local processing and take the results along to other hosts in their itinerary. The MM is provided on a few (or all depending on the trade-offs) of the host computers. At any particular instant of time only one MM is the Master MM and others are more dormant in the sense that they are used only to check on the availability of the Tahiti server hosts and do some local assessment of the network conditions. DMAs can be located optimally in order to minimize network traffic incurred by monitoring and the delay in obtaining monitoring data. The system is organized in a hierarchical or a non-hierarchical fashion depending on the type of networks being monitored or managed. In the case of networks consisting of sub-networks, monitoring tasks requiring first the *collection* and then the *aggregation* of raw data from each sub-network can be implemented by distributing agents at each intermediate level (co-operating with MMs).

Each agent will be in charge of producing the required level of data aggregation for a specific sub-network and providing high-level information to the other agents in the hierarchy. Alternatively, a non-hierarchical MA organization can be used, which is particularly suited to non-hierarchical networks and non-hierarchical monitoring tasks. Using a variety of design patterns, we produce a robust and flexible network monitoring implementation. We have to remember that the life span of a monitoring agent is independent of the computing process that creates it. Figure 1 shows the overall framework for the new distributed network monitoring system.

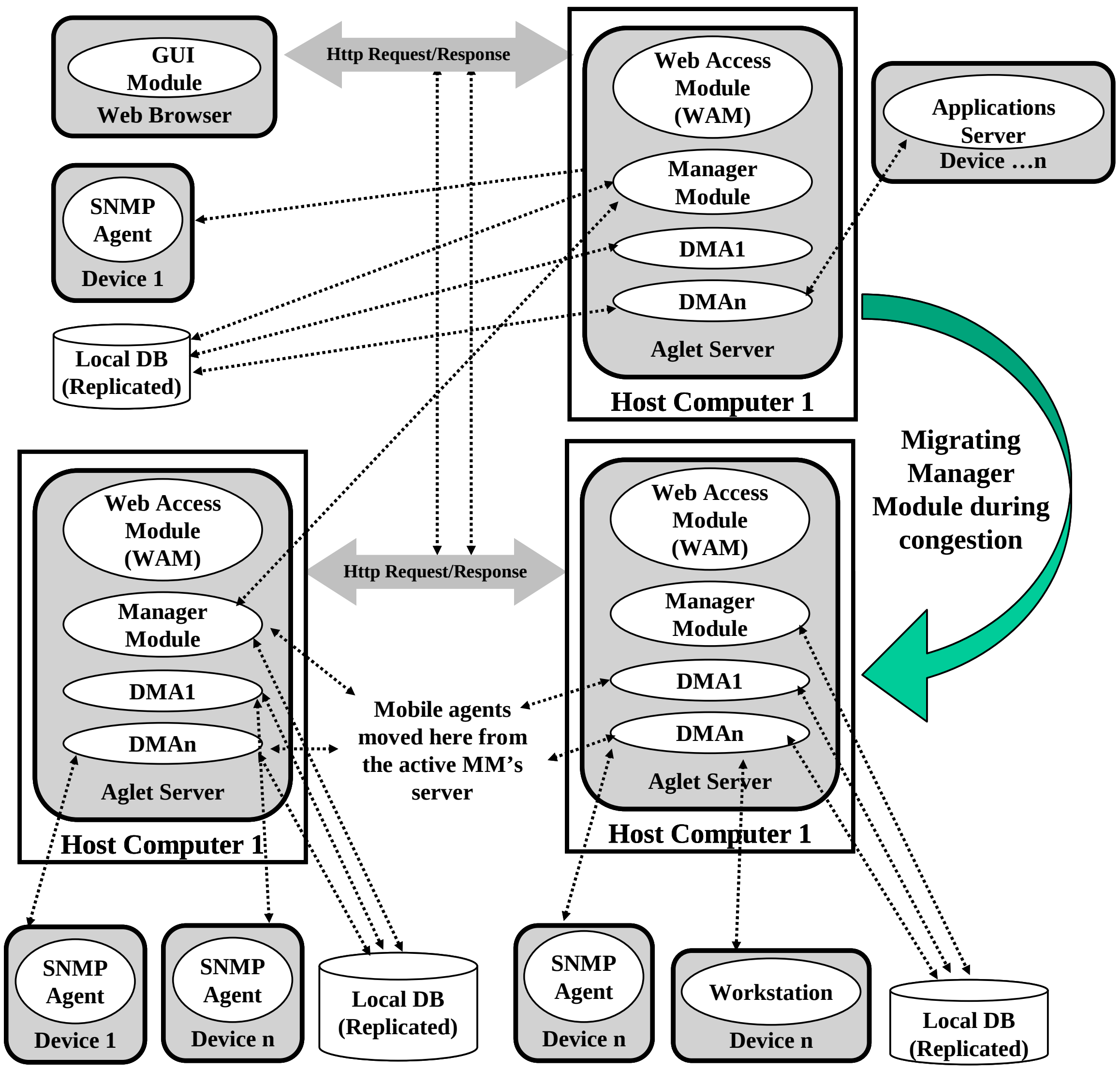

Figure 1. Framework for the distributed network monitoring system.

2.2. Functionality

We provide a brief overview of the functionality of our monitor, which encompasses data collection, statistics, trend analysis, report generation and distribution control.

2.2.1. Data Collection

If a network device has an SNMP daemon running on it then we use SNMP protocol to get specific network performance measures. If a device or host doesn't have SNMP then we make use of other methods (ping, trace route, etc.) to obtain performance measures. Data is collected using Poisson sampling intervals with an upper bound on sampling interval of 3 minutes. This ensures better network state assessment than periodic sampling that is predictable. For each sample collection, the time taken to make a particular query is subtracted from the sample interval. Beyond the relatively *raw* data collection capability our system will be able to perform statistical computations regularly and on demand. The specific computations performed by any DMA will vary depending on the purpose for which the agent was made initially. In some cases, the agent will perform all the required computations using data collected locally (in its own sphere of control). In other cases an agent may perform some computations, but forward some collected or computed information to another component for additional computations. In yet other cases, the same mobile agent will move over different locations and collect the necessary information based on the user's querys.

2.2.2. Statistics

Statistical summaries can be requested for nodes and/or ports of a node. Simple statistics include utilization, throughput, and error rate computations for SNMP enabled device. Measures such as availability, packet loss, and delay are used for non-SNMP devices. The data collected is stored along with the date/time value of when it was collected. Counts or values per unit time and averages per set of unit times are computed. The user may query based on various time units - in seconds, minutes, hours or days. Measurement periods of basic time units, weeks or months are utilized. This aids in summary and report production. Statistical gathering has an optional start and stop time. Peak value of measure and time during measurement period is noted and based on the option a report is generated after each measurement period.

2.2.3. Trend Analysis

It is critical to do long term trend analysis so that the need for costly or complex network upgrades can be predicted well in advance. To properly provide input for trend analysis, the data computed via the methods specified must be collected in a consistent manner over the specified period of time. It is possible to perform trend analysis on utilization, throughput, and error rate at a port or availability at a node. A periodic report is generated which may be specified by the administrator (privileged). Also, thresholds can be specified to cause alarms, for example if utilization reaches a certain level or the rate of utilization increase changes dramatically. A marked increase in error rate or decrease in availability may also trigger an alarm.

2.2.4. Report Generation

A report is generated at a specified rate or at specific times as specified by the user. The user may specify the location where the report will be delivered using a mobile agent for this purpose.

2.2.5. Configuration Change Control

A peer aglet host is able to take over the responsibility of another aglet server if the primary component, or the system in which it resides, become unavailable. This will be facilitated by periodic handshake between the Aglet servers. Also at each server, we monitor utilization of the system's resources on which they reside. If the resource is over-utilized, the MM will be notified so that alternate server may be assigned.

2.2.6. Alarm Management

An alarm occurs when a processing node becomes irresponsive or resource used by a specific AS exceeds threshold values. This is done by programming a DMA in such a way that it sends an alarm (in the form of a message) whenever a particular threshold value is compromised. It is also possible to disable the alarm temporarily. The administrator will have control over setting up alarms. The DMA can be configured to listen for traps issued by SNMP agents.

2.2.7. Graphs

Packets are sent periodically or as a Poisson stream. Lost packets are treated as having an infinite delay. Records consisting of <fromipaddress, toipaddress, performance_measure1, performance_measure2, … performance_measureN, time> are constantly stored in the local database. **Figure 2** shows the graph generated for the roundtrip time taken for a packet from one place to another against time units (a round trip time of 100 signifies that the host is unreachable). If the device is SNMP enabled then we can plot the utilization of a particular interface/port against time or errors against time as shown in Figure 3.

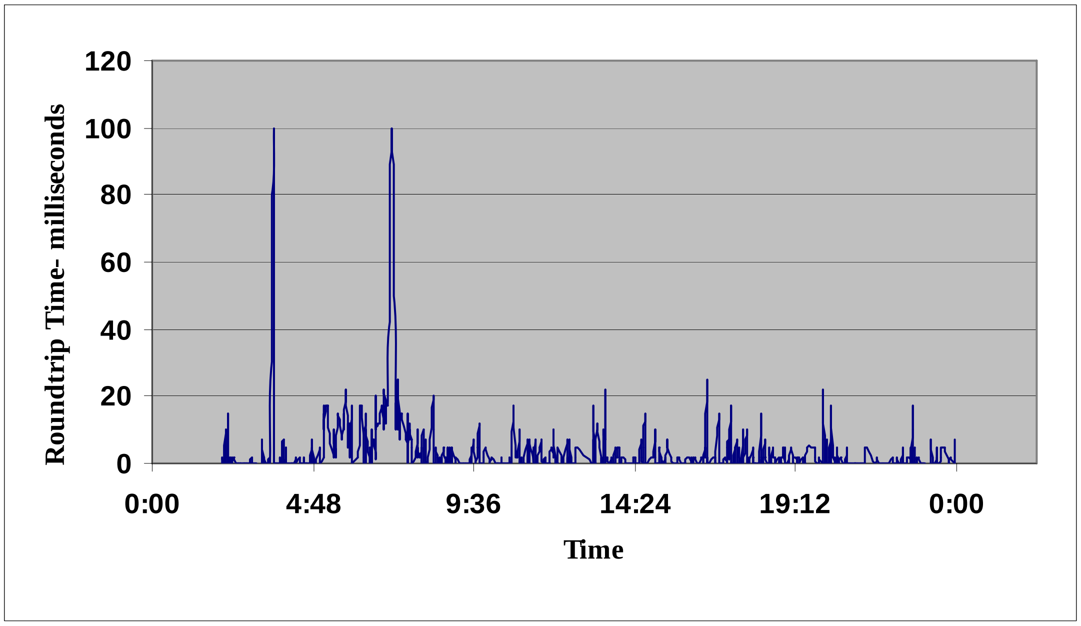

Figure 2. Average roundtrip packet travel time between two machines.

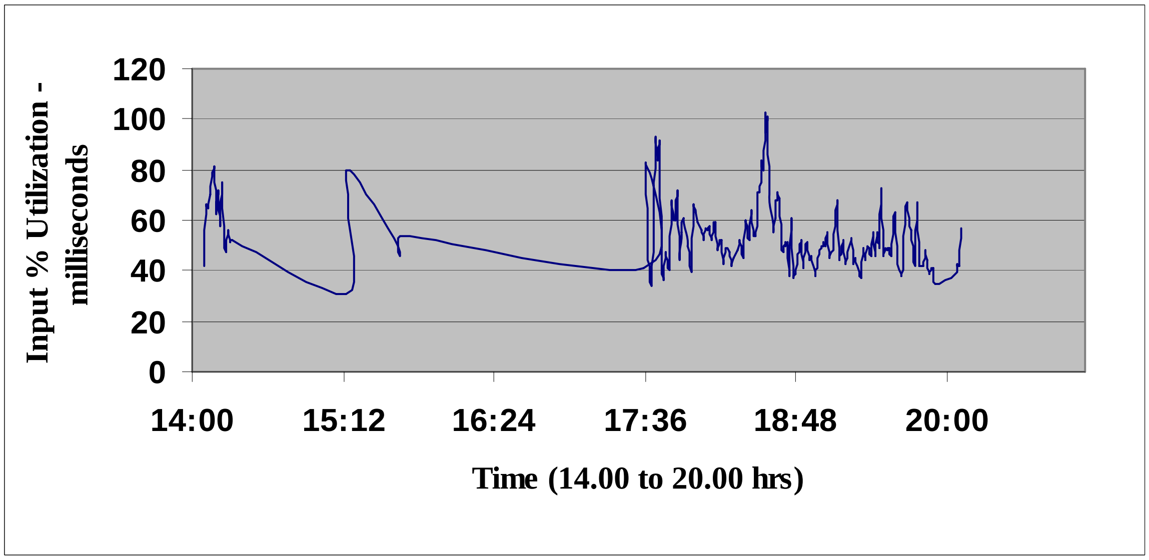

Figure 3. Percentage input utilization of a port.

2.3. Special Scenarios

Special scenarios are: congestion, station down, and link down are analyzed in this section.

2.3.1. Congestion

If a particular host is seen as a bottleneck then, the MM can dispatch a few or all of the DMAs which are running on that host to other hosts and the devices can be monitored from there. This would decrease the amount of traffic at that host thereby eliminating the bottleneck. We implemented one more feature to increase the distributed nature of the overall system, that is, to make the manager module mobile. During periods of extreme congestion at the MM's local machine, and when other MMs are not available to take up the responsibility, the Master MM would move to a new place and start its managing process from there after informing all the concerned hosts and agents.

2.3.2. Monitoring Station Down

When a monitoring station (i.e., a host) goes down, the MM detects the loss through periodic handshake between the various MMs and inventory. The MMs communicate with each other every few minutes to check connectivity to the various hosts. If there is a problem it is reported and stored in the database. An email (or paging) alert is sent to the administrator. If the host is not being able to be accessed from any other host then, the MM obtains the local information about the various devices that were being monitored by the down host and prepares appropriate mobile agents to monitor from its own location or from other nearby host and dispatches them accordingly. To check whether the host is up and it's the Tahiti sever that is not responding we use the ordinary Internet ping command.

2.3.3. Link Down

If the host is accessible from a nearby host then the MM sends an email alert indicating that the link between the MM and the host in discussion is down. It then sends a message to the nearby host's manager module to take over the handling of the host in question.

3. Monitoring using SNMP and some standard procedures

3.1. For SNMP enabled devices

In network monitoring, '*polling*' is a frequent operation. In order to indicate a system state, most often an aggregation of multiple variables is required, known as a health function (HF). For example to define the input percentage utilization of an interface we use a broadcast model, namely generate mobile agents with the specific goal and sending them to the AS from where they would monitor the device under discussion. We program the DMA in such way as to sample (all in one packet exchange) the values of ifInOctects, ifOutOctects, and sysUptime.

Then, we sample all three again (after some interval) and use the three deltas (differences) to compute this HF for a T1 line [7]:

$$Input\%utilization = \frac{Delta(ifInOctects)*8}{Delta(sysUptime)*154*2}$$

We compute likewise for the output percentage utilization. Also, in the traditional model, the 3 Object Identifiers (OIDs) are grouped into a single ‘get’ request packet. This takes a lot of space and a lot of overhead to be passed over the network, while on the other hand we use DMAs, which would compute HFs and pass over only the necessary details, thereby acting as a semantic compressor of large amounts of data.

Another major drawback with SNMP is involving ‘*bulk transfer*’ of data, like transfer of large SNMP tables. Traditionally, get-next requests are used that adversely impact network resources. With the increase in management information that needs to be transferred, the get-bulk operation could be used, but this has the intricacy of selecting max-repetitions (i.e., the number of rows to be retrieved). Finally the OID scheme contributes to this high overhead of information exchange. Therefore acquiring SNMP table locally (through successive get-next requests) and then encapsulating it before moving to the next host or returning to the manager would be far more efficient [4]. This can be done by strategic placement of the DMAs. We could also apply a filtering function on the selected SNMP tables according to our needs. In addition we could use domain or global level filtering (using the DMAs’ multi-node itineraries). This is achieved by comparing/merging the results already collected with those that have been just obtained or processed. This way we can prevent processing bottlenecks at the manager and also control the DMA’s size from growing rapidly. Regarding the DMA interaction with the SNMP agent, we have used the AdventNet SNMPv1 package freely available on the Internet [8].

For detecting a congestion problem, we simply calculate the discard rate. Similarly detecting interface errors gives us an idea of hardware, cable, or line problems. Using IpGroup and ICMPGroup variables we can detect routing problems. Similarly using TCP Group and UDP group we could find the corresponding applications or connections running on the system under consideration.

If monitoring of percentage error rates for interfaces of multiple machines is needed an itinerary based DMA is dispatched, which sequentially visits all the machines, interacting with SNMP agents at each individual node. It does local calculations and returns after completing the tour. If conventional centralized SNMP application were used, calculation of the above health functions introduces processing burden at the monitoring station and network traffic overhead.

3.2. Application Monitoring

DMAs are effectively used for system monitoring where large amount of system related data are to be collected for analysis later on. DMAs are equipped with the names of hosts to be visited, their pattern of migration through systems and the time frame in which they would be monitoring the visited hosts.

We can also monitor the availability of various services such as like HTTP, FTP, SMTP, Telnet, SSH, and IMAP by using standard probes on well-known ports and also calculate transmission delay and other parameters using public domain software like ping and trace route [9].

3.3. Data Design and Distribution

We store information obtained while monitoring along with the time value. Information is stored in various places depending on its type – in the agent itself, at a local storage pool, or at remote site to which agents may have access. We use database replication [9][11] for our framework along with information contained within agents in order to provide fault-tolerance. In our approach, agents have access to all the data available, via each other. Each agent extracts information from other agents and the database as necessary.

Appropriate privileges are implemented to access and modify the required local databases in the agents' itinerary and also to access other agents' information.

4. Experiments and Results

Our major concern is the viability of using mobile agents (i.e., aglets) for a network monitoring application. For this purpose, Aglet latency is measured in an itinerary pattern. Comparison of our approach with the client-server technique is made and the results are shown.

We monitor a campus network with multiple LANs. Multiple manager (monitoring) hosts situated in various LANs are provided. The network administrator selects one or more devices and the agent server (i.e., agent host), from where he wishes to conduct the monitoring. The monitoring agent server selection for each particular device is done so that we could minimize the network traffic generated by periodic exchange of management information (requests by the monitoring agent and responses from the devices).

Another consideration is the latency involved in moving agents around the network. This is very important when it comes to practical application of a distributed network monitor. Depending on the job, aglet size may vary from very *small* (lines of code in the order of 10^n where n <=2) to *large* (number of lines of code = 10^n, n > 2). Dedicated Mobile Agents (DMAs) used for monitoring specific devices over a sample (periodic or based on Poisson distribution) interval are rather small in size where as *itinerary* (or *summary*) agents that travel around for collecting requested information are comparatively larger. Therefore the size of an agent is an important factor to consider for efficient design of any application.

Latency is influenced by the size of an aglet in two ways. First is the time taken for the movement of the aglet across the network over to other agent servers. The time taken is also influenced by other external factors like geographic location and network congestion. Second, serialization is influenced by the aglet size. Serialization converts a Java object to bytecodes and transmission occurs in the form of a byte stream. Because aglets use object serialization mechanism to prepare for transportation over the network, there is an obvious overhead with increasing size.

For these reasons we conducted the following experiments:

We measure an agent’s time to complete a round-trip around the various agent servers. This provides us a measure for estimating the time required for summary information collection.

We also measure the effect of increasing agent size on aglet latency.

4.1. Experiment Setup

We have chosen four servers. All servers run Windows 2000 and have the Aglets Software Development Kit (*aglets1.1.0*) and the Java Virtual Machine (in JDK1.1.8) installed.

The first experiment obtained round-trip times of agents traveling around in the network over the various servers. We measure the hop time between various servers to obtain the relevant information over repeated trials. Tables 8.1 shows the results with maximum, minimum and mean hop times over the various trials between different servers in the itinerary.

Table 1. Roundtrip time between servers

Travel Between	***Maximum Hop Time (milliseconds)***	***Minimum Hop Time (milliseconds)***	***Mean Hop Time (milliseconds)***
Server A and Server B	530	180	320.25
Server B and Server C	1100	210	330.90
Server C and Server D	1200	300	428.30
Server D and Server A	1080	189	390.45

The second experiment deals with increasing the aglet size proportionally (in steps of 100 lines of code) and each time measure its effect on the aglets travel time and thus the latency. We regard server A as the *machine* from which we dispatch the *Mover* agents using a *TimeServer* agent. As we stated, server B is in the same LAN that provides for one-hop reference measurements. Servers C and D are each located on different networks.

In the experiment the *TimerServer* aglet is created on server A. This aglet spawns a *Mover* agent and dispatches the *mover* to server B. The mover aglet moves from the source host to the destination host and back. The TimerServer aglet is responsible for maintaining communication with the mover agent. The Mover agent is disposed off upon its return. The data collected involves two values of time (in milliseconds), one collected at the time of dispatching the mover agent and the other collected just after it returns. By obtaining both the timestamps on the same server (server A) we are able to eliminate any synchronization problems of the system clocks. Similar experiments are carried on for the other servers C and D separately with server A as the reference host. The round-trip time is calculated as an average of the values obtained over 10 trials (this constitutes one run). We now increase the size of the Mover agent in steps and perform the same experiment. We add additional code in such a way that it does not constitute to increase in processing time at the destination. This is done by having a conditional expression evaluated at run-time and placing the extra code inside the *if* loop of the condition. This condition is used in such a way that it always evaluates to false. Figure 4 shows the roundtrip time from server A to other servers plotted against the aglet size.

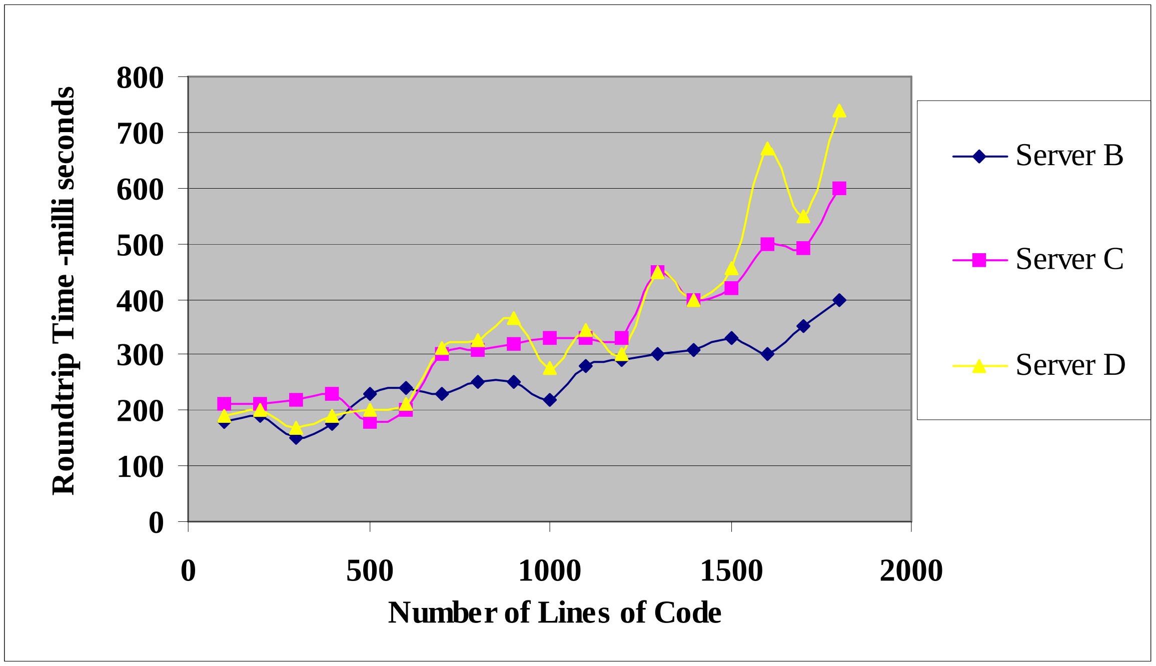

Figure 4. Aglet size Vs roundtrip times from server A to servers B, C and D.

4.2. Comparison with Client-Server Techniques

In order to show that mobile agents are applicable and also efficient for use in performing basic network management/monitoring functionality, we have compared our system with the basic client-server mechanism. The response times between C/S and mobile agent paradigms are compared as shown in Figure 5. From the machine where the manager is running a DMA is dispatched to retrieve a given number of samples (ranging from 1 to 700) from the router. This DMA resides on an agent server which is one hop away from the router. This way the DMA locally collects the required number of samples and comes back to the agent server where the manager is running. In the client-server technique, SNMP requests are sent continuously from the manager site to the router directly for the desired number of samples and the response packets are received at the manager site continuously. The time taken for both the DMA and the client-server technique to obtain the given number of samples is plotted in Figure 5. These times are calculated from the manager site. We find that the response time for retrieving smaller number of samples is almost similar. But as the number of samples increases, there is a clear advantage of the mobile agent technique. This corroborates that using mobile agents could minimize network latency by reducing the exchange of management information beyond various LANs.

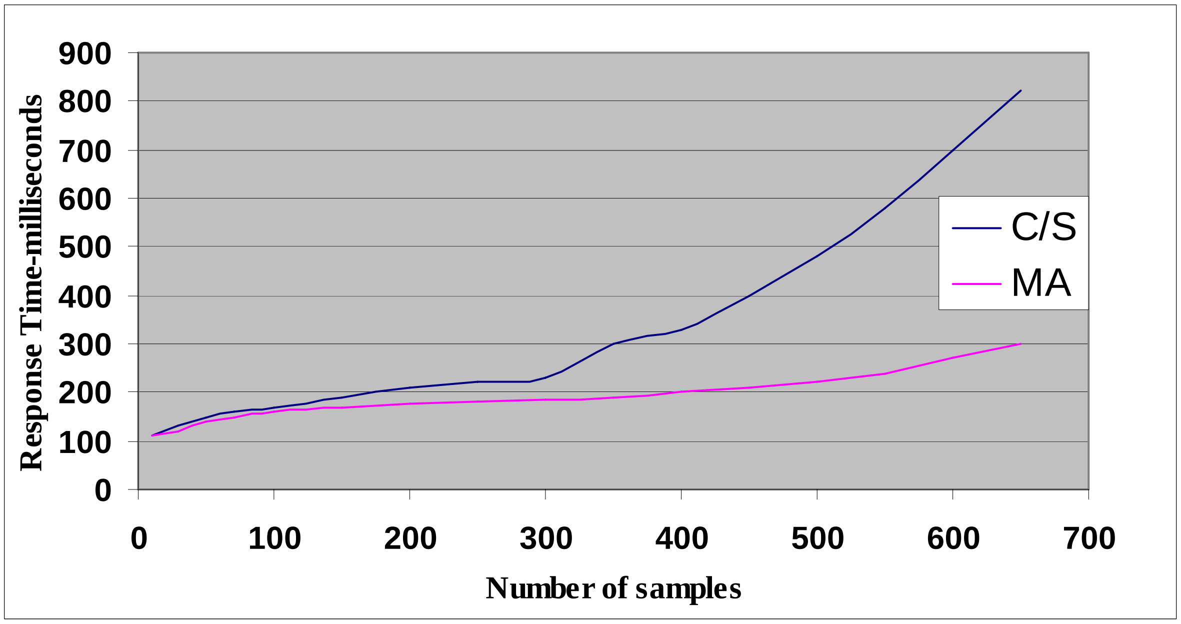

Figure 5. Response times for various samples using C/S and mobile agents.

5. Conclusions

A scalable, fault-tolerant and dynamic network-monitoring tool using the proposed framework was developed. Issues of distributed architectures were discussed and argument that mobile agents are a good way to achieve distributed network monitoring against client-server techniques was made. Experiments were conducted to verify the same. We find that mobile agents are an efficient method of reducing the effects of network latency on the network monitoring application. For collecting large number of network parameter samples, the mobile agent technique was found to outperform the client-server technique when the number of samples is large. We found that over a particular threshold size, the aglet class fails to be loaded. This factor was taken into consideration when devising itinerary agents that travel around the network collecting summary or statistics.

5.1. Future Work

Work on making the mobile agents more ‘network aware’ is in progress, namely, better algorithms for the DMA’s to adapt to network variations like population variations (number of nodes to be monitored), spatial variations (stable differences in the quality of different links) and temporal variations (changes in the quality of a link over a period of time). The mobile agents could possess dynamic decision making ability based on various parameters whether to remain stationary, move through a set of machines sequentially or spawn out child agents. Dynamic placement decisions for population and temporal variations can be made based on the mobility policy presented by Brewington, Gray and Moizmi. [12].

To make mobile agents more appealing for network monitoring/management purposes the mobile agent systems should be more scalable, namely, the overhead of agent migration and inter-agent communication could be reduced. In the long run, agent-tracking, debugging and visualization issues could be tackled for improving scalability. Also different services like network-sensing modules would be needed to collect and analyze the host, repository and network conditions and make effective plans for accomplishing a desired operation. More effective planning algorithms using artificial intelligence techniques can be implemented into mobile agent based systems. Finally varied analysis should be carried out in order to identify tradeoffs between scalability, fault-tolerance, implementation complexity and functionality.

6. References

[1] A. Sahai, and C. Morin, “Towards Distributed and Dynamic Network Management,” *Proceedings of the 1998 IEEE Network Operations and Management Symposium*, vol. 2, pp. 455-464, New Orleans, USA, February 1998.

[2] A. Puliafito and O. Tomarchio, “Using Mobile Agents to Implement Flexible Network Management Strategies,” *Computer Communication Journal*, vol. 23(8), pp. 708-719, April 2000.

[3] M.K. Kona, and C-Z. Xu, “A Framework for Network Management Using Mobile Agents,” *Proceedings of the First IEEE Int'l Workshop on Internet Computing and E-Commerce*, San Francisco, USA, April, *2001*.

[4] D. Gavalas, D. Greenwood, M Ghanbari, M. O’Mabony, “Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology,” *Computer Communications*

Journal, vol. 23, no. 8, pp. 720-730, April 2000. http://snmp.cs.utwente.nl/bibliography/articles/general/gavalas-1.pdf

[5] H.M. Deitel, P.J. Deitel, “Java : How to Program, 4th Edition,” *Prentice Hall Publishers*, August 2001.

[6] T. Berners-Lee, R. Fielding, and H. Frystyk, “RFC 1945, Hypertext Transfer Protocol--HTTP/1.0”, May 1996.http://www1.ics.uci.edu/pub/ietf/http/rfc1945.html.

[7] “FAQ-Simple Network Management Protocol Part 1”, November 2001. http://www.snmp.com/FAQs/snmp-faq-part1.txt

[8] AdventNet, November 2001. http://www.adventnet.com/

[9] R. Siamwalla, R. Sharma, and S. Keshav, “Discovering Internet Topology,” Cornell University Report, November 1998. http://www.cs.cornell.edu/skeshav/papers/discovery.pdf

[10] M. Wiesman, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, “Understanding Replication in Databases and Distributed Systems,” *Proceedings of the International Conference on Distributed Computing Systems (ICDCS’2000)*, pp. 262-274, Taipei, Taiwan, April 2000.

[11] R. Vlach, “Mobile Agents and Databases,” *Proceedings of DATASEM'99*, Masarykova Univerzita, Brno, 1999. http://aglaja.ms.mff.cuni.cz/~vlach/

[12] B. Brewington, G. Gray, and K. Moizmi, “Mobile Agents in Distributed Information Retrieval,” *Intelligent Information Agents, M. Klusch Editor, Springer Verlag*, ch. 12. 1999.