Agent-Based Network Monitoring*

Alan Bivens, Li Gao, Mark F. Hulber and Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute, Troy, NY USA 12180-3590
Email: {bivenj, gaol, hulber, szymansk}@cs.rpi.edu

Abstract

As networks grow larger and more complicated, their management also becomes more difficult, especially when there is no central authority to coordinate management. In many cases, management efforts have imposed a great deal of traffic on the network, causing noticeable delays by network users. The realistic scalability of these systems is also a problem in most management applications. When the management tasks change, every node running the management application must be updated. Several research efforts are underway in the area of proactive network problem avoidance which often relies on a large amount of current and historical network data to build models of network traffic. This paper describes a distributed object, agent based framework for facilitating management of large networks as a foundation for supporting proactive network problem avoidance. The framework serves as a middleware between a collection of independent network management agents and network nodes. The presented work is done in collaboration with a network problem avoidance group at Rensselaer Polytechnic Institute, to provide current network performance data in a non-intrusive and reliable manner.

1 Introduction

In support of decentralized and proactive network management we are developing a Distributed Object-Oriented Repository System (DOORS). At the core, DOORS provide a secure, efficient, and fault tolerant method for the acquisition, management, manipulation, aggregation, and caching of network data and objects. We use mobile agents and a dynamic interface to support management and collection protocols such as LDAP, SNMP or NDS. Our goal is to deploy DOORS at a 100K node network owned by the Lucent Technologies that cannot be managed using traditional strictly hierarchical approaches. In the first phase of this project, described in this paper, we are primarily supporting the collection, aggregation, and management of SNMP data and related objects. The second step, however, is to combine our application with the network problem avoidance group to perform useful operations on the data collected to provide a proactive network management environment. In this environment, the intelligence of the agent will have a large impact. This and other advantages of the agent-based approach are described in subsequent sections of this paper.

This project has many diverse applications. Through DOORS, network performance visualization and other "upstream" application components can channel requests for objects and data. Requests outside the domain of a repository can be managed and forwarded to an appropriate cooperating repository. Regardless whether the request is within or outside the domain of a repository, the request is managed and responded

*This work was partially supported by DARPA grant F30602-97-C-0274 and IBM Fellowship. The content does not necessarily reflect the position or policy of the U.S. Government or IBM Corp.

Experiment Time	Interval	Collection Time	CPU Utilization	Seconds per Request
7200s	1s	111s	.0154	.0154
	10s	31s	.0043	.0430
	15s	34s	.0047	.0708
	30s	18s	.0025	.0750
	60s	9s	.0012	.0750

Figure 1: Impact of SNMP Data Polling on Network Router

to in a manner virtually transparent to the requester. Through several mechanisms, the DOORS take steps to meet the temporal requirements of the requester. If recent data is available in the DOORS, the object and its data are immediately provided to the requester. Otherwise, the DOORS, having been configured with the tools to collect this type of object, either communicate with an existing agent or launch an appropriate agent in order to provide the object to the requester. In anticipation of frequent requests from one or multiple sources, the DOORS may proactively collect network data in order to meet requests in an efficient manner.

This paper focuses on the first phase of the development of our DOORS infrastructure, emphasizing the following components:

- Data Management Requirements
- DOORS Architecture
- Implementation Details
- Preliminary Timing Results

2 Data Management Requirements

The currently experienced growth of networks has been accompanied by the proliferation of support at various levels to manage networks and their activity. Directory services such as SNMP and, more recently, NDS have been developed, used, and grown over time to collect and disseminate information about networks. As the scale of the network management effort grows, the collection processes and management efforts themselves can put a significant load on the networks being managed.

Our goal is to minimize the negative effects of obtaining network data, while offering more functionality than provided by the existing systems. Figure 1 shows the impact of polling SNMP data from a router. We intend to keep this impact minimal by channeling access for the same data through repositories and, when applicable, preprocessing some of the data, thereby decreasing the total network traffic created. Figure 1 confirms that even for polling intervals as short as 1 second, the CPU utilization is less than 2%.

Network management places complex requirements on the physical location of the network data. Three major factors are performance, availability, and bandwidth usage. Performance, in this case, has to do with client queries and agent updates. For the clients and agents, their repository must be "nearby" in the sense of the physical layout of the network. This would imply the repository resides on the same subnet or is at most a few hops away from routers that are assigned to it. However, it's unrealistic to expect a repository to reside on every subnet. One repository per some small set of physically close subnets should be sufficient. For performance reasons, the optimal location of the repository would be the network region for which it is holding data. However, data for a network region should be available during periods when that region is unreachable. Therefore, the repository should be somewhere nearby without actually residing in the region.

Another important issue is bandwidth usage. One of the main goals of this project is to provide its services with an absolute minimum impact on the network. It is likely that clients will frequently request data for networks outside the domain of its local repository. In this case, the client sends the request to its local repository and it is the repository's responsibility to retrieve the data from the nonlocal repositories. Section 4.2 describes how this situation is handled.

The repository itself assists in keeping the impact on the network minimal by the functionality it passes on to the client through the requesting language. The repository can handle cases such as a client wanting to request a certain SNMP variable every 4 seconds for an hour, in one command. The repository's requesting language allows the client to request any number of variables from a router at any interval for any time period in one request. This allows the system to use a push method of requesting data. If a standard network management protocol were used for the previous example, the client would have to make a separate request every four seconds for an hour. This type of request could noticeably impact network performance making the results obtained for large networks useless. However, under the DOORS the client makes one request in response to which an agent is dispatched to a polling station, usually close to the requested machine (router). The dispatched agent will query the machine (router) every 4 seconds for an hour. This can make a very big difference in bandwidth used by the network management system. Many times applications such as network problem avoidance need large amounts of current data. For such application, minimizing the data collection effect on the network is of utmost importance.

3 DOORS Architecture

DOORS use several components to retrieve and process network data. A simple graphical representation of these components working together for a local data request (involving only one repository) is given in Figure 2. The purpose of each component will be given here, whereas the specifics, including what is actually used, will be described in the next section.

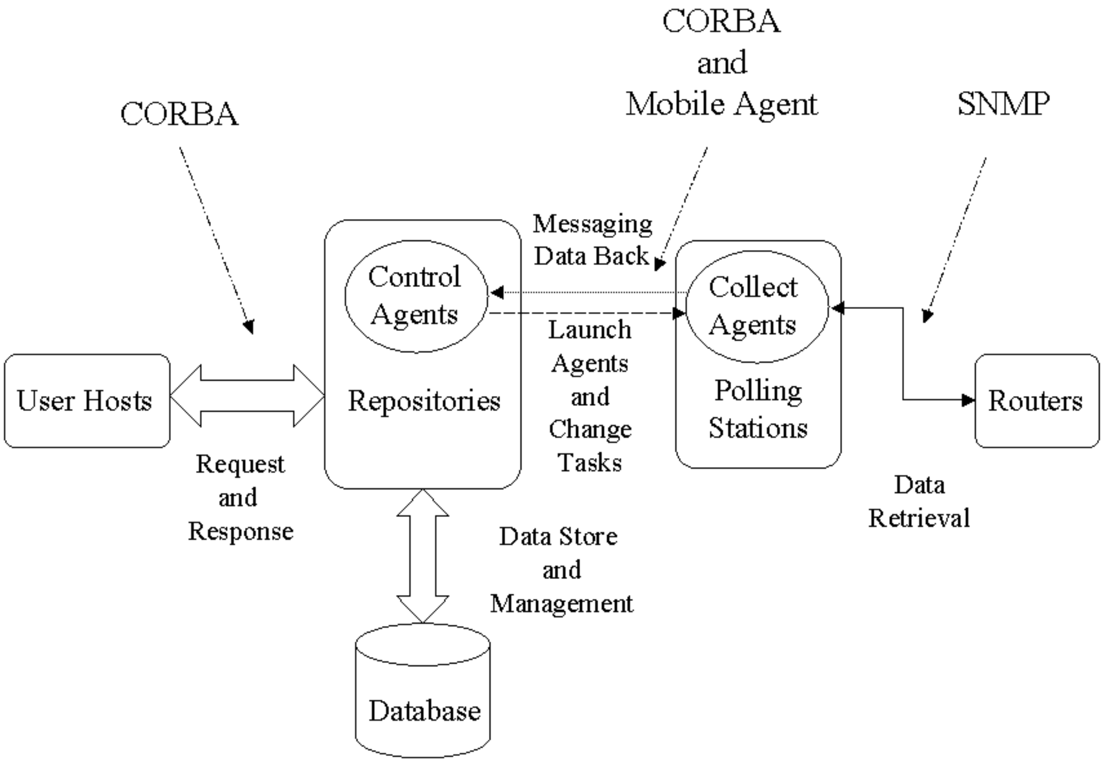

Figure 2: DOORS Architecture

3.1 Client Interface

The Client interface is simply a device used to communicate with the repository. It was designed to be simple, so it could be both versatile, and easy to develop. It could be a small stand-alone application or part of a larger application which could involve visualization, network problem avoidance, or other "upstream" application components. It only needs to formulate and send a request in a form recognizable to the repository. Pictured as the "User Hosts" box in Figure 2, the client communicates with only one local repository. In the current implementation, there is a GUI interface that enables the user to define the SNMP data that needs to be collected, together with the collection period and interval.

3.2 Repository

The actual repository is responsible for the controlling of agents and coordinating requests from different clients as well as other repositories. Upon receiving a request, the repository will determine if a recent copy of the data requested exists in the storage system. If a recent copy is present, a request will not need to be issued to the router, reducing network traffic, turnaround time, and router load. Otherwise, of course, the request to the router is made.

The repository is not only a reactive entity. If patterns are detected of certain data being requested every day at a certain time, for example, the repository can retrieve that data in anticipation of the request coming. This is an additional way in which the system decreases the response time of the system.

3.3 Mobile Agents

The mobile agents are sent from the repository to a polling station to request the actual data from the destination object (typically a router).

Mobile agents give us many benefits in this implementation and offer many more benefits for future versions of this system. They help to reduce the network load by passing back only the data necessary for the client and the repository. This is especially useful when additional processing functionality is added to the agent. In this case the agent will return the result of some computation or manipulation of data, utilizing the paradigm of "taking the computations to the data, rather than the data to the computations." An example of such processing is stripping off the string identification produced by SNMP and reformatting numerical data into a binary form to save the amount of bytes that must be sent back from the router to the repository.

Mobile agents also give us a great framework for updating and adding new functionality. The agent encapsulates protocols and procedures, so when a change needs to be made or functionality needs to be added, only the agent needs to be changed. Under any other implementation, each client and request receiving agent would have to be updated individually and consistently. If the network management system is running on a very large network, this task becomes very expensive.

The agents execute asynchronously and autonomously, so once an agent has been assigned to a job, the user is free to process other tasks. In addition, the agents are naturally heterogeneous. They are both computer and transport-layer independent because they depend only on their executing environment.

Perhaps the most important advantage of mobile agents is their robustness. Mobile agents can be made to react dynamically to different conditions, which makes it easier to build a robust, fault-tolerant system. For example, if a host is shut down, all agents working on that machine will be given a warning and time to dispatch. This way the agent could collect the data from another machine or return to the issuer of the agent to explain exactly what happened. If conventional methods were used to collect data, either garbage would be sent back, or an unpredictable error message would be produced.

3.4 Polling Station

Polling station is a critical component, because it would be difficult to send an agent to sit on the router itself to request data. Many routers use a custom operating system and heterogeneity of platforms make the code maintenance for the data collection difficult. In addition, running the collection programs directly on the routers could introduce unacceptable load on the routers. To avoid such an overload and for security reasons, a group of routers may have a *polling station* allocated to them. This station is a designated SNMP server for them. Therefore, our data collecting agent travels to the polling station and from there collects the data from the router with minimum network traffic. The polling station needs to run an agent daemon which can receive the agents and allow them to execute their tasks. Most agent daemons, including the Aglets system that we use in DOORS, are lightweight applications, so they do not take much CPU time or memory.

3.5 Storage System

A sophisticated storage system is not necessary for a basic functionality of DOORS system. In the prototype discussed here, there is no storage system, so every request for data is sent to the router. However, the future versions of DOORS will use a lightweight database as a storage system to hold historical and current data retrieved with the agents, as well as meta-data used to configure parts of the system. In addition to the storage system, the repository will maintain a cache of the most recent data retrieved for quick matches with client requests.

4 Implementation Details

Building our prototype, we made several decisions regarding the tools used. There are three major components we built upon:

- Object model and primary development language
- Method for data management
- Infrastructure for data collection

Our primary concerns in selecting these tools were portability and extensibility. We have chosen to use the CORBA object model as the primary vehicle for communication and management of distributed network objects that provide us with a real and usable infrastructure. Currently we are working with Iona's version of CORBA called Orbix and its counterpart OrbixWeb.

Portability of our code together with our selection of OrbixWeb and Aglets agents made a choice of Java as our primary development language a natural one. We recognize that Java programs may not be as efficient code written in languages such as C++. However, Java is beneficial in terms of portability and wide-spread use by implementors of agent systems.

4.1 Physical Layout of Repositories

We have considered three models for organizing repositories through the network: hierarchical, network, and peer models. In the hierarchical model there are two types of repositories: foreman and worker. The worker repository is responsible for information about its own domain. It communicates only with its foreman repository. A foreman supervises one or more workers and is responsible for managing all incoming requests for all the domains to which its workers are assigned. The worker would usually handle a small domain, such as a small number of close subnets, whereas the foreman would be responsible for all the workers in a larger domain, such as a building or remote office. This model lends itself easily to hierarchical aggregation of data at the foreman repositories.

In the network model, the repositories transfer data in a fashion similar to network routers. Queries between repositories go through neighbor repositories. The information flow between two distant repositories can be cached at any intermediate repository along its path. Queries for the same data could then be answered by this intermediate repository, thus providing efficient use of network bandwidth. However, meaningful aggregation of data would be difficult.

In the peer model, queries between repositories are made directly with the end node repositories. Unlike the hierarchical and network model, there are no intermediate repositories. Each repository is directly responsible for handling all queries for information within its own domain. This model is the easiest to implement, and, therefore, our first prototype is done this way. However, network usage is highest and no intermediate caching can be done.

4.2 Repository-to-Repository Communication

As described in the introduction, clients request data from their local repository. However, the information requested may reside within the domain of some other repository. Therefore, repository-to-repository communication is necessary for retrieval of nonlocal data.

From the information model standpoint, the repository-to-repository communication is forwarding client requests between repositories and receiving the data from the source repository to send back to the client. When a repository is caching data from other repositories for its clients, it should use the same consistency model used for clients requesting data from repositories. Different types of consistency models are outlined in [6]. For example, if the client requests a temporal update of data (client should receive updates every x units of time), then the repository should request the same type of updates from the repository sending the data.

The consistency issues become complicated when there are multiple clients requesting the same data with different consistency requests on the data. In this case, the repository should intelligently combine the consistency models.

Hierarchical aggregation of data is also useful. A foreman repository can aggregate information about its worker domains. For example, if the network traffic is slow in all worker domains, the foreman may aggregate this information to indicate that the traffic is slow for the entire domain.

5 Preliminary Timing Results

As previously stated, we retrieve SNMP data from the routers that are targeted in client queries. SNMP could be used to retrieve data from a distance instead of a more complex system such as the one that we have described in this paper. However, a slightly more complex system can provide fault tolerance, ease of data management, and ability to aggregate data and requests. In this section, we will compare the timing of different components of our system to evaluate the cost of the added advantages.

Tests were run collecting SNMP data directly from a router that was two hops away from the clients using a Java SNMP package and different numbers of clients, each requesting data with certain time intervals. The same tests were then run using a subset of DOORS that included Aglets and the SNMP package, and finally the tests were run using a DOORS prototype that included Orbix, Aglets, and the SNMP package. The router in our tests was a SUN Ultra2 workstation *(coffeepot.cs.rpi.edu).* The polling station and the repository were one SUN Ultra1 machine *(eggbeater.cs.rpi.edu).* Coffeepot and Eggbeater are in the same subnet, i.e. zero hops away from each other. The five user workstations, a SUN Ultra10, a SUN Sparc10, and three SUN Sparc5's, were two hops away from the router. The average times for servicing client requests at specified intervals over the total test time of 1000 seconds were collected, together with the variance, and percentage of data requests honored.

We wanted to determine exactly how much time the added DOORS functionality incurred. To this end, we forced our system to emulate the way a client with nothing but the SNMP protocol would request and receive data. Therefore, each client requests data at each interval to receive the new piece of data. However, typical use of DOORS is to have the client request the data once, and have the repository send the data back to the client at the given frequency. The test results of the system emulating the simple SNMP system are shown below with a short description of the test details.

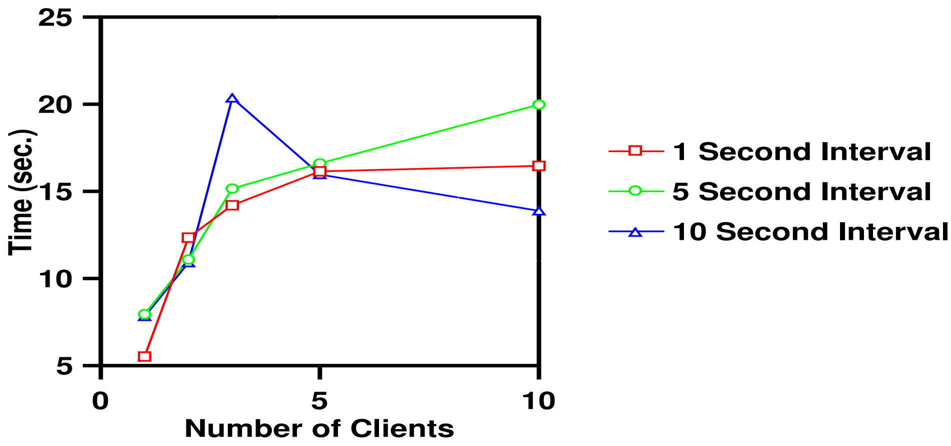

Figure 3: Timings using only SNMP

Figure 3 shows the test results from running only SNMP in a simple client application. This case has the lightest overhead and therefore offers no added advantage. Each client simply polls the router over a period of 1000 seconds with the given interval. In large networks, many of these requests may timeout if the load on a polled workstation or the corresponding network path is heavy.

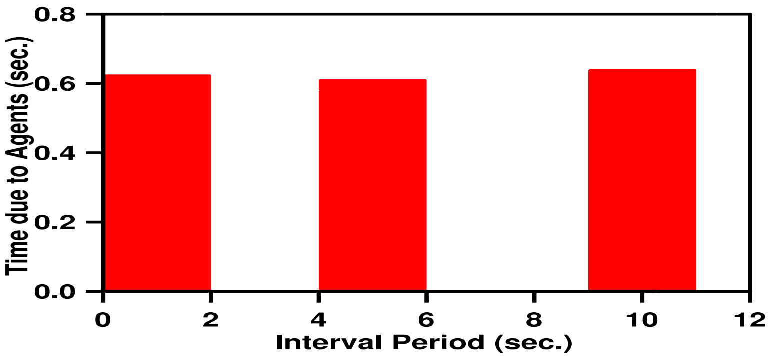

Figure 4: Timings using Aglets and SNMP with 1 client

The graph in Figure 4 summarizes the test run using the Aglets agents and SNMP package. In this case an agent is sent from the repository to the polling station where the agent sits and sends repeated requests to the designated router. This test implementation does not support several clients using agent technology, therefore, only data for one client is provided. This experiment was done to determine what role the use of agents played in any added cost. Only additional time is plotted, so a quick comparison with the data in Figure 3 indicates that this overhead is about 10% for the single client.

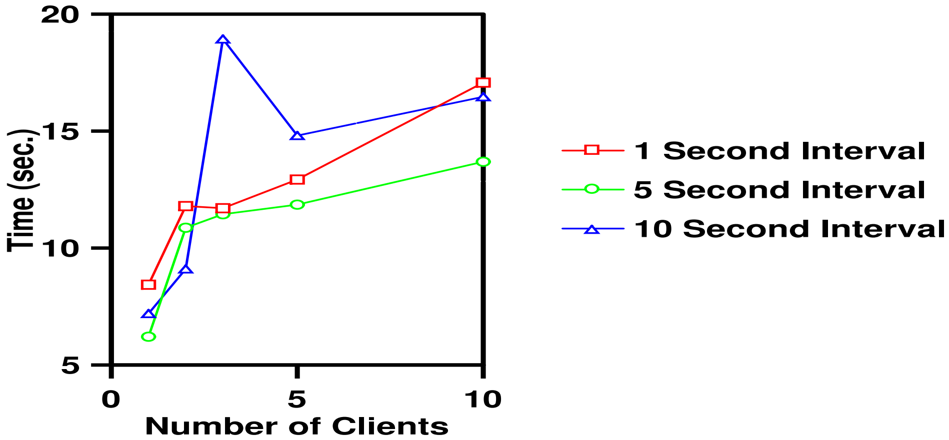

Figure 5: Timings using OrbixWeb, Aglets, and SNMP with Agent Interval of 1 second

The plot in Figure 5 presents the test run using the prototype DOORS which included OrbixWeb, Aglets, and SNMP. Each client is an OrbixWeb client linking to an OrbixWeb server (repository) containing an Aglets daemon. The requests from the clients to the repository are at the intervals noted in the figure legend. When the communication of the client to the repository takes place, the Aglets daemon sends an agent to the polling station. Once the agent gets to the polling station, it polls the router at an interval of one second. Our implementation uses a pull technology between the clients and the Aglets daemon. When the client requests data, it first lets the Aglets daemon know the requests, and then asks for the information at the intervals shown in the figure legend. This allows the client to specify how old the data may be.

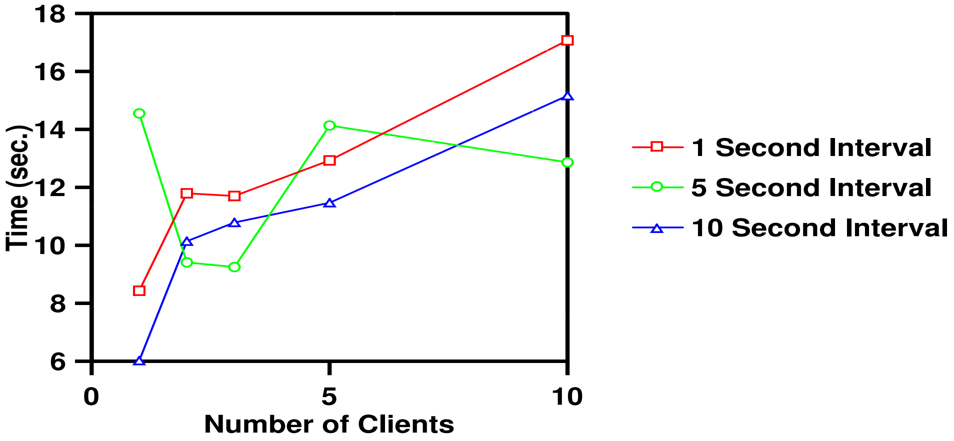

Figure 6: Timings using OrbixWeb, Aglets, and SNMP with Agent Interval equal to the client request interval

The graph in Figure 6 details a test run exactly like the one presented in Figure 5, except of the frequency of data collection. In this test, when an agent got to the polling station, it polled the router with the interval specified by the client.

5.1 Analysis of Tests

To enable an easy comparison of performance of different methods, the plots in Figures 7 and 8 show the timings of each method with 5 sec. and 10 sec. intervals, respectively. Several conclusions can be drawn from these data.

- As the number of clients increase, the average time needed to retrieve data also increases. However, the increase in time is not linear, it has a diminishing growth factor. This means that we are not paying too much for added clients.
- The cost of using Aglets agents is very low thanks to the low cost of running Aglets deamon. An agent consumes polling station cycles only if the data that it collects were requested by the user.

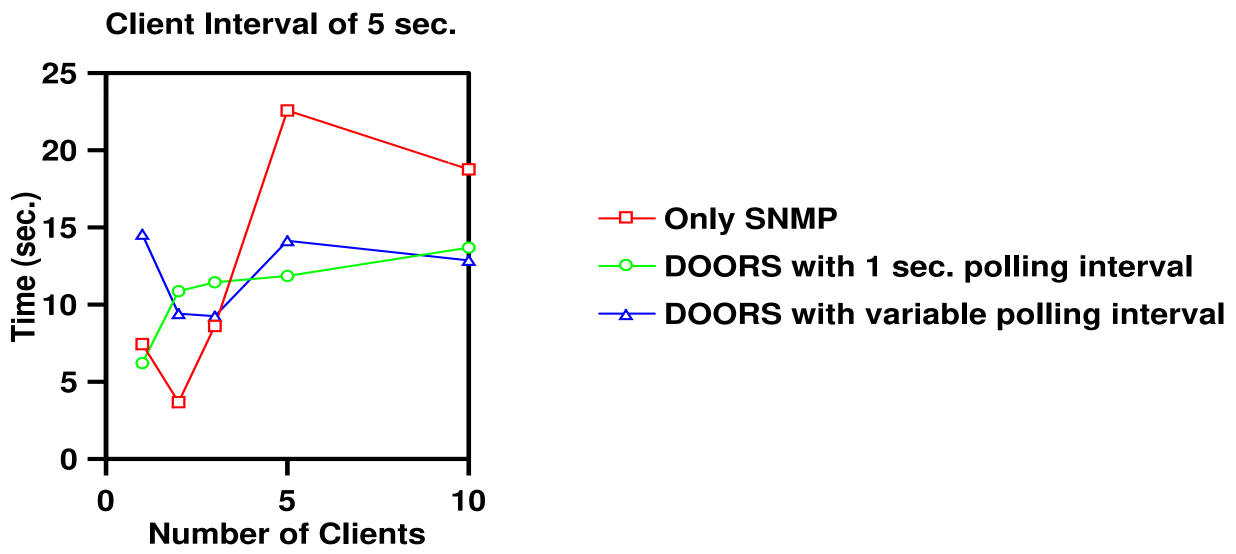

Figure 7: Comparison between the DOORS system and the direct SNMP method running with an interval of 5 sec.

- While running the tests, we noticed that sometimes the direct SNMP method did not return a value for some requests. This usually happened when more than two clients were active. We believe that this failure was caused by heavy network traffic created by the several clients' continuous polling. In our application, the only time this may happen is when the user attempts to poll values with a strict time constraint. For example, if the user needs values that are less than 3 seconds apart, but the repository has a 5 second interval values, the frequent data collection may constraint the router.

- Most importantly, as seen in Figures 7 and 8, the performance of DOORS either outperforms or is comparable in performance to the direct use of SNMP. At the same time, DOORS, compared to direct SNMP polling, reduce network traffic, decrease the load on routers and provides improved accessibility and fault tolerance.

It should be noted that the tests were run on the network in the Computer Science Laboratory at Rensselaer Polytechnic Institute. Therefore, all network locations were relatively close. We believe that deploying DOORS on a large networks will result in further improvement of performance compared to the direct use of SNMP.

6 Related Work

Bauer et al. use a repository for management of distributed applications in the MANDAS (Management of Distributed Applications and Systems) project [2]. The authors concentrate on an area other than network

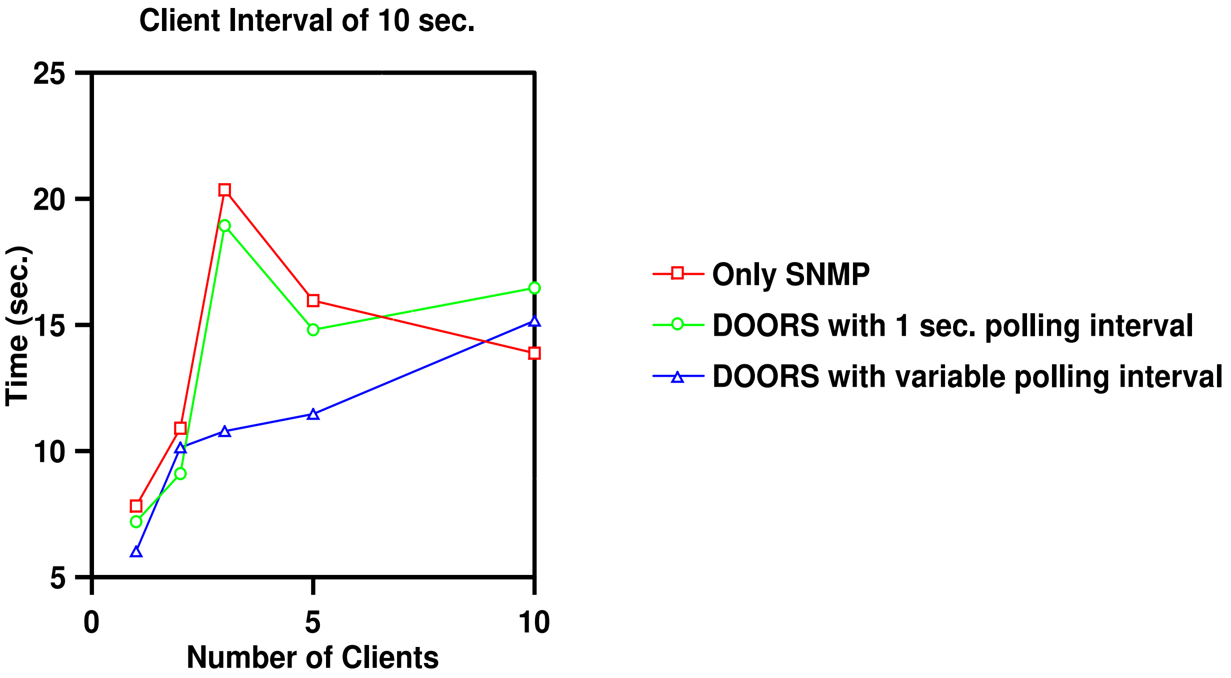

Figure 8: Comparison between the DOORS system and the direct SNMP method running with an interval of 10sec.

management, but there are many similarities between their work and ours. They use a Management Information Repository (MIR) to hold information about their distributed applications. However, they have only one centralized repository. Our implementation will use distributed repositories with advanced communication methods for transferring data between the repositories.

Harista et al. describe MANDATE (MAnaging Networks Using DAtabase TEchnology), which is a proposed MIB (Management Information Base) for network management [3]. The authors propose data operators that interact solely with their MIB which holds information about the network. MIB's are similar to our network of repositories. Implementation of MANDATE is client-server based with sophisticated client caching. Our implementation is based on distributed databases with caching between repositories for performance and availability.

7 Summary

Large networks consisting of thousands of nodes cannot be managed using the strict hierarchical approaches of the past. In this paper we discuss proactive network management and the role played by our Distributed On-line Object Repository System (DOORS) in the integration and support of the collection of the network management data. We outline the requirements which the network management projects place on DOORS and discuss its current implementation. Our next steps include adding a lightweight database in the repository and utilizing larger networks for testing. We will also look at other data collecting methods in addition to SNMP to retrieve different types of data. Our hope is to continue improving our prototype system for this network management project and extend DOORS to support other applications requiring distributed databases. As network problem avoidance techniques continue to grow and the size and complexity of networks continue to get bigger, having a non-intrusive network management utility like DOORS will become more and more valuable.

Through our experiments we have demonstrated that we are able to add functionality and some transparency to the collection of network performance data with reasonable overhead introduced. We have demonstrated the use of mobile agents in our implementation. The use of these agents has provided for increased flexibility and adaptability of our system.

References

[1] M. Baldi, S. Gai, and G. P. Picco. Exploiting code mobility in decentralized and flexible network management. In *Mobile Agents*, pages 13–26, Berlin, Germany, April 7-8 1997. First International Workshop, MA, Springer Verlag.

[2] M. A. Bauer, R. B. Bunt, A. El Rayess, P. J. Finnigan, T. Kunz, H. L. Lutfiyya, A. D. Marshall, P. Martin, G. M. Oster, W. Powley, J. Rolia, D. Taylor, and M. Woodside. Services supporting management of distributed applications and systems. *IBM Systems Journal*, 36(4):508–526, 1997.

[3] J. R. Harista, M. O. Ball, N. Roussopoulos, A. Datta, and J. S. Baras. MANDATE: MAnaging Networks using DAtabase TEchnology. *IEEE Journal on Selected Areas in Communications*, 11(9):1360–1372, December 1993.

[4] Danny B. Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets.* Addison-Wesley, Massachusetts.

[5] G. Mansfield, E. P. Duarte Jr., M. Kitahashi, and S. Noguchi. Vines: Distributed algorithms for a web-based distributed network management system. In *Worldwide Computing and Its Applications*, pages 281–293, Tsukuba, Japan, March 10-11 1997. International Conference, WWCA, Springer Verlag.

[6] S. Parthasarathy and S. Dwarkadas. InterAct: Virtual sharing for interactive client-server applications. Presented at 4th Workshop on Languages, Compilers and Run-Time Environments, CMU, Pittsburgh, May 1998.