COMPLIMENTARY POLLING MODES FOR NETWORK PERFORMANCE MANAGEMENT EMPLOYING MOBILE AGENTS

Damianos Gavalas[†], Dominic Greenwood[*], Mohammed Ghanbari[†], Mike O'Mahony[†]

[†]Electronic Systems Engineering Department,
University of Essex, Colchester, CO4 3SQ, U.K.
E-mail: {dgaval, ghan, mikej}@essex.ac.uk

[*]Fujitsu Telecommunications Europe Ltd.,
Northgate House, St. Peters Street, CO1 1HH, Colchester, U.K.
E-mail: D.Greenwood@ftel.co.uk

Abstract

Several distributed Network Management (NM) architectures, exploiting the advantages of Mobile Agents (MA), have been recently proposed to answer some of the limitations intrinsic to client-server based centralised NM, such as information bottlenecks and lack of flexibility. However, when considering network performance management, they fail to address scalability problems. In this paper, we introduce two efficient, lightweight polling modes based on MAs that address these limitations. Both real-time and off-line NM data acquisition is considered. The introduced modes are shown to outperform SNMP-based polling both in terms of response time and bandwidth consumption.

1. Introduction

Network Management Systems (NMS) in service today are typically based on a centralised model, which involves a management application (manager) and several managed entities (agents) embedded within Network Elements (NE). Protocols, such as SNMP, are used for interaction between managing and managed parties. Several drawbacks are apparent with this approach: due to rigid design time definitions, NMS functionality cannot be dynamically updated, whilst frequent polling is known to result in substantial data transmission rates and processing bottlenecks.

All these problems have triggered an evolution towards distributed NM, hence leading to several Mobile Agent Frameworks (MAF) reported in [1][2][3]. MAs provide a powerful software interaction paradigm that allows code migration between hosts for remote execution. The data throughput problem can be addressed by delegation of authority from managers to MAs, where these agents are able to filter and process data locally without the need for transmission to a central manager. However, neither [1] nor [2] consider remote processing issues and, as a result, the manager host still suffers from a computational burden. These issues are addressed in [3].

All the aforementioned works involve frequent MA transfers, when the collection of management statistics is considered. In addition to the apparent network overhead, these frameworks are not scalable as they assume a 'flat' network architecture, i.e. a single MA is launched from the manager platform and sequentially visits all the managed NEs, regardless from the underlying topology. Thus, for large networks the round-trip delay for the MA will greatly increase, whilst the extracted statistics will not be accurate and reliable due to the non-negligible time intervals between the acquisition of each data sample for every NE. For such reasons, these MAFs are not appropriate for Network Performance Management (NPM), which represents the main focus of this paper.

NPM involves gathering and logging of data, which may be analysed off-line or in real-time. That process helps in determining the performance, throughput and availability of network resources. The advantage of analysing the data in real-time is that it allows sophisticated NMSs to foresee possible congestions or failures and take preventive measures before the actual error occurs. On the other hand, collected data may be used to build daily, weekly or monthly reports to assist the administrator in network planning. In such cases there is no need for real-time NM data and, hence, an alternative and complimentary polling mechanism should be applied.

Therefore, we propose two MA-based polling modes intended to provide an efficient method for obtaining both real-time and off-line management data. In the first approach, called *Get 'n' Go* (GnG), used to collect real-time data, the network is partitioned into several domains and a single MA object is assigned to each of them. In every Polling Interval (PI), this MA sequentially visits all NEs within the network domain and obtains the requested information before returning to the manager. The second polling scheme, called *Go 'n' Stay* (GnS), targets the acquisition of data to be analysed off-line, where the need to obtain data in short time frames is no longer an imperative. Thus, we introduce a method where an MA object is broadcasted to all managed devices; the MA remains there for a number of PIs and collects an equal number of samples before returning to the manager. The infrastructure described in [3] has been extended in order to support the deployment of the introduced polling modes.

The paper is organised as follows: Section 2 provides a brief description of the MAF used to support this work, while Section 3 describes in detail the two introduced polling modes. A performance analysis is given in Section 4, with experimental results reported and discussed in Section 5. Section 6 concludes the paper.

2. Mobile Agent Network Management Framework

The MA-based NM framework has been entirely developed in Java [4] as it offers the platform independence required for the management of distributed heterogeneous environments. This and other features, such the rich class hierarchy for communication in TCP/IP networks and its already wide acceptance for the development of distributed applications, positions Java as an ideal platform for MA-oriented management services.

Our infrastructure consists of four major components [3], illustrated in Figure 1:

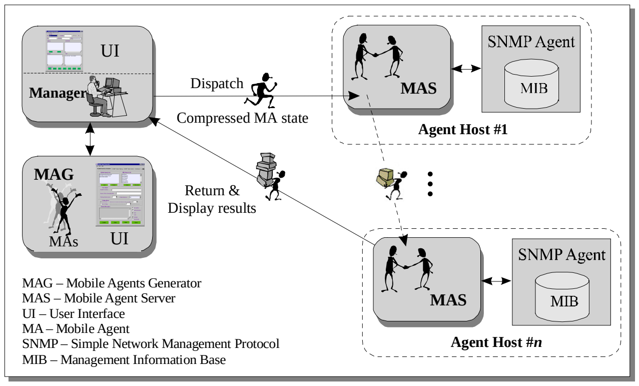


Figure 1: The Mobile Agents-based Infrastructure

Manager Application: The manager application, equipped with a browser style User Interface (UI), co-ordinates monitoring and control policies relating to the NEs. Active agent processes are *discovered* by the manager, which maintains and dynamically updates a 'discovered list'.

Mobile Agent Server (MAS): The interface between visiting MAs and legacy management systems is achieved through MAS modules, installed on every managed device. The MAS resides logically above the SNMP agent, creating an efficient run-time environment for receiving, instantiating, executing, and dispatching MA objects. It also provides requested management information to active MAs and protects the host system against external attack. The MAS composes four primary components (see Figure 2):

- Mobile Agent Listener,
- Security Component,
- Service Facility Component,
- Migration Facility Component,

while Network Discovery Daemon and Class Loader Daemon exist outside the boundary of the MAS.

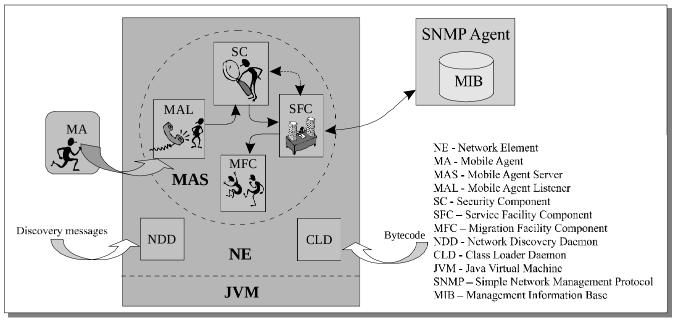

SNMP Agent
MIB
SC
MAL
SFC
MA
MFC
MAS
Discovery messages
NDD
NE
CLD
Bytecode
JVM
NE - Network Element
MA - Mobile Agent
MAS - Mobile Agent Server
MAL - Mobile Agent Listener
SC - Security Component
SFC – Service Facility Component
MFC – Migration Facility Component
NDD - Network Discovery Daemon
CLD - Class Loader Daemon
JVM - Java Virtual Machine
SNMP – Simple Network Management Protocol
MIB – Management Information Base

Figure 2: The Mobile Agent Server structure

Mobile Agent Generator (MAG): The MAG is essentially a factory for constructing customised MAs in response to service requirements. Such MAs may dynamically extend NMS functionality, post MAS initialisation, to accomplish management tasks tailored to the needs of a changing network environment. The MAG's operation is described in detail in [3]. Its functionality has been extended though, so as to allow the operator (through a dedicated UI) to specify:

- Whether GnG or GnS polling will be used;
- The polling frequency (i.e. the PI's duration);
- Whether or not to encrypt collected data.

It should be emphasised that the transfer of the MA bytecode is performed only once, through broadcasting it to all active MASs at MA construction time. Thereafter, the transfer of persistent state, obtained from *serialising* [4] the MA instance, is sufficient for the MAS entities to recognise the incoming MA and recover its state. In contrast, in [1] and [2] the transfer of both the MA's bytecode and persistent state is required, resulting in a much higher demand on network resources, as bytecode size is typically much larger than state size.

Pre-defined MA property sequences are stored in configuration files and parsed at manager initialisation to instantiate the corresponding MA objects. These properties may be modified at runtime.

Mobile Agent: From our perspective, MAs are Java objects with a unique ID, capable of migrating between hosts where they execute as separate threads and perform their specific management tasks. MAs are supplied with an itinerary table, a data folder where collected management information is stored and several methods to control interaction with polled devices. The MA's state information is compressed (using the Java *gzip* utility) before being transferred to the next destination host.

3. Polling Modes

3.1. Get 'n' Go Polling Mode

Traditional SNMP-based polling involves a flood of request/response messages as shown in Figure 3(a). This naturally leads to a significant proportion of available bandwidth being used for management data.

On the other hand, recently reported NM MAFs [1][2][3] assume a flat network structure, hence as the number of managed devices grows, the network becomes increasingly unmanageable. This is a consequence of having a single MA responsible for obtaining NM data from each device in every PI (see Figure 3(b)), causing serious scalability problems. In addition, the order in which managed devices are visited is arbitrary. This represents a significant problem when the management of remote LANs is considered, as a travelling MA may have to be transferred several times across expensive and low-bandwidth WAN links during its lifetime.

As previously mentioned, the concept behind GnG polling is to partition the managed network into several logical/physical domains. The partitioning criteria are specified by the administrator and may correspond to: (i) the number of nodes assigned to each MA, (ii) the physical distribution of polled devices, or (iii) a hybrid of these two approaches. The number of MAs required per PI is automatically evaluated, and their individual itineraries instantly specified. For instance, in Figure 3(c), an MA object polls the devices of the remote LAN, whereas a second MA is assigned to the network segment local to the manager host. With the GnG approach, MAs are required to visit a limited number of devices and, as a result, the overall response time is minimised. This factor suggests GnG as a suitable polling scheme for the acquisition of real-time data.

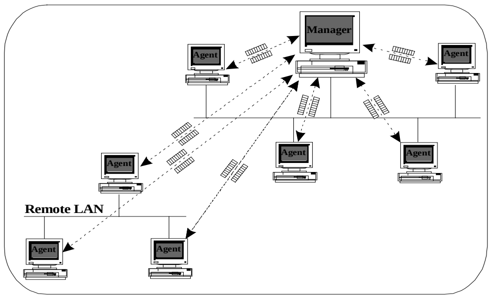
In terms of implementation, GnG polling is carried out through *Polling Threads* (PT). PTs are started and controlled by the manager application; each of them corresponds to a single polling instance. When started, PTs retrieve polling definitions and schedules from their associated configuration files and poll NEs on a regular basis. Specifically, PTs instantiate and launch the required number of MAs (supplied with their corresponding itinerary) and then 'sleep' for one PI. When this period elapses the same process is repeated. Meanwhile, a manager's listener daemon receives the MAs that return to the manager carrying their collected data.

PTs may be synchronised in such way that they are not initiated simultaneously. This ensures that the traffic around the manager host will be distributed over time.

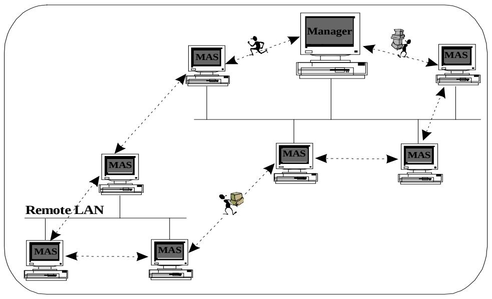
3.2. Go 'n' Stay Polling Mode

GnS polling introduces an alternative approach to NPM, targeting data intended for off-line analysis. This reduces the number of MA transfers whilst the volume of data carried by each MA increases. Hence, the proportion of *useful* management information within the MAs state is substantially increased, compared to the other approaches where a large number of MA transfers may be required to obtain a few data samples.

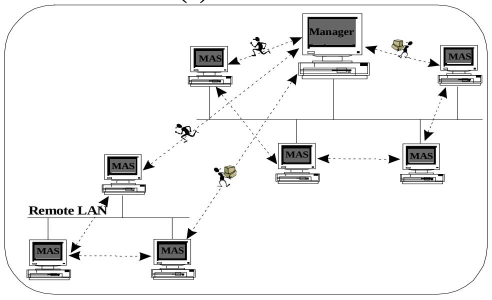
Specifically, every PT broadcasts at regular intervals an MA object to all agent hosts. The MAs then remain active on the hosts for *p* PIs (where *p* is specified by the administrator). At the end of each PI they obtain a sample of the requested data set and encapsulate it into their state. The MAs then sleep for one PI and then awaken to obtain



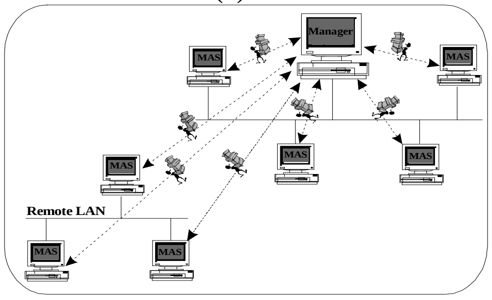
(a) Centralised



(b) Flat



(c) Get 'n' Go



(d) Go 'n' Stay

Figure 3: Approaches to polling.

another sample. When the p PIs elapse, the MAs return to the manager to deliver the acquired samples (see Figure 3(d)). Meanwhile, PTs suspend execution for a duration given by the product of PI and the number p of PIs that MAs remain on the managed devices ($PI \times p$). When this period expires, they resume operation and the process is repeated.

Clearly, there is a trade-off between bandwidth consumption and response time. As p increases, so does the response time to the manager. However, as the MA transfers become sparser the network overhead imposed by polling reduces. If, for instance, p=100, MA objects will be broadcasted every 100 PIs. When changing to p=50, the MA transfers are doubled but the response time is halved. In the extreme case that p=1, the response time is minimised and GnS mode becomes similar to SNMP-based polling and identical to GnG, when each MA is assigned to a single device. The administrator is given the option to modify the value of p and also dynamically change the polling mode from GnS to GnG and vice versa, depending on the managed network traffic conditions and the necessity for obtaining certain types of management information. When changing from GnS to GnG, automatic network domain segregation takes place.

4. Performance Analysis

The performance of our polling modes is tested using a function, known as a *health function* (HF) [5] that aggregates multiple Management Information Base (MIB) variables into a more meaningful indicator of system state. For example, the percentage *E(t)* of IP output datagrams discarded over the total number of datagrams sent during a specific time interval, is a function of *five* MIB-II objects:

$$E(t)=\frac{(ipOutDiscards+ipOutNoRoutes+ipFragFails)*100}{ipOutRequests+ipForwDatagrams} \quad (1)$$

Thus, when centralised polling is employed, the value of *E(t)* is computed by the manager when all the object values appearing in Eqn. (1) have been returned by the static agents, in separate response packets. Hence, if $S_{req/res}$ is the average request/response size, and in the general case that polling of n devices for v operational variables is applied, the wasted bandwidth for i PIs would be:

$$B_{centr} = 2*S_{req/res}*n*v*i \quad (2)$$

If on the other hand GnG or GnS polling is used, the HF calculation is performed at the local agent host, leading to more balanced processing distribution. Also, considerable compression of data is achieved since a large number of operational variables are reduced to a single value. Thus, assuming an MA average compressed code size of S_c and compressed state information size S_s, the resulting overhead for GnG polling (with d network domains) would be,

$$B_{GnG} = n*S_C + d*S_S*(\left\lceil \frac{n}{d} \right\rceil + 1)*i,\ d \le n \quad (3)$$

as the v variables are aggregated into one. The first term of the equation describes the overhead imposed when broadcasting the MA code to all MASs, while the second represents the bandwidth consumed by the MA state transfers between the manager and the polled devices (each MA is assigned to $\left\lceil \frac{n}{d} \right\rceil$ NEs). Thus, for large v and i, GnG mode is less bandwidth intensive than centralised polling.

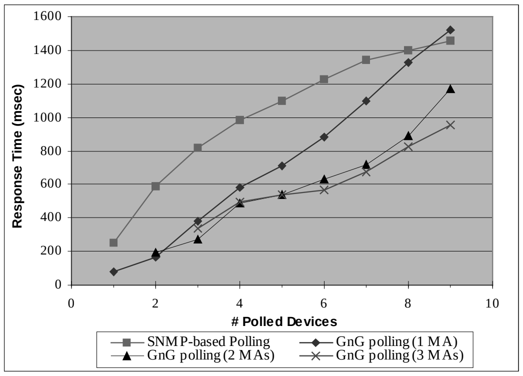
Similarly, for GnS polling,

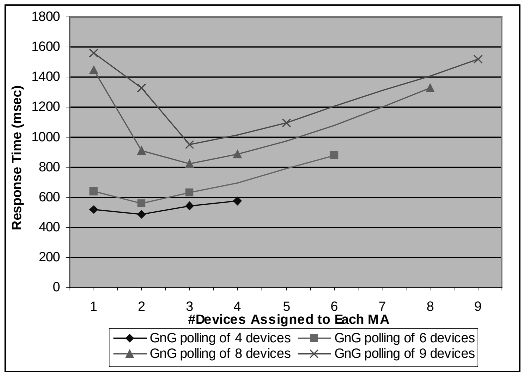
$$B_{GnS} = n*S_c + (2*n*S_S)*\left\lceil \frac{i}{p} \right\rceil \quad (4)$$

as MAs remain on the managed devices for p PIs. For a large p, GnS mode becomes the most lightweight polling approach in terms of bandwidth consumption.

5. Experimental Results

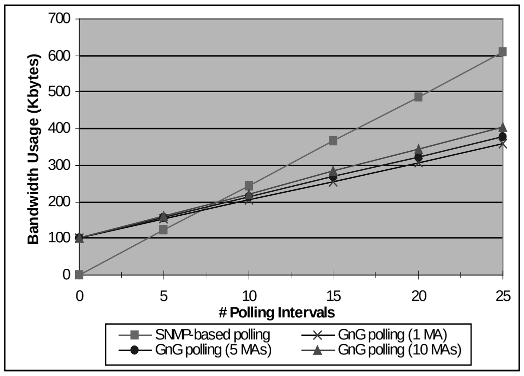
The experimental testbed comprises a network of several Solaris and WinNT machines. We assume the physical distribution of managed devices in the network as



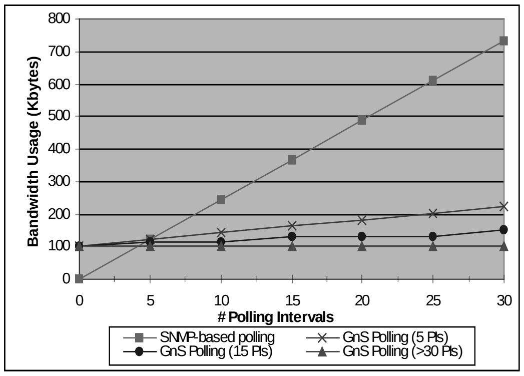


(a) (b)

Figure 4: Response time of (a) SNMP-based vs. GnG polling, (b) GnG polling as a function of the number of devices assigned to each MA object.



(a)



(b)

Figure 5: Bandwidth consumption for SNMP-based against (a) GnG and (b) GnS polling modes.

arbitrary to these experiments.

Response Time

In Figure 4, GnG polling mode is compared to an SNMP implementation [6] in terms of the response time for the acquisition of the HF result of Eqn. (1). Figure 4(a) shows that the flat approach (using one MA) does not scale well as the number of NEs increases. This makes it necessary to partition the managed network into several domains in order to maintain lower response times over the SNMP-based polling scheme.

Depending on the managed network size, the optimum number of domains can be determined from the minimum point of the corresponding curve of Figure 4(b). For instance, in a network of six devices, the response time is minimised when each MA is assigned to two NEs, i.e. the network is segmented into three domains. Hence, the manager application (through the PTs) should autonomously adapt the number of domains according to the current number of managed devices. This issue is currently under investigation.

Bandwidth Consumption

Figure 5 illustrates the traffic overhead imposed when applying SNMP-based, GnG and GnS polling schemes, according to equations (2), (3) and (4), respectively.

For this experiment, $S_{req/res}$=50 bytes (on application layer), S_c=2.1Kb, S_s=205 bytes, while n=50 devices and v=5 operational values. It is noted that the starting point for GnG/GnS polling is at 105Kbytes (50×2.1Kb), derived from the first terms of Eqn. (3) and (4). Clearly, both GnG and GnS modes represent a notable improvement over SNMP-based polling. In addition, the segmentation of the managed network into several domains (use of more than one MAs in GnG polling) does not seriously effect bandwidth consumption, while GnS mode becomes more attractive as p increases.

The selection of the appropriate polling mode (and the associated parameters) is, therefore, a compromise between network overhead and response time, depending primarily on the type of management data to be collected.

6. Conclusions

Two novel polling modes that exploit the capabilities of MAs in NPM have been described. First, we introduced the GnG mode that improves the scalability of recently reported MAFs and may be used for obtaining real-time NM data as overall response time is minimised. This method is also preferable when considering the management of remote LANs.

Concerning the off-line analysis of management data, we propose the GnS mode. In this approach, MAs collect a larger amount of data before returning to the manager leading to a direct reduction in the number of MA transfers.

In both cases, remote data aggregation methods are applied to reduce the bandwidth usage. Empirical results indicate a significant improvement in both response time and traffic overhead when comparing the introduced polling modes to traditional centralised polling.

References

[1] Ku H., Luderer G., Subbiah B., "An Intelligent Mobile Agent Framework for Distributed Network Management", Proceedings of the IEEE Globecom'97, 1997.

[2] Susilo G., Bieszczad A., Pagurek B., “Infrastructure for Advanced Network Management based on Mobile Code”, Proceedings of the IEEE/IFIP NOMS'98, 1998.

[3] Gavalas D., Greenwood D., Ghanbari M., O’Mahony M., “An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology”, Proceedings of the IEEE ICC’99, 1999.

[4] Arnold K., Gosling J., “The Java Programming Language”, Addison-Wesley, 1996.

[5] Goldszmidt G., “On Distributed Systems Management”, Proceedings of the 3rd IBM/CAS Conference, 1993.

[6] AdventNet, URL: http://www.adventnet.com/.