Enabling Mobile Agent Technology for Intelligent Bulk Management Data Filtering

D. Gavalas, M. Ghanbari, M. O'Mahony
Communication Networks Research Group, Electronic Systems Engineering Department,
University of Essex, Colchester, U.K.
{dgaval,ghan,mikej}@essex.ac.uk

D. Greenwood
Distributed Network Management and Agent Technology Research Group
Fujitsu Telecommunications Europe, Colchester, U.K.
D.Greenwood@ftel.co.uk

Abstract

The intrinsic scalability limitations of traditional centralised Network Management (NM) become significantly more pronounced when transfers of bulk network monitoring data are considered. This fact has encouraged a trend towards distributed management intelligence, which promises more flexible and scalable solutions. However, distributed Mobile Agent (MA) based NM architectures reported in the literature do not adequately address scalability problems when considering data intensive NM applications. In this paper, we present three novel applications in which MAs are used to perform management data aggregation, acquire SNMP table snapshots and filter SNMP table contents subject to filtering expressions. Both real-time and off-line NM data acquisition is considered. The applications, supported by a lightweight management framework described in previous work, are shown to outperform SNMP-based polling in terms of bandwidth consumption.

Keywords

Distributed network management, Mobile agents, Information aggregation, SNMP table filtering.

1. Introduction

Standardised network management (NM) protocols, such as the Simple Network Management Protocol (SNMP), are based on the centralised, client/server model, where a central station (manager) collects and analyses data retrieved from physically distributed servers (agents). This process typically involves massive transfers of management data causing considerable strain on network throughput and creating a processing bottleneck at the manager host.

These scalability problems suggest distribution of management intelligence as a rational approach to overcome the limitations of centralised NM. The IETF has

proposed an approach, known as RMON (Remote MONitoring) [1], which introduces a degree of decentralisation. RMON monitoring devices (*probes*) collect management statistics from their local domain (e.g. an Ethernet segment), providing detailed information concerning traffic activity. However, this approach is cost intensive, as it typically requires a stand-alone RMON compliant device (probe) in every network segment. In addition, RMON represents a rather inflexible approach, as the control operations of a probe may be set/modified only at configuration time.

In terms of research activities, Management by Delegation (MbD) [2] represents a first effort towards decentralisation of management functionality. The initial approach was to download management scripts that were compiled and executed at the agent side. However, the advent of Java has made this task significantly simpler.

The idea of management distribution is taken further by solutions that exploit Mobile Agents (MA), which provide a powerful software abstraction that allows code migration between hosts for remote execution [3]. MAs can be regarded as a 'superset' of static delegation agents proposed in the MbD paradigm, in the sense that they may be downloaded to a managed device and execute an intelligent management function (as a separate process), having the additional benefit of mobility. The data throughput problem can be addressed by delegation of authority from managers to MAs, which are able to process and filter data locally without the need for transmission to a central manager.

The advantages brought about by using MAs, in comparison with the static delegation agents proposed by MbD, are summarised in the following:

- Ease in modifying existing management functions. This is because management functions are developed/modified centrally, with the modifications taking instant effect. When static agents are used (each of which is dedicated to a managed device), each must be updated by an 'update' message broadcast by the manager. Frequent modifications would create a considerable amount of traffic.
- Efficient use of computing resources on the managed entities, as management functions are executed only as long as the MAs reside and are active on the Network Elements (NE) [3].
- The mobility of MA components intrinsically implies a domain or global level view of the managed network, as MAs visit several or even all the network hosts. This allows MAs to apply a second stage of management data filtering (in addition to the first stage which is performed on the host where the data were originally collected), at domain or network level. That leads to a further reduction of bandwidth consumption as only a small portion of the acquired data is sent to the manager, whilst relieving the manager host from a considerable processing burden.

These advantages have attracted much attention to MA technology, with several Mobile Agent Frameworks (MAF) proposed for NM applications [4][5][6][7]. However, most of these frameworks have been mainly used in traffic analysis as well as in fault and configuration management areas. In contrast, not

much work has been undertaken concerning network performance management and network monitoring. This is because the overhead imposed by the frequent MA transfers, required to perform NEs polling, may even surpass that of centralised models. The factors contributing to that high overhead are:

(i) The prohibitively bandwidth 'expensive' MA size;

(ii) No appropriate method for remote processing/filtering of network monitoring data or SNMP tables has been proposed/implemented and, as a result, the volume of data transferred to the manager is not significantly decreased.

In order to deal with the first problem, we have presented an infrastructure that performs dynamic MA-based NM [7] utilising a lightweight MA code transfer scheme (discussed in the succeeding section). In [8], we proposed two complimentary polling modes that improve the infrastructure scalability: In the first approach, termed 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 to obtain requested information before returning to the manager. The administrator may define a maximum number of devices per domain to minimise response time. The second polling scheme, termed Go 'n' Stay (GnS), targets the acquisition of data to be analysed off-line: an MA object is broadcasted to all managed devices and remains there for a number of PIs collecting an equal number of samples, before returning to the manager.

In this paper, we focus on the second of the aforementioned problems using the infrastructure described in [7] to employ NM data intelligent filtering applications. In particular, we describe ways to: (i) 'multiplex' several Management Information Base (MIB) values into more meaningful aggregated values, (ii) efficiently acquire snapshots of SNMP tables, and (iii) filter tables' contents on using arbitrarily complex filtering expressions (wherein logical AND and OR operators may be incorporated to correlate individual filtering functions). Regarding the table filtering application, we introduce the idea of domain or global filtering. Specifically, we exploit the multi-node movement that MAs often undertake that allows them to perform a superjacent level (second stage) of data filtering, in domain or even in network level.

The paper is organised as follows: Section 2 provides an overview of the MAF used to support this work. Section 3 deals with three proposed applications of MAs in network monitoring. A quantitative evaluation in terms of the bandwidth consumption is given in Section 4, with Section 5 concluding the paper.

2. Mobile Agent Network Management Framework

The MA-based NM framework has been entirely developed in Java [9] chosen mainly due to its inherent portability, rich class hierarchy and dynamic class loading capability. The main reason that motivated the development of a new MA infrastructure from scratch, rather than using an existing research or commercial platform has been the fact that the latter adopt a heavy-weight approach regarding

MAs transfers, in addition to the convenience in structuring a system tailored to certain management applications, according to their special requirements.

A key issue affecting the performance of an MA-based NM infrastructure is the size of travelling MA entities. In a typical Java-based MAF, both the MA *code* and *state* are required at the destination to instantiate the received MA objects. Nicklisch et al. [10] identified three alternative agent code transfer schemes: (a) "*push*" (MA code is sent from a code repository to all the managed devices prior to MA transfers); (b) "*pull*" (code is loaded by the NE from a repository upon the MA's arrival); (c) "*migrate*" (migrating code is sent from one non-repository NE to another).

To the best of our knowledge, all existing MA-based NM framework implementations, such as [4][5][6], use the "migrate" code transfer scheme, i.e. MA's bytecode is transferred along with its state in every single MA transfer. This results in a high demand on network resources, as code size is typically much larger than state size. In our infrastructure the transfer of the MA bytecode is performed only *once* (at its construction time), through broadcasting it to the active managed devices, i.e. we use the "push" code transfer scheme [7]. Thereafter, the transfer of persistent state is sufficient for the MAS entities to recognise an incoming MA and recover its state.

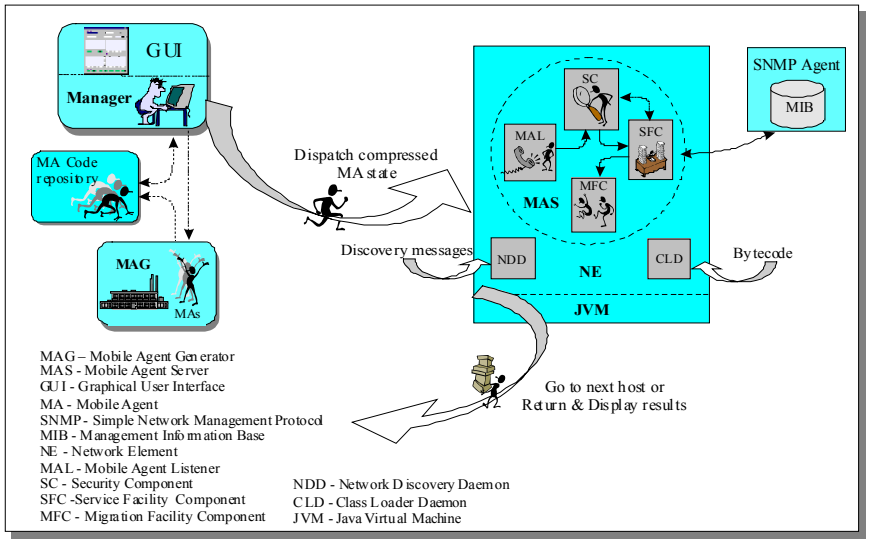

Figure 1. The Mobile Agents-based NM Infrastructure

Our framework consists of four major components, illustrated in Figure 1:

(I) Manager Application

The manager application, equipped with a browser style Graphical User Interface (GUI), 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'.

(II) 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 standard SNMP agent, creating an efficient run-time environment for receiving, instantiating, executing, and dispatching MA objects. Integration with SNMP was vitally important to maintain compliance with the legacy management systems. The MAS composes four primary components (see Figure 1):

- Mobile Agent Listener (MAL): a daemon that listens on well-known TCP and UDP ports for incoming MAs.
- Security Component (SC): acts as the system's protective shield. The RSA algorithm has been implemented to provide both authentication of incoming MAs and encryption of sensitive NM information.
- Service Facility Component (SFC): serves as an interface to the physical resources. The MA makes use of this component to interact indirectly with the SNMP agent and obtain system information.
- Migration Facility Component (MFC): upon an MA's request, it serialises [9] and dispatches the MA object to the next host.

(II) Mobile Agent Generator (MAG)

The MAG is essentially a tool for automatic MA code generation allowing the construction of customised MAs in response to service requirements. Generated MA code is stored into the MA code repository (see Figure 1). 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 [7]. However, its functionality has been extended so as to allow the operator (through a dedicated GUI) to specify: whether the constructed MA will be used for GnG or GnS polling; the polling frequency (i.e. the polling interval's duration); the transport protocol to be used for the MA transfers (either TCP or UDP); the security policies.

(IV) Mobile Agents (MAs)

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.

3. Intelligent filtering applications of NM data

The management operations defined in the IETF approach are usually very low-level, as the management station can typically only get and set values on the

management agents' MIB. Semantically rich operations, such as *get-column*, *get-row* or *get-table* are not available yet. Using the MA-based approach, sequences of primitive operations can be grouped into higher-level operations, sent to the NEs and executed independently of the management station. That results in improved performance by reducing the number of messages exchanged between the agent and the management station, thereby limiting the load in the area surrounding it. In the following sections, we describe three novel applications of MAs on network monitoring, demonstrating their ability to minimise management data overhead.

3.1. Health Function Evaluation

Polling is a frequent operation in NM as there are often several object values that require constant monitoring. Cases often occur however, where one or two MIB variables are not a representative indicator of system state and hence an aggregation of multiple variables is required, known as a *health function* (HF) [11]. For instance, *five* MIB-II objects [12] are combined to define the percentage *E(t)* of IP output datagrams discarded over the total number of datagrams sent during a specific time interval,

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

where MIB-II is an example of a MIB supported by all available SNMP agents.

In the SNMP model, the least 'expensive' option would be to group the five Object Identifiers (OIDs) into a single *get* request packet. The response packet would then include the OIDs along with the requested values, with the OIDs typically occupying more space than the actual values. On the other hand, the MAs constructed by the MAG tool are able to perform *semantic compression* of management information. Thus, the value of *E(t)* is computed locally, with a single value returned to the manager station. Hence, the manager is relieved from processing NM data, while the MAs state size remains as small as possible. MAs can also be instructed to transmit computed values only when certain thresholds are crossed. When the collected values are intended for off-line analysis, GnS polling mode may be employed and multiple samples wrapped into the MA's state before delivered to the manager application, reducing the associated network overhead to a great extent.

3.2. SNMP Table Polling

Some of the major drawbacks with SNMP are related to the bulk transfer of data, e.g. the transfer of large SNMP tables. The widely deployed SNMPv1 was not designed for transferring large amounts of data. In addition, the total amount of management information that needs to be transferred has been increased greatly, e.g. IP routing tables and TCP connection tables are continuously growing. Later protocol versions (v2c & v3), apart from their limited installation basis on managed devices, do not answer this problem sufficiently, even though they provide a *get-bulk* operation [13].

Let us consider the retrieval of an SNMP table consisted of thousands entries. When using the *get-next* operator (SNMPv1 model) the table retrieval requires at least one get-next operation per table row (see Figure2.a). Apart from the apparent impact on network resources, this operation is known to experience significant latency, especially when the management of remote LANs is considered (each get-next operation should be completed before the next one can start) [14]. In addition, the non-negligible time intervals between the acquisition of each individual object value leads to potential inconsistencies (different sections of the table might reflect updates at different times).

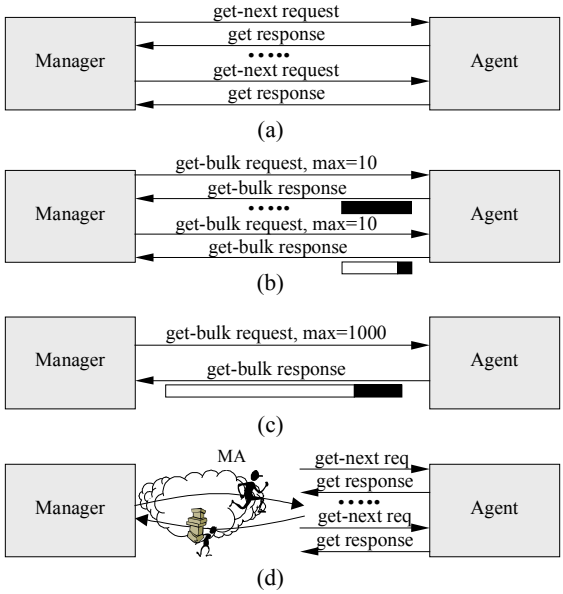

Figure 2: Acquiring an SNMP table snapshot through: (a) successive get-next requests, (b) multiple get-bulk requests, (c) a single get-bulk request, (d) MA migration and locally issued get-next requests.

The situation improves with the introduction of the get-bulk request, which adds to the ability of SNMP to retrieve large blocks of data efficiently by specifying a maximum number of successive values to be returned (*max-repetitions*) [13]. That means that the human manager has to guess a value for the max-repetitions parameter. Using small numbers for max-repetitions may result in too many message exchanges (Figure 2.b). Using large numbers, however, may result in an 'overshoot' effect [14]: the agent returns data that do not belong to the table the manager is interested in. This data will be sent over the network back to the manager just to be discarded (Figure 2.c).

Another factor contributing to the high overhead of the various SNMP implementations is the OID naming scheme. In particular, the OIDs of the objects involved in bulk transfers are characterised by a high degree of redundancy, i.e.

multiple occurrences of identical portions of OIDs can be observed. Therefore, redundant information is transferred, resulting in higher network overhead than strictly needed.

Here, we propose a way to improve the retrieval of SNMP tables both in terms of network overhead and latency. An MA object is dispatched by the manager and visits a pre-determined number of hosts. At each place of contact, when received by the local MAS entity daemon, the MA acquires an SNMP table through successive get-next requests (see Figure 2.d). The table contents are then encrypted, if desired, and encapsulated into its state before moving to the next host or returning to the manager. The MA may alternatively obtain several snapshots of the table (with a pre-determined frequency) and wrap them all into its state before delivering to the manager for further analysis (GnS polling mode).

The overall latency is also reduced (especially for large tables), as the round-trip delay of each request/response message exchange is significantly smaller. An inviting side effect of that is the improved consistency of the acquired values.

The SNMP table is encapsulated into the MA's state as a two-dimensioned array. That solves the OID redundancy problem, since the OIDs are not returned to the manager at all; the table's OID (which is the common prefix for all the table objects) added to the value's location into the array (column, row) is sufficient to build the corresponding object's OID.

New polling operations, used to acquire specific SNMP table views may be added/modified at runtime, specifying the SNMP table and the hosts to be polled, the polling interval, the polling mode (either GnG or GnS), the transport protocol to be used, etc. The implementation of the local interactions through get-bulk (instead of get-next) requests is currently under investigation and expected to decrease further the overall latency.

Regarding the MA-SNMP agent interaction implementation we re-used publicly available software as much as possible. Thus, we have used several classes of the AdventNet SNMPv1 package [15] offering MIB browsing capability, abstracting MIB nodes, issuing SNMP requests, etc. The higher-level operations, built on the top of AdventNet package classes (e.g. get-table) and invoked by incoming MAs, have been integrated into the SFC component of MAS modules.

3.3. SNMP Table Intelligent Filtering

In most existing network monitoring applications only a small portion of the obtained data is proved useful as, in by far the majority of cases, bandwidth is consumed to learn nothing other than that the network is operating within acceptable parametrical boundary conditions. This is because the processing action, i.e. the filtering of management data takes place on the manager and not on the NE side. Therefore, we propose a third application of MAs on network monitoring exploiting their ability to download management logic in order to perform intelligent filtering of NM data in selected SNMP tables. Specifically, this type of MA is able to acquire an SNMP table and subsequently apply a pre-determined filtering pattern to it.

The filtering operators offered in the current implementation are classified in *Arithmetic* (Max, Min, Bigger, Less) and *Textual* (Match, Exclude). These operators typically take as input the acquired SNMP table and filter it keeping only the rows for which a given element (defined by its column index) meets certain criteria, e.g. is greater than a threshold value or matches a given text string.

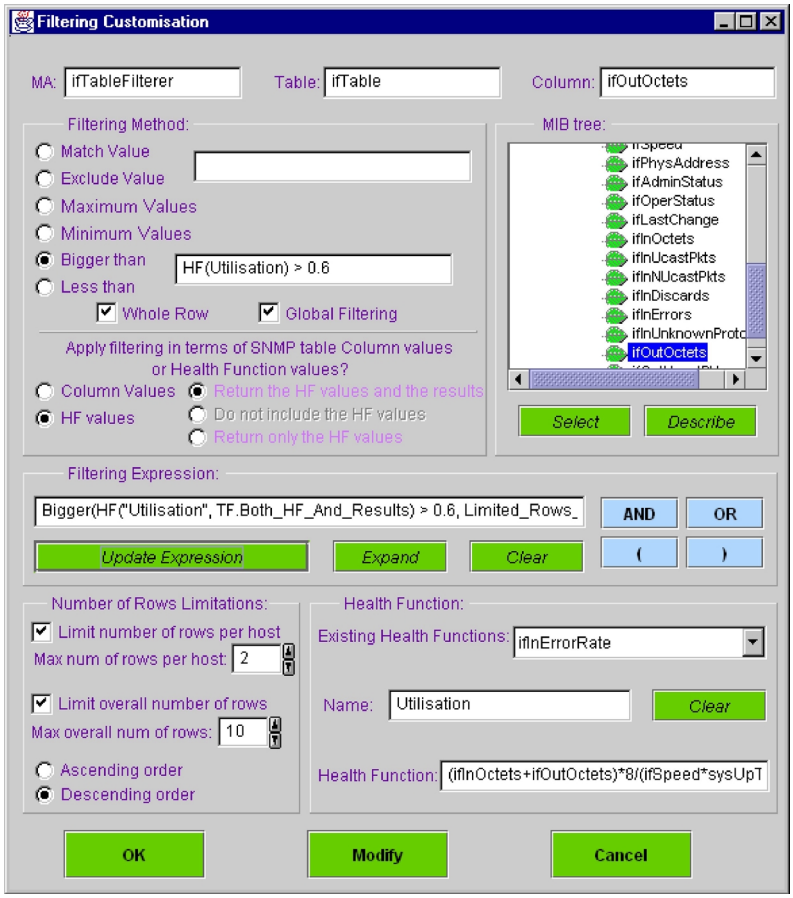

Figure 3: Customising the SNMP table filtering operation parameters

When arithmetic operators are considered, the user may set (through the GUI shown in Figure 3, launched from the MAG user interface) limitations on the maximum number of rows that may be returned from individual hosts, the maximum overall number of rows, whether the results will be sorted in ascending or descending order, etc. For instance, the method definition of the *Bigger* operator is:

```
String[][] Bigger (String table[][], int colIndex, double biggerThan, int
                   rowsPerHost, int overallRows, boolean ascending);
```

It is also noted that the filtering operation may be either based on: (i) a given table column, e.g. for the MIB-II *interfaces* table (ifTable) [12], "*get the two rows*

(interfaces) with the maximum number of incoming octets (max ifInOctets)", or (ii) on a pre-defined HF, e.g. "*return the two more heavily loaded interfaces*" (see Figure 3).

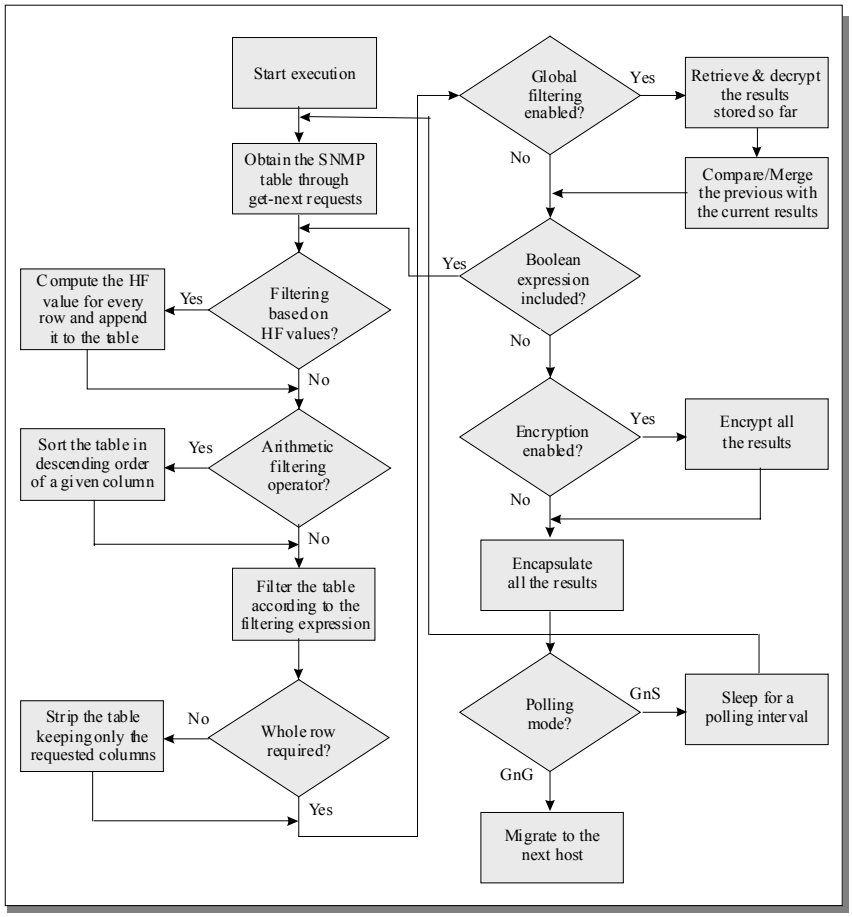

Figure 4: SNMP table filtering operation flow diagram

For instance, the invocation of the *Bigger* method with the following parameters:

```
String[][] bigger = Bigger (Utilisation (getTable ("ifTable")), HFcolumn, 0.6,
                    2, 10, false);
```

returns the two more heavily loaded interfaces from each host, in descending order, given that their utilisation is greater than 60%. A maximum overall number of ten interfaces will be returned to the manager coming from any of the polled NEs. The *Utilisation* method, included in the MA's code, takes as a parameter the interfaces table and calculates the utilisation HF for each one of its rows:

$$U(t) = \frac{(ifInOctets + ifOutOctets) * 8}{(ifSpeed * SysUpTime * 100)} \quad (2)$$

The HF values are appended to the interface table, which is then sorted into HF values order. The resultant table is scanned and the rows (two, at maximum) with HF values bigger than 0.6 are returned, in descending order. For the case where only specific table columns are desired (e.g. only the number of octets sent out of the most heavily loaded interfaces), the rest of the columns will be removed.

In addition to the simple filtering issues discussed so far, we introduce the concept of *domain* or *global level* filtering. In particular, we exploit the multi-node movement of MAs to perform an additional level (second stage) of data filtering, in domain or even in network level. This is achieved by comparing/merging the results already collected with these that have been just obtained/processed. Hence, not only is the manager host relieved from processing bottlenecks, but the MA's state size is prevented from growing rapidly (and therefore the network overhead is further reduced), since the amount of information stored in the MA's data folder basically remains constant.

It should be emphasised that global filtering fits comfortably into both the GnG and GnS polling modes. In the former case the MA may, for example, return the two most heavily loaded interfaces found in the entire *network*, whereas in the latter, record a utilisation peak on a host within a given observation period. When employing GnS polling, higher polling frequencies may be used without putting any additional load on the network (since this will not affect the MA's state). In GnG polling mode, the results will be delivered to the manager and then displayed on a graphical table component, with the interface information drawn in different colours, depending on the host they are arriving from. This filtering operation is summarised in the Figure 4 flow diagram.

MAs may be constructed with the ability to apply arbitrarily complex boolean expressions, namely logical *AND* and *OR* operators correlating individual filtering functions. An example employing the *AND* operator could be to: "*return the interfaces with utilisation 0.6<U(t)<0.8*". The output array *bigger* of the *Bigger* method (mentioned in a previous example) would then be passed as a parameter to the *Less* operator to apply a second level of filtering (see Figure 4):

String[][] final_result = *Less* (*bigger*, HFcolumn, 0.8, 2, 10, false);

In contrast, when the *OR* operator is considered, the two output tables (arrays) resulting from the individual expressions are *concatenated*. For example, to view the TCP connections with either '*listen' or* '*established'* state (see the definition of the MIB-II tcpConnTable [12]), the following expression needs to be applied:

String [][] final_result = *concat* (*Match* (getTable ("tcpConnTable"), tcpConnState, "listen"), *Match* (getTable ("tcpConnTable"), tcpConnState, "established"));

4. Quantitative Evaluation

We have compared our proposed applications with the AdventNet SNMPv1 implementation [15]. A detailed analysis of response time measurements can be found in [8]. Here, we focus on performance measurements related to network overhead.

We consider a network of 50 managed devices. The SNMP *get* request/response message is 90 bytes long, on average (at MAC layer), while every extra value included in the SNMP packet's *varbind* list represents an additional overhead of 17 bytes, on average. Table 1 summarises all the MA attributes required to evaluate the network overhead of the introduced applications. The MA code size corresponding to the three filtering applications is retained minimum causing a light effect in terms of the bandwidth usage.

	HF Evaluation	SNMP table polling	SNMP table filtering
Compressed code size (in Kbytes)	1.25	1.36	1.95
Compressed (initial) state size (in bytes)	381	384	447
State size increment per sample (in bytes)	2 (one extra value)	68 (8×21 extra values)	13 (1×21 extra values)

Table 1: Attributes of the MA classes corresponding to the three proposed applications

Figure 5a compares the performance of SNMP-based polling against the MA-based approach, when the calculation of the HF appearing in Eqn. (1) is considered. Despite the object values aggregation, GnG mode does not outperform the centralised polling (the performances of these two approaches will start to converge for larger number of aggregated values). In addition, the segmentation of the managed network into *five* domains (employing 10 MAs in GnG polling) does not seriously effect bandwidth consumption, while reducing greatly the overall response time [8]. Things improve with GnS mode, which becomes more attractive as the number of polling intervals (PI) that MAs remain on the devices, increases. It is noted that the starting point for GnG/GnS polling is at 62.5Kbytes, representing the overhead imposed when broadcasting the compressed MA code to all NEs (50×1.25Kb).

Figure 5b, drawn on a logarithmic scale, compares our table polling method with the traditional approach, when considering the retrieval of an interfaces table consisted of *eight* rows (8×21 entries). We assume each get-next request retrieving a whole table row. The MA-based approach clearly surpasses SNMP-based polling due to the lightweight data 'encoding' method employed and the fact that the *whole* table is wrapped into MA's state before delivered to the manager. It is worth noting that the network partitioning into *five* domains provides better results in this case. This is explained by the rapid growth of the MA's state [8] encountered when

a single MA object is responsible for polling all NEs, since the MA state size is increased by 68 bytes each time a NE is visited (see Table 1).

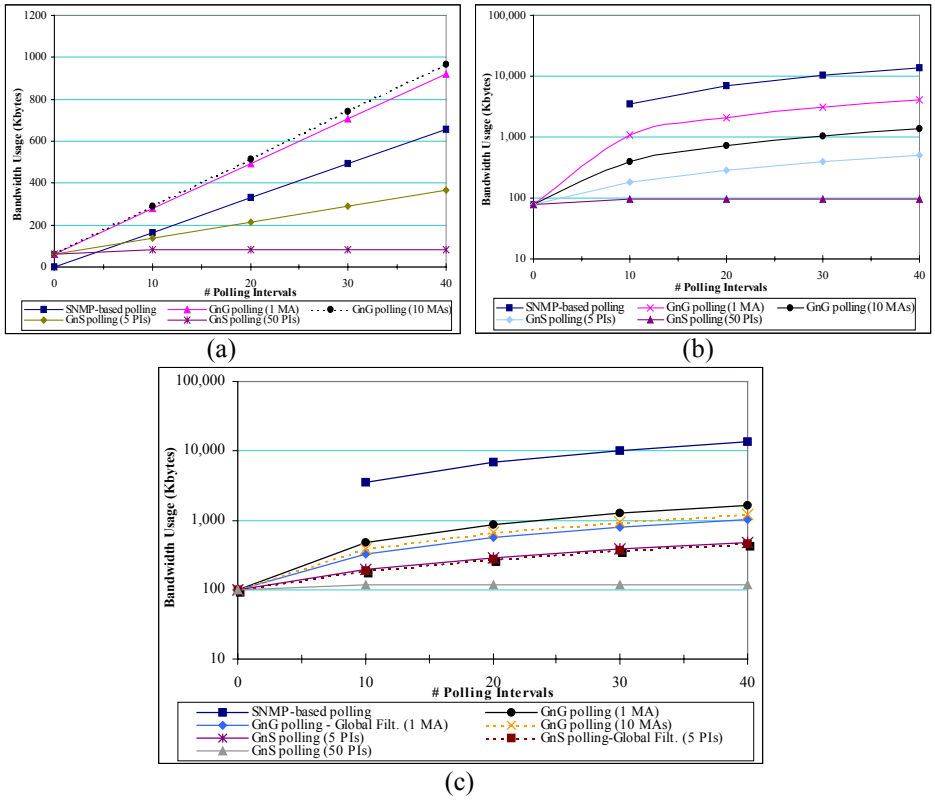

Figure 5: Bandwidth consumption of SNMP-based polling against MA-based proposed applications: (a) HF evaluation, (b) SNMP table polling, and (c) SNMP table filtering.

The same experiment is repeated for the table filtering application, with the pre-eminence of the former against SNMP-based approach being more distinct (see Figure 5c). Here, we assume MAs to bring back only the most heavily loaded interface (from each host). It is also noted that global filtering improves the framework's performance both in GnG and GnS cases, since it keeps the size of the MAs state constant (only the most heavily loaded *network* interface is returned).

5. Conclusions

In this paper, we have presented three applications of intelligent/mobile agents on network monitoring. The applications have been built on the top of a flexible, lightweight MAF described in previous work [7]. MAs have been utilised to: (i) aggregate several MIB values into more meaningful network health indicators, (ii)

acquire SNMP tables snapshots, and (iii) filter SNMP tables contents applying complex filtering expressions. We have also exploited MAs ability to realise multi-node itineraries to introduce the concept of domain/global filtering, where MAs use the knowledge/information already collected to perform a superjacent level of data filtering.

Empirical results indicate a significant improvement on traffic overhead when testing the proposed applications in realistic management scenarios and comparing them against traditional centralised polling.

Future work will address:

- Investigation of the problem of integrating legacy non-SNMP components in the NM framework.
- Exploration of other NM areas where MA technology can be employed, such as network performance testing or software distribution.

References

[1] S. Waldbusser, "Remote Network Monitoring Management Information Base", RFC 1757, 1995.

[2] Yemini Y., Goldszmidt G., Yemini S., "Network Management by Delegation", Proceedings of ISINM'91, pp. 95-107, 1991.

[3] Bieszczad A., Pagurek B., White T., "Mobile Agents for Network Management", IEEE Communication Surveys, Vol. 1, No 1, pp. 2-9, September 1998.

[4] Susilo G., Bieszczad A., Pagurek B., "Infrastructure for Advanced Network Management based on Mobile Code", Proceedings of NOMS'98, pp. 322-333, 1998.

[5] Sahai A., Morin C., "Towards Distributed and Dynamic Network Management", Proceedings of NOMS'98, 1998.

[6] Feridun M., Kasteleijn W., Krause J., "Distributed Management with Mobile Components", Proceedings of the IM'99, pp. 857-870, 1999.

[7] 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 ICC'99, pp. 1362-1366, 1999.

[8] Gavalas D., Greenwood D., Ghanbari M., O'Mahony M., "Complimentary Polling Modes for Network Performance Management Employing Mobile Agents", Proceedings of Globecom'99, 1999.

[9] Sun Microsystems: "Java Language Overview – White Paper" [On-line] (1998), http://www.javasoft.com/docs/white/index.html.

[10] Nicklisch J., Quittek J., Kind A., Arao S., "INCA: An Agent-Based Network Control Architecture", Proceedings of the 2nd Int. Workshop on Intelligent Agents for Telecommunication Applications (IATA'98), pp. 143-155, 1998.

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

[12] McCloghrie K., Rose M., "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC 1213, 1991.

[13] Stallings W., "SNMP, SNMPv2, SNMPv3 and RMON 1 and 2", 3rd ed., Addison Wesley, 1999.

[14] R. Sprenkels and J.P. Martin-Flatin, "Bulk Transfers of MIB Data", The Simple Times, 7(1):1-7, 1999, http://www.simple-times.org/.

[15] AdventNet, http://www.adventnet.com/.