Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology

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

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

[*]Distributed Network Management and Agent Technology Research Group
Fujitsu Telecommunications Europe Ltd.,
Northgate House, St. Peters Street, CO1 1HH, Colchester, U.K.
E-mail: D.Greenwood@ftel.co.uk

Abstract

Mobile Agents (MA) have been proposed as a solution for distributed Network Management (NM). However, most MA-based infrastructures exhibit scalability limitations when data intensive management applications are considered. Therefore, we present three novel applications, tailored to transfers of bulk network monitoring data, in which MAs are used to perform data aggregation, acquire atomic SNMP table views and support selective retrieval of SNMP table objects that meet specific selection criteria. The proposed applications are supported by a lightweight management framework described in previous work. A quantitative evaluation, in terms of bandwidth usage, shows that these applications surpass SNMP-based polling performance.

Keywords

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

1. Introduction

Current Network Management (NM) systems are typically designed according to a centralised NM paradigm characterised by a low degree of flexibility and re-configurability. Management interactions are based on a centralised, client/server model, where a central station (manager) collects, aggregates and processes data retrieved from physically distributed servers (agents[1]). Widely deployed NM standards, such as the Simple Network Management Protocol (SNMP) [3] of the TCP/IP protocol suite or the Common Management Information Protocol (CMIP) [9] of the OSI world, are designed according to this rigid centralised model. Within these protocols, physical resources are represented by managed objects. Collections of managed objects are grouped into tree-structured Management Information Bases (MIB[2]) following the Abstract Syntax Notation 1 (ASN.1) format.

The centralised approach in NM is known to present severe efficiency and scalability limitations: the process of data collection and analysis typically involves massive transfers of management data causing considerable strain on network throughput and processing bottlenecks at the manager host. All these problems suggest distribution of management intelligence as a rational approach to overcome the limitations of centralised NM. The Internet Engineering Task Force (IETF) has proposed an approach, known as RMON

[1] The term 'agent' here refers to static management agents, which should not be confused with Mobile Agents.

[2] MIB is an SNMP term, which corresponds to the term Management Information Tree (MIT) used in CMIP. Hereafter, we ignore the difference for sake of simplicity.

(Remote MONitoring) [14], 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.

In terms of recent research activities, Management by Delegation (MbD) [17] represents a clear effort towards decentralisation of management logic. 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 interaction paradigm that allows code migration between hosts for remote execution [2][18]. The data throughput problem can be addressed by delegation of authority from managers to MAs, which are able to filter and process data locally without the need for transmission to a central manager. This ability has attracted much attention to MA technology, with several Mobile Agent Frameworks (MAF) proposed for NM applications [4][5][12][16]. 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 network elements (NE) polling, may even surpass that of centralised models. The factors contributing to that high overhead are:

(i) The prohibitively 'costly' MAs size.
(ii) No appropriate method for remote processing/filtering of network monitoring data or SNMP tables has been proposed 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 [5] utilising a lightweight MA code transfer scheme (discussed in the succeeding section). In [6], we proposed two complimentary polling modes that improve the infrastructure scalability: 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: 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 [5] to employ NM data intelligent filtering applications. In particular, we describe ways to: (i) aggregate several MIB values into more meaningful values, (ii) efficiently acquire atomic snapshots of SNMP tables, and (iii) filter tables' contents by applying arbitrarily complex filtering expressions. Regarding the table filtering application, we introduce the idea of *domain* or *global* level 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. This is achieved by merging the results collected by the hosts that have been already visited with the results that have been just obtained.

The paper is organised as follows: Section 2 discusses the advantages that MA-based approach brings to network monitoring applications over the static delegation agents of MbD paradigm and RMON. Section 3 provides an overview of the MAF used to support this work. Section 4 deals with the proposed applications of MAs on network monitoring. A quantitative evaluation of the bandwidth usage is given in Section 5, with Section 6 concluding the paper and suggesting several topics for further work.

2. Advantages of MA-based approach in NM

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

(i) Ease in modifying existing management functions. This is because management functions are developed/modified centrally, with the modifications taking instant effect. In case that static agents are used, each of them has to be updated by an 'update' message broadcasted from the manager to the managed devices. Frequent modifications would create a considerable amount of traffic.

(ii) Efficient use of computing resources on the managed entities, as management functions are executed only as long as the MA resides and is active on the NEs.

(iii) MAs can provide all the functionality offered by static delegation agents, having the additional benefit of mobility. In that sense, MAs can be regarded as a 'superset' of MbD agents and thus, whenever a static agent is sufficient for performing a management function, an MA can be sent to managed device and remain there until no longer needed.

(iv) 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. That allows the MAs to apply a second stage of management data filtering, at domain or network level, i.e. compare/merge the results acquired from each host with those already collected during their itinerary and keep only the values conforming to certain restrictions. That leads to a further reduction of NM data transfers as only a small portion of the originally acquired data is sent to the manager, while relaxing the manager host from a considerable processing burden.

(v) The impact on network bandwidth caused by MA migrations is not necessarily high. This issue is discussed later in this section.

RMON is also shown to present several deficiencies compared to MA-based NM:

(a) Typically, a stand-alone RMON compliant device (probe) is required to monitor the traffic activity of a single network segment, leading to considerable increase of cost when the management of multiple segments is considered. In contrast, in MA-based approach there is not any constraint on the number of segments that can be traversed/managed by MA objects.

(b) The control operations of a RMON probe may be set/modified only at *configuration* time. That represents a rather inflexible solution in comparison with the MA-based approach where a management function may be configured/modified at *runtime*.

(c) RMON is adequate for providing only *traffic-oriented* statistics since the status of the network is determined by direct inspection of the packets flowing in it, rather than inspection of the devices status, like in the mainstream (centralised) approaches that offer *device-oriented* statistics. That is not the case in the MA-based approach where management operations may be applied in both NE and network level.

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. [11] 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 MA's arrival); (c) "*migrate*" (migrating code is sent from one non-repository NE to another).

To the best of our knowledge, all the existing MA-based NM framework implementations, such as [4][12][16], 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 much higher demand on network resources, as code size is typically much larger than state size. In our infrastructure [5] the transfer of the MA bytecode is performed only *once* (at its construction time), through broadcasting it to the active managed devices, i.e. the "push" code

transfer scheme is used. Thereafter, the transfer of persistent state is sufficient for the MAS entities to recognise an incoming MA and recover its state.

3. The Mobile Agent Framework

The MA-based NM framework has been entirely developed in Java [15] chosen mainly due to its inherent portability, rich class hierarchy and dynamic class loading capability.

Our framework consists of four major components [5], illustrated in Figure 1:

1. The Manager application;
2. The Mobile Agent Server (MAS);
3. The Mobile Agent Generator (MAG);
4. The Mobile Agents.

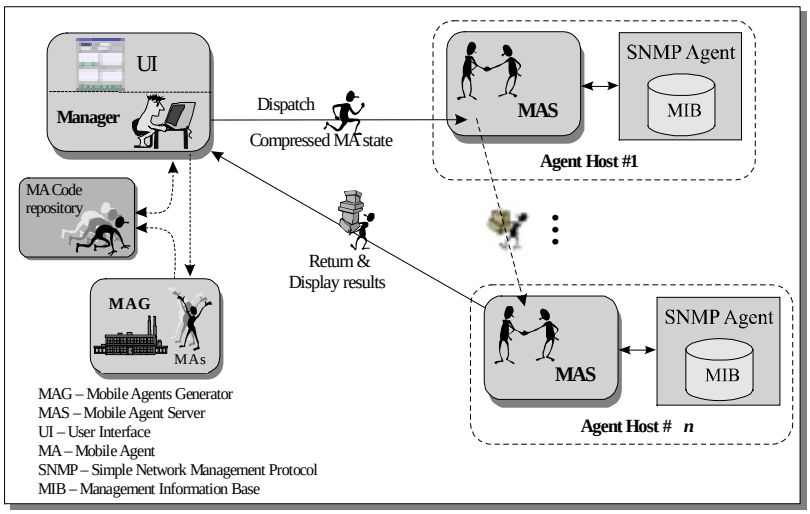

Figure 1. The Mobile Agents-based NM Infrastructure

3.1. 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'.

3.2. 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.

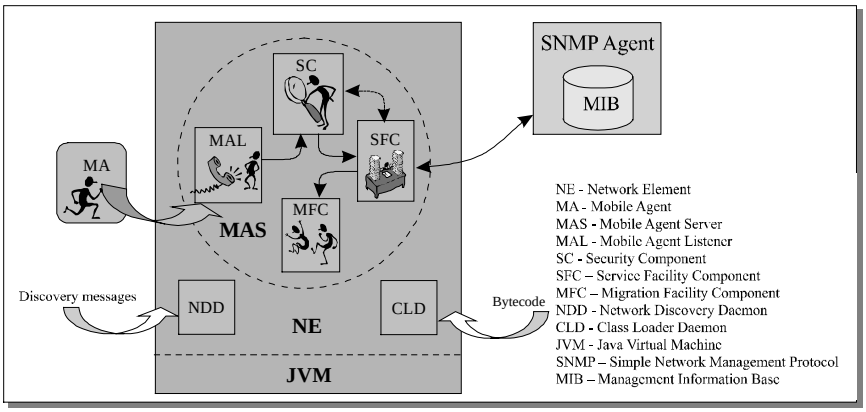

Figure 2. The Mobile Agent Server structure

The MAS 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 two supplementary threads exist outside the boundary of the MAS: (i) Network Discovery Daemon, (ii) Class Loader Daemon.

3.3. 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 an MA code repository (see Figure 1). Such MAs may dynamically extend the infrastructure's 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 [5]. However, its functionality has been extended so as to allow the operator (through a dedicated UI) 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 transmission protocol to be used for the MA transfers (either TCP or UDP);
- Whether MAs authentication and data encryption are applied or not.

3.4. 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. The MA's state information is compressed (using the Java *gzip* utility) before being transferred to the next destination host.

4. 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 MIB object values. 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. This brings forward the benefit of modularity and parallelism into the management architecture, and provides 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.

To facilitate the construction of service-oriented MAs, we have refined the MA model described in [5] by introducing an MA *superclass*, which provides the root attributes mentioned in the preceding section. In addition, we have implemented several classes each of which extends the MA superclass and corresponds to a specific management function. These in turn are sub-classed by the MA classes created by the MAG. This flexible hierarchical design minimises MA bytecode and eases the creation of service specialised MAs.

In the following sections, we describe three novel applications of MAs on network monitoring, demonstrating their ability to minimise management data overhead.

4.1. Health Functions 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) [7]. For instance, *five* MIB-II [10] objects 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,

$$E(t) = \frac{(ipOutDiscards + ipOutNoRoutes + ipFragFails) * 100}{ipOutRequests + ipForwDatagrams} \qquad (4\text{-}1)$$

where MIB-II is an example of a MIB being supported by all the SNMP-enabled NEs.

In the SNMP model, the least ‘expensive’ option would be to group the five Object Identifiers (OIDs) in 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 compute HFs, thereby providing a way to semantically compress large amounts of data into single indices representing portions of the system status. Thus, a single value is returned to the manager station, relieving it from processing NM data, while the MAs state size remains as small as possible. MAs can also be instructed to transmit computed values only in the case that certain thresholds are crossed. When the collected values are intended for off-line analysis, GnS polling mode may be employed and multiple samples be wrapped into the MA’s state before delivered to the manager application, reducing the associated network overhead to a great extend.

4.2. 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 [14].

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 Figure3.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) [13]. In addition, the non-negligible time intervals between the acquisition of each individual object value leads to potential inconsistencies (different sections of the table will reflect updates at different times).

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*) [14]. 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 3.b). Using large numbers, however, may result in an 'overshoot' effect [13]: 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 3.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.

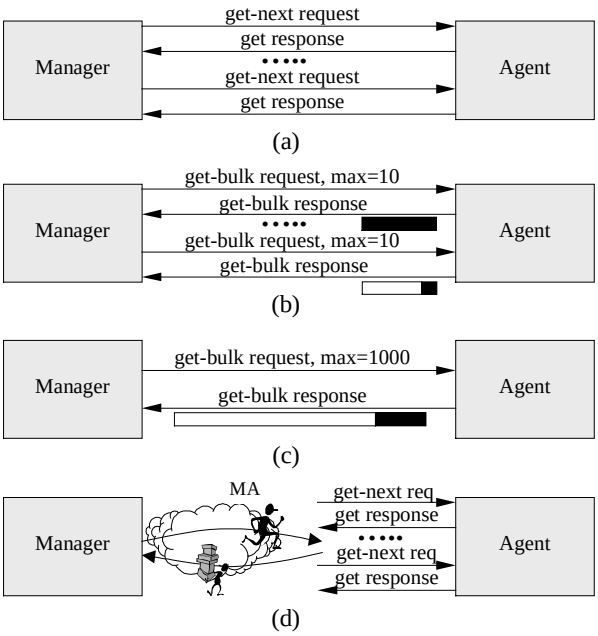

Figure 3: Acquiring an SNMP table snapshot through: (a) successive get-next requests, (b) multiple get-bulk requests, (c) a single get-bulk request, (d) MA migration and locally issued get-next requests.

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 its corresponding polling thread and visits a pre-determined number of hosts. At each place of contact, when received by the local MAS entity, the MA acquires an SNMP table through successive get-next requests (see Figure 3.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 also 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 in 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 (see Figure 4).

Regarding the MA-SNMP agent interaction implementation, instead of reinventing the wheel and developing everything from scratch, we re-used publicly available software as much as possible. Thus, we have used several classes of the AdventNet SNMPv1 package [1] 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 Service Facilitator Component of MAS modules.

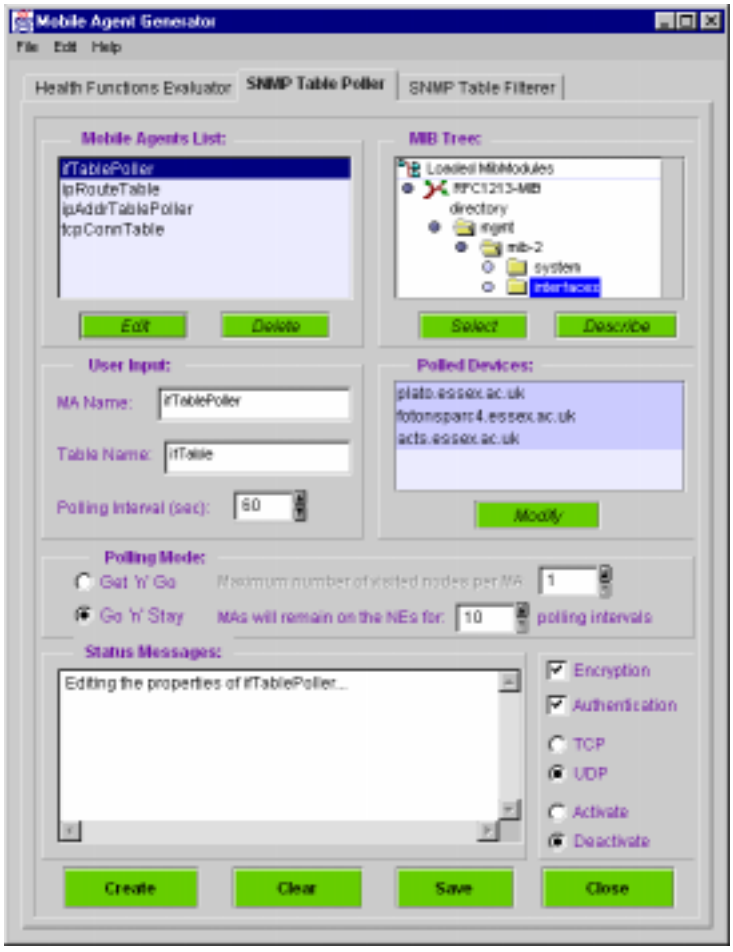

Figure 4: Configuring the SNMP table polling operation through the Mobile Agent Generator User Interface

4.3. SNMP Table Intelligent Filtering

In most existing network monitoring applications, the retrieved bulk data are usually utilised to feed a processing module responsible for extracting results in a more high-level and 'understandable' form. These results may be used later on to aid on capturing utilisation or error rate peaks, indicate failed devices, foresee possible congestion points, propose future network upgrades, etc. In fact, only a small portion of the obtained values 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 on 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) with their corresponding method definitions and parameters shown in Figure 5. 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.

```
String[][] Max (String table[][], int colIndex, int rowsPerHost, int overallRows, boolean
                ascending);
String[][] Min (String table[][], int colIndex, int rowsPerHost, int overallRows, boolean
                ascending);
String[][] Bigger (String table[][], int colIndex, double biggerThan, int rowsPerHost, int
                   overallRows, boolean ascending);
String[][] Less (String table[][], int colIndex, double lessThan, int rowsPerHost, int
                 overallRows, boolean ascending);
String[][] Match (String table[][], int colIndex, String matchedValue);
String[][] Exclude (String table[][], int colIndex, String excludedValue);
```

Figure 5: Arithmetic and textual filtering operators method definitions

When arithmetic operators are considered, the user may set 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.

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) [10], "*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 6).

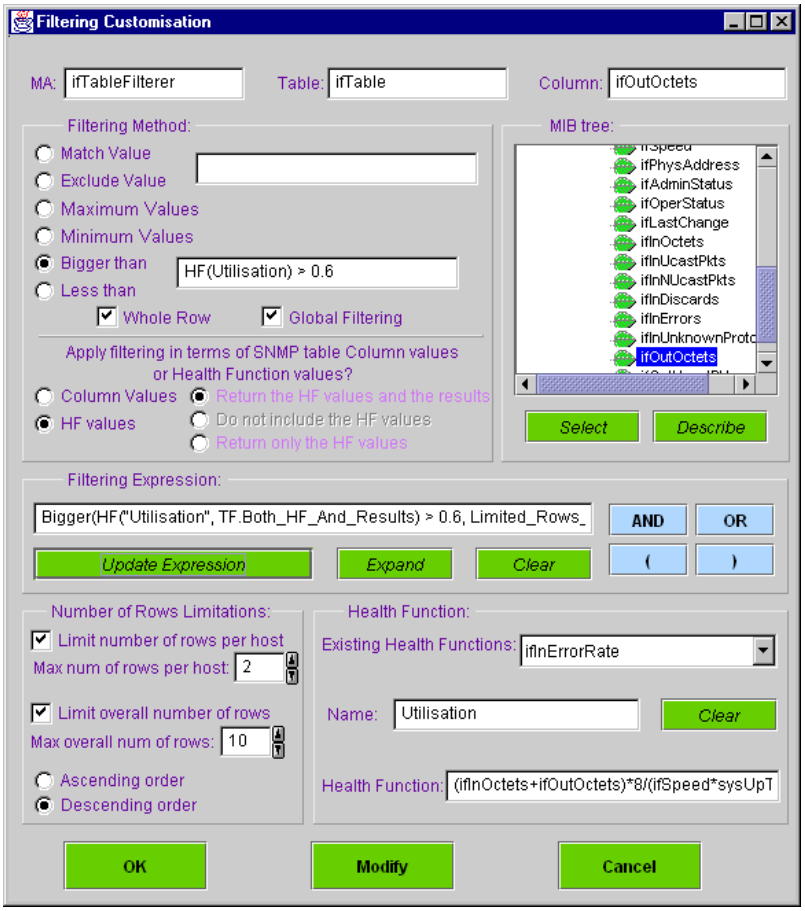

Figure 6: Customising the SNMP table filtering operation parameters

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, in descending order, given that their utilisation is greater than 60%. A maximum 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:

$$U(t) = \frac{(ifInOctets + ifOutOctets) * 8}{(ifSpeed * SysUpTime * 100)} \qquad (4\text{-}2)$$

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 greater 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 sample rates 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 7 flow diagram.

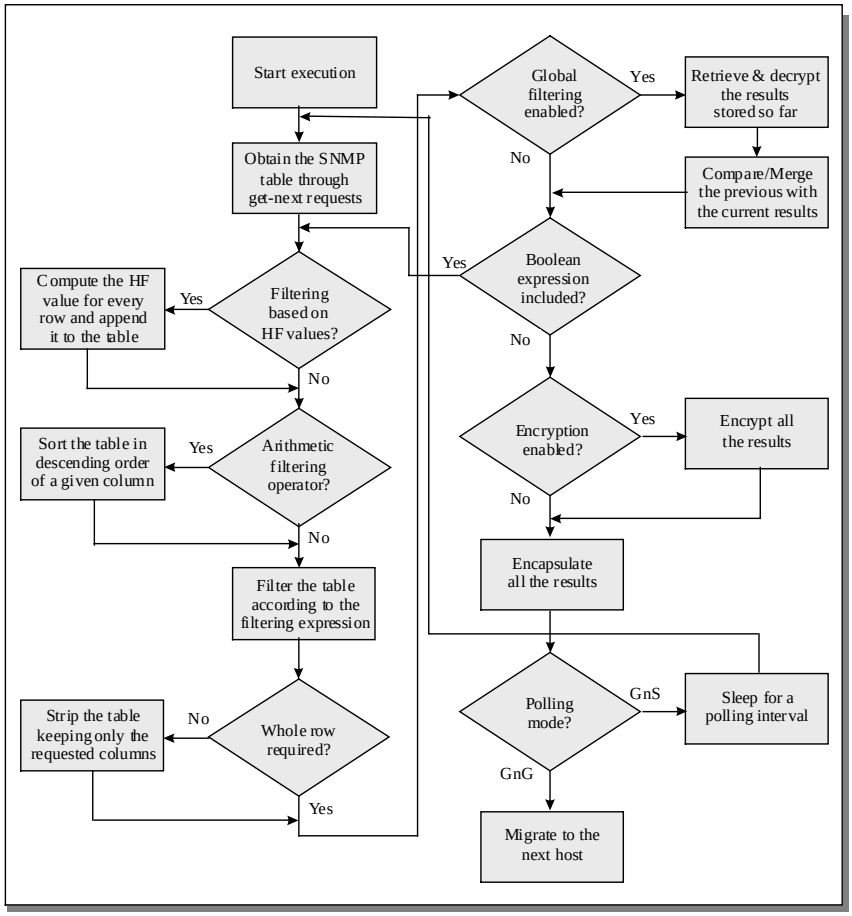

Figure 7: SNMP table filtering operation flow diagram

In addition to the scenarios already examined, the construction of filtering expressions with increased complexity has been also considered. Thus, 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 would be: "*return the interfaces with utilisation 0.6<U(t)<0.8*". The output array *bigger* of the *Bigger* method invocation (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 7):

String[][] 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 simply concatenated. For example, to view the TCP connections with either '*listen' or* '*established'* state (see the definition of the MIB-II tcpConnTable [10]), the following expression needs to be applied:

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

Relevant work has been reported in [8], which describes ways to create MIB views within an agent. Views are created by defining operations on SNMP tables using a View Definition Language (VDL). However, the support of a VDL interface (*VDL translators*) increases the footprint of delegation agents on network devices, while these agents need to be always updated as soon as a new control operation is introduced. This is not a prerequisite in the MA-based approach, where the local MAS modules are unaware of the incoming MAs functionality.

5. Quantitative Evaluation

The network overhead imposed by the proposed applications has been compared with the AdventNet SNMPv1 implementation [1]. 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.

	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 8(a) compares the performance of SNMP-based polling against the MA-based approach, when the calculation of the HF appearing in Eqn. (4-1) is considered. Despite the object value 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 [6]. 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 8(b), drawn on 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 [6] encountered when a single MA object is responsible for polling all NEs (the MA state size is increased by 68 bytes each time a NE is visited).

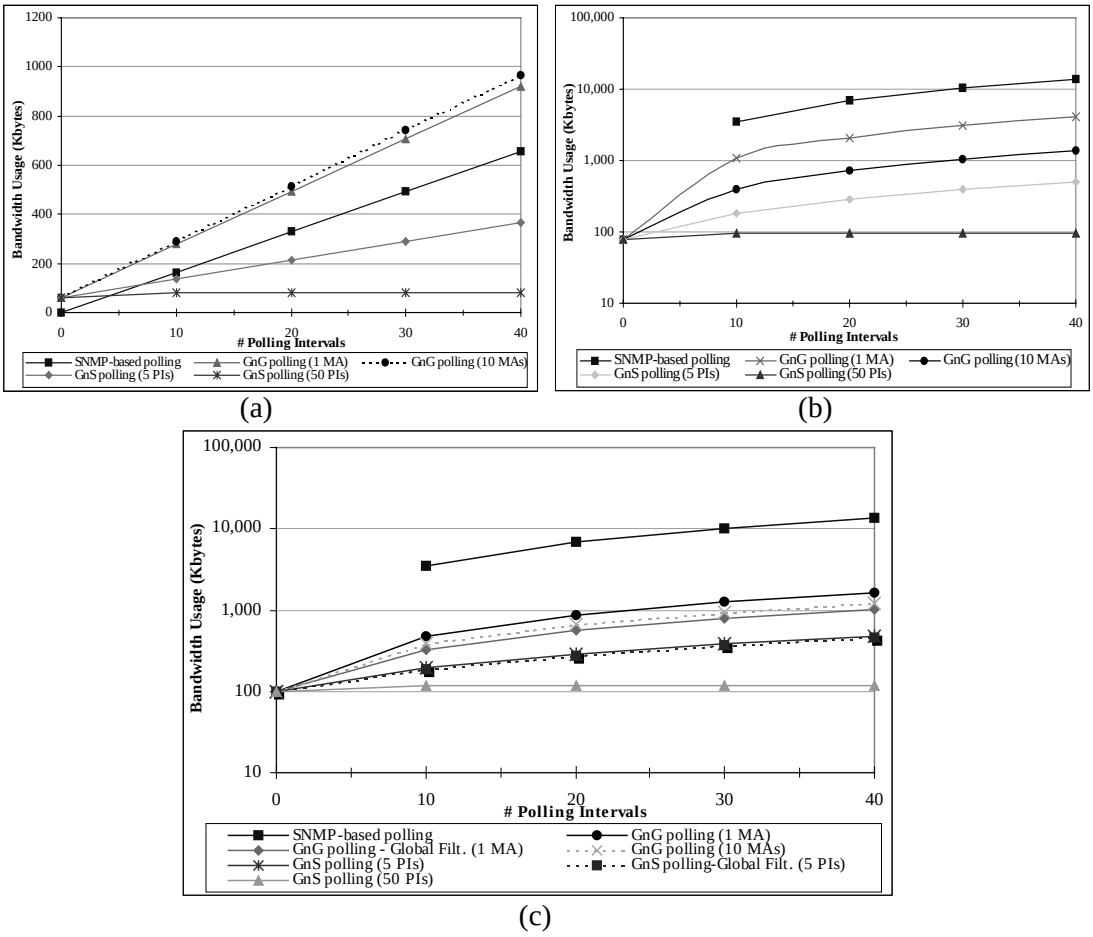

Figure 8: Bandwidth consumption of SNMP-based polling against MA-based proposed applications: (a) HF evaluation, (b) SNMP table polling, and (c) SNMP table filtering.

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 8.c). 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 (in this case, the most heavily loaded *network* interface is returned). The bandwidth usage is reduced even more when single table entries (instead of entire rows) are requested (this is not shown here).

6. Conclusions

We have presented three applications of intelligent/mobile agents on network monitoring. The applications, which have been built on the top of lightweight MAF described in previous work [5], address the scalability limitations of centralised NM that become

significantly more pronounced when transfers of bulk network monitoring data are considered. In particular, 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. Both real-time and off-line NM data acquisition is considered.

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

Ongoing work addresses:

- Assignment of higher priorities to MA threads to ensure faster execution of time critical management functions.
- Exploration of other NM areas where MA technology can be employed, such as network performance testing or software distribution.

References

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

[2] Baldi M., Gai S., Picco G.P., "Exploiting Code Mobility in Decentralised and Flexible Network Management", Proceedings of the 1st International Workshop on Mobile Agents (MA'97), pp. 13-26, 1997.

[3] Case J., Fedor M., Schoffstall M., Davin J., "A Simple Network Management Protocol (SNMP)", RFC 1157, 1990.

[4] Feridun M., Kasteleijn W., Krause J., "Distributed Management with Mobile Components", Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99), pp. 857-870, 1999.

[5] 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 International Conference on Communications (ICC'99), pp. 1362-1366, 1999.

[6] Gavalas D., Greenwood D., Ghanbari M., O'Mahony M., "Using Mobile Agents for Distributed Network Performance Management", accepted to the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99), 1999.

[7] Goldszmidt G., "On Distributed Systems Management", Proceedings of the 3rd IBM/CAS Conference, 1993, http://www.cs.columbia.edu/~german/papers.html.

[8] Goldszmidt G., "Network Management Views using Delegated Agents", Proceedings of the 6th IBM/CAS Conference, 1996, http://www.cs.columbia.edu/~german/papers.html.

[9] ISO/IEC 9596, Information Technology, Open Systems Interconnection, Common Management Information Protocol (CMIP) – Part 1: Specification, Geneva, Switzerland, 1991.

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

[11] Nicklisch J., Quittek J., Kind A., Arao S., "INCA: An Agent-Based Network Control Architecture", Proceedings of the 2nd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'98), LNCS vol. 1437, pp. 143-155, 1998.

[12] Sahai A., Morin C., "Towards Distributed and Dynamic Network Management", Proceedings of the IEEE/IFIP Network Operation and Management Symposium (NOMS'98), 1998.

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

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

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

[16] Susilo G., Bieszczad A., Pagurek B., "Infrastructure for Advanced Network Management based on Mobile Code", Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'98), pp. 322-333, 1998.

[17] Yemini Y., Goldszmidt G., Yemini S., "Network Management by Delegation", Proceedings of the 2nd International Symposium on Integrated Network Management, pp. 95-107, 1991.

[18] Zhang D., Zorn W., “Developing Network Management Applications in an Application-Oriented Way Using Mobile Agent”, Computer Networks And ISDN Systems (30) 16-18 (1998) pp. 1551-1557.

List of Acronyms

ASN.1: Abstract Syntax Notation 1
CMIP: Common Management Information Protocol
GnG: Get ’n’ Go
GnS: Go ’n’ Stay
HF: Health Function
IETF: Internet Engineering Task Force
MA: Mobile Agent
MAF: Mobile Agent Framework
MAG: Mobile Agent Generator
MAS: Mobile Agent Server
MbD: Management by Delegation
MIB: Management Information Base
MIT: Management Information Tree
NE: Network Element
NM: Network Management
OID: Object Identifier
PI: Polling Interval
RMON: Remote Monitoring
SNMP: Simple Network Management Protocol
UI: User Interface
VDL: View Definition Language

Damianos Gavalas received his BSc degree in Informatics (Computer Science) from University of Athens, Greece, in 1995 and his MSc degree in telecommunications from University of Essex, U.K., in 1997. He is currently pursuing a Ph.D. in electronics engineering at the University of Essex. His research interests include distributed computing, mobile agents, network and systems management. He is a student member of the IEEE Computer and Communication societies.

Dominic Greenwood was born in Llanidloes, Wales in 1970. He received the Ph.D. degree from Staffordshire University in 1997, specialising in the phase characterisation and auto-adaptive compensation of non-linear effects in network traffic. Since February 1997, he has been working for Fujitsu Telecommunications Europe Ltd., conducting research into the application of agent technology to distributed network management. He will shortly be joining Fujitsu Laboratories of America to continue work in this field. He has filed one international patent and has authored over 25 publications.

Mohammed Ghanbari received the BSc degree in electrical engineering from Aryamehr University of Technology, Tehran, Iran, in 1970, the MSc degree in telecommunications and the PhD in electronics engineering, both from University of Essex, U.K., in 1976 and 1979, respectively. After working for almost ten years in industry, he started his academic career in the Department of Electronic Systems Engineering, University of Essex, where he currently holds the position of the Professor. His research interests are video compression, video networking and performance management.

Mike O'Mahony received a Ph.D. from the University of Essex in 1977, for research into digital transmission systems. In 1978, he joined British Telecom working on research into fibre-optic systems, and in 1988 became the Head of the Inland Systems Section with overall responsibility for research into terrestrial systems and networks. In 1991, he rejoined the University of Essex as Professor of Communication Systems and Networks. He has published over 160 papers.