An Infrastructure for Distributed and Dynamic Network Management based on Mobile 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.
Tel: +44 (0)1206 872425 E-mail: {dgaval, ghan, mikej}@essex.ac.uk
[*]Fujitsu Telecommunications Europe Ltd.,
Northgate House, St. Peters Street, CO1 1HH, Colchester, U.K.
Tel: +44 (0)1206 363002 E-mail: D.Greenwood@ftel.co.uk

ABSTRACT

The use of Mobile Agent technology to distribute and delegate management tasks promises to overcome the scalability and flexibility limitations of the centralised Network Management paradigm. An efficient, lightweight infrastructure based on Mobile Agents is described, which addresses these issues by distributing management operations amongst managed devices and hence reducing processing load and bandwidth usage. A Mobile Agent Generator that facilitates the creation of new mobile agents for additional services is also presented. Our infrastructure is shown to outperform SNMP both in terms of polling response time and bandwidth consumption when considering data intensive operations.

1. INTRODUCTION

Most contemporary Network Management Systems (NMS) are characterised by poor scalability through being strongly rooted in the centralised, Client/Server model. In such systems, core management logic resides on a central station (manager) which 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 a processing bottleneck at the manager host. The de-facto standard SNMP for TCP/IP networks suffers from such disadvantages and additionally, architecture inflexibility as the functionality of both managing and managed parties is rigidly defined at design time. Security weaknesses of SNMP should be also highlighted.

A rational approach to overcome the inherent limitations of the centralised NMS is to distribute Network Management (NM) operations and move management intelligence as close as possible to the managed resources. A first approach addressing this issue has been Management by Delegation (MbD), proposed in [1] However, alternative approaches to distributed NM, based on Mobile Agent (MA) technology, have recently attracted considerable attention [3][4][5]. MAs introduce a new software communication paradigm that allows code migration between hosts for remote execution [2]. 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 [3].

In the MA architecture described in [4], MAs are launched from a manager, visit each network element (NE) sequentially and return collected data to the manager station. Another MA-based approach reported in [5] describes an architecture that makes use of mobile code to interact in a simulated network environment. However, neither [4] or [5] considers remote processing issues and, as a result, the manager host still suffers from a computational burden induced by processing bottlenecks. In addition, no authentication or encryption features have been integrated into [4], while only the former has been identified in [5].

Hence, in this paper we present an infrastructure that exploits the benefits of MAs to carry out semantically rich management operations in a highly scalable, flexible and efficient manner. A manager application has been developed to co-ordinate the policies of monitoring and controlling the NEs. In order to minimise the impact on network bandwidth, agent code migrates sequentially between managed devices and any necessary processing is performed locally. This alleviates the need for broadcast polling and results in a significant reduction in NM data at the source, as only directly specified information is returned to the manager.

The interface between the visiting agents and legacy management systems is achieved through Mobile Agent Servers (MAS). Several security issues have also been addressed including authentication of visiting MA instances and encryption of sensitive management information.

We also introduce a novel tool prototype, the Mobile Agent Generator (MAG), which creates MAs in response to service requirements. Such MAs may be generated post MAS initialisation to accomplish intelligent management tasks, tailored to the needs of a changing network environment. Thus, the MAG realises a flexible infrastructure through the creation of new MA instances, which *dynamically* extend NMS functionality.

The paper is organised as follows: Section 2 provides an overview and implementation details of our infrastructure. The suitability of the infrastructure when exposed to a data intensive NM operation is examined in Section 3. Experimental results are reported and discussed in Section 4, with Section 5 drawing conclusions and considerations for future work.

2. INFRASTRUCTURE OVERVIEW

The proposed infrastructure employs Java [6] due to its inherent platform independence, making it suitable for portability within distributed heterogeneous environments. This and other features, such as strong networking support suggest Java as a suitable platform for developing MA-based applications. Also, Java programs compile to bytecode, which executes within Java Virtual Environments, such as the Java Virtual Machine (JVM). Current trends [7] indicate that JVMs may soon be integrated into many network and computing resources.

The infrastructure, as illustrated in Figure 1, consists of the following major components:

1. The Manager, responsible for launching MAs and displaying returned results.
2. The MA code, capable of migrating between the managed entities to collect information based on pre-defined policies.
3. The MAS, capable of receiving MAs and providing an interface to the local physical resources.
4. The MAG, a factory that generates service-oriented MA objects.

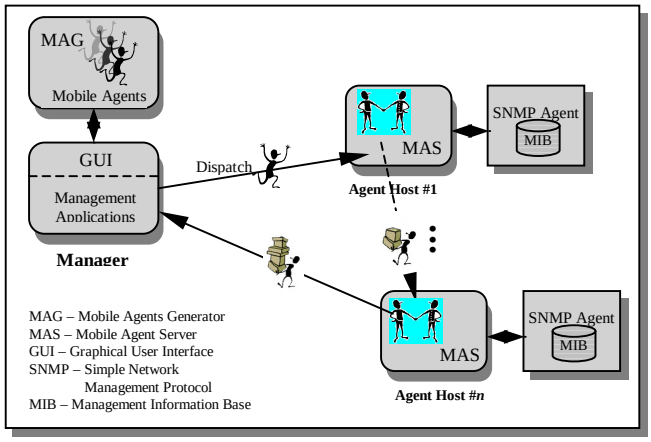

Figure 1: The Mobile Agents-based Infrastructure

2.1. MANAGER IMPLEMENTATION

The manager performs monitoring and control operations through its interaction with devices running agent processes. Active processes are 'discovered' through a broadcast poll at manager initialisation or whenever a new agent process starts operation. In the latter case, any active manager is notified and the host name appended to that manager's 'discovered' list.

In the event of a pending MA dispatch the manager assigns it an itinerary including all active agents hosts, unless a travel plan is manually specified. The MA's state information is then compressed (using the Java gzip utility) and transferred to the first destination host.

The manager application is equipped with a Graphical User Interface (GUI) consisting of a MIB browser plus 'problem', 'event' and 'results' panels. Currently, additional facilities allow:

- ✓ Polling of agents for specific object values;
- ✓ Initialisation of automated network discovery processes;
- ✓ Acquisition of on-line MIB variable descriptions;
- ✓ File logging of operational results.

2.2. MOBILE AGENT IMPLEMENTATION

An MA object is identified by its code (behavioural description), state information (modifiable variables) and attributes (static/permanent information). From our perspective, MAs are Java classes supplied with an itinerary table, a vector to store gathered data, the Object Identifier (OID) string(s) of requested object(s) and a number of methods that facilitate interaction with polled devices. These methods are called, for example, to obtain the OID string(s) or the next host to be visited, to update the data vector, etc. The MA code is minimised in order to reduce bandwidth requirements. Through the process of serialisation, the state of an MA object can be saved, transferred through the network and reconstructed (de-serialised) at the receiving node.

2.3. THE MAS: INTERFACE TO MANAGED RESOURCES

The interface between the visiting MAs and the legacy system is achieved through MAS modules (Figure 2), installed on every agent server. Functionally, the MASs reside above standard SNMP agents, defining an efficient run-time environment for receiving, instantiating, executing, and dispatching incoming MA objects, whilst protecting the system against malicious attacks. The SNMP agent process starts automatically at MAS initialisation and is 'killed' when the MAS application terminates.

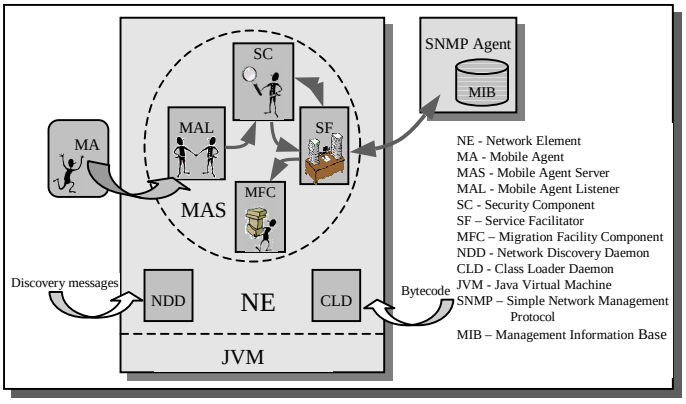

Figure 2: The Mobile Agent Server structure

A principal MAS component is the Mobile Agent Listener (MAL), a daemon listening on a well-known TCP port for incoming MAs. Upon arrival of an MA its code is decompressed and de-serialised (its state is loaded). The MAL then returns to listening mode and the MA object is passed to the Security Component (SC), which acts as the system's protective barrier. Specifically, the SC verifies the authenticity of the received MA through the use of security keys, ensuring that only trusted agents, dispatched by authorised hosts, are allowed instantiation. The RSA algorithm [9], based on the 'public-private pair of keys' paradigm, has been implemented providing both authentication and encryption features.

Upon successful authentication the MA is activated and provided a handle to the Service Facilitator (SF) component, which serves as an interface to the SNMP agent. The SF calls the MA method that returns the requested object OID string(s). The corresponding system information is then obtained through interaction with the SNMP agent and processed, if necessary, by an automatically invoked MA method. The value acquired,

either directly by the system or as a result of computation, is passed to the SC sub-system, encrypted and encapsulated into the MA's state. Ultimately, through the Migration Facility Component (MFC), the MA will be serialised and dispatched to the next host, or returned to the manager at the end of its operational travel plan. The life cycle of an MA object and its interaction with the MAS application is summarised in Figure 3.

Two additional threads run on each NE, outside the boundary of the MAS: the Network Discovery Daemon (NDD) that allows the manager to 'discover' active agent processes and the Class Loader Daemon (CLD), whose role will be discussed in Section 2.4.

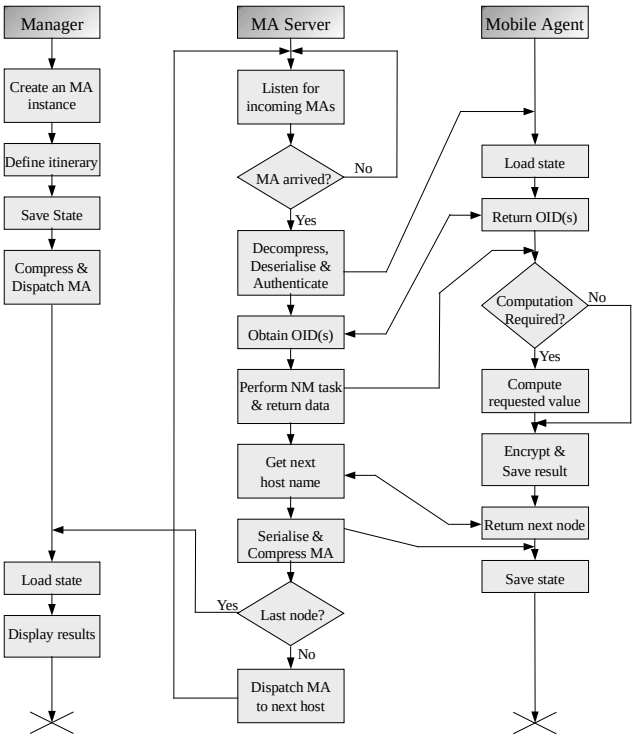

Figure 3: Flow diagram of a Mobile Agent's life cycle

2.4. MOBILE AGENT GENERATOR

The MAG (Figure 4), is essentially a factory for constructing customised MAs. In the context of this paper, generated MAs are designed to poll static management agents according to their operational function requirements. A GUI, dedicated to the MAG tool, allows the operator to:

- assign a name to the MA;
- define functional requirements;
- set the polling frequency to determine how often instances of the constructed MA will be launched;
- specify the classes of network devices to be polled.

Options for editing, deleting or updating an existing MA instance are also available.

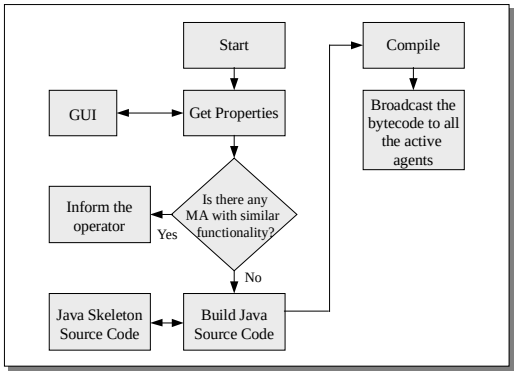

Figure 4: Mobile Agents Generator functional diagram

The MAG uses a skeleton Java source code with slots containing the MA's specified properties. The Java code created is then compiled and the generated Java bytecode compressed and transferred through TCP connections to all operating agent hosts. The MA's properties are compared, prior to its construction, against those of the existing MA classes to ensure that there is no other with the same functionality.

On the agent side, the CLD (see Figure 2) receives and decompresses the transmitted bytecode, validates the included Java class and stores it in a designated space. It should be emphasised that the transfer of the MA bytecode is performed only once, at MA construction time. From that point forward the transfer of persistent state, obtained from serialising the instance of the MA, is sufficient for the MAS entity to recognise the incoming MA and recover its state. In contrast, both [4] and [5] apply a policy that requires the transfer of both the MA's bytecode and persistent state, resulting in higher demand on network resources. To illustrate, for an MA with a 2.72Kb bytecode, its corresponding state information is only 260 bytes long (205 bytes compressed).

An alternative approach would be to keep a unique MA structure able to deal with arbitrary complex management operations. However, this would result in the unnecessary transfer of dynamically growing code, much of which may only be occasionally executed.

The MAG component ensures the framework remains sufficiently flexible by using run-time customisable MAs for specialised management tasks without the need for reconfiguration, re-installation or re-instantiation of either the manager or the agent applications. The MAG functionality can be easily extended in order to cover a wider range of management tasks.

3. MA OPERATIONAL FUNCTIONS

The applicability and performance of our infrastructure has been tested on a complex and data intensive management operation, i.e. polling. Compared to SNMP-based polling, which is inherently centralised and results in a flood of request/response messages (Figure 5), the use of MAs to collect and return the requested management information is shown to improve the NMS scalability and flexibility.

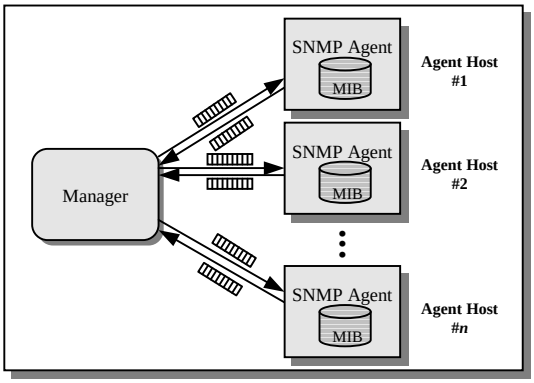

Figure 5: Centralised (SNMP-based) polling

Polling is a common operation in NM as there are often several object values that require constant monitoring. Cases occur, however, where one or two MIB variables are not a representative indicator of system state and a large number of them may require aggregation to provide a meaningful function (known as a *health function*) [10]. For example, the percentage *E(t)*, of the IP output datagrams discarded over the total number of datagrams transmitted in a specific time interval, is defined as:

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

Where *ipOutNoRoutes*, *ipFragFails*, *ipOutDiscards*, *ipOutRequests* and *ipForwDatagrams* are standard MIB-II objects [8]. A useful indication of the instantaneous network state would then be provided by the derivative:

$$E'(t) = \frac{E(t+\Delta t) - E(t)}{\Delta t} \quad (2)$$

Where Δt is the polling interval.

Thus, when centralised polling is employed, all the operating agents will receive five queries corresponding to the five object values appearing in (1). The value of *E(t)* is then computed by the manager when all object values have been returned. If on the other hand, MA-based polling is used, the health function calculation is performed at the local agent host, leading to a more balanced distribution of the computational burden. Moreover, a considerable compression of data volume is achieved at the source since a large number of observed operational variables is reduced to a single value. Also, the MAS entity is unaware of the health function formulation or any computational details, thereby increasing framework flexibility.

However, the two approaches also differ in terms of total response time. In the general case that *k* variables are required, the overall time for centralised polling, will be:

$$T_{centr} = k*(\max [2(t_{del}+*t_p)+ t_s])+nt_{comp} \quad (3)$$

While for distributed polling:

$$T_{distr} = (n+1)*(t_{del} + t_{s/d})+knt_s+nt_{comp} \quad (4)$$

Where *n* is the number of agents being polled, t_{del} is the average network latency between the manager and the agent or any pair of agents, t_p the processing time required for a simple request, $t_{s/d}$ the time taken to serialise/de-serialise an MA, t_s the time needed to access the system resources, and t_{comp} the *E(t)* function computation time.

As indicated by (3), T_{centr} is defined as the maximum time taken to obtain *k* operational values from any of the polled devices, plus the time required to compute *n* health function results. T_{distr} is defined as the total transition time to visit each device and return to the manager (hence *n*+1), plus the time needed to access and process the management information locally. It is noted that T_{centr} and T_{distr} scale linearly in relation to *k* and (*n*+1), respectively.

4. RESULTS

I) Response Time

The proposed infrastructure has been tested on a network comprised of Solaris and WinNT machines and compared to the conventional SNMP model. AdventNet's implementation of SNMP in Java [11] has been used as a comparison measure. Measurements of response time required to acquire a health function value have been recorded, as a function of (i) MIB variables, for a fixed number of polled devices (Figure 6a), and (ii) Managed devices, for a constant number of MIB objects (Figure 6b).

Figure 6(a) demonstrates that MA-based polling has superior performance in a limited size network, when health functions combining a large number of objects are employed. This is a key issue when considering time critical management operations. On the other hand, Figure 6(b) clearly depicts that the response time for MA-based polling increases linearly, see Eqn. (4), as opposed to SNMP-based polling response time that stabilises as the number of polled devices increases.

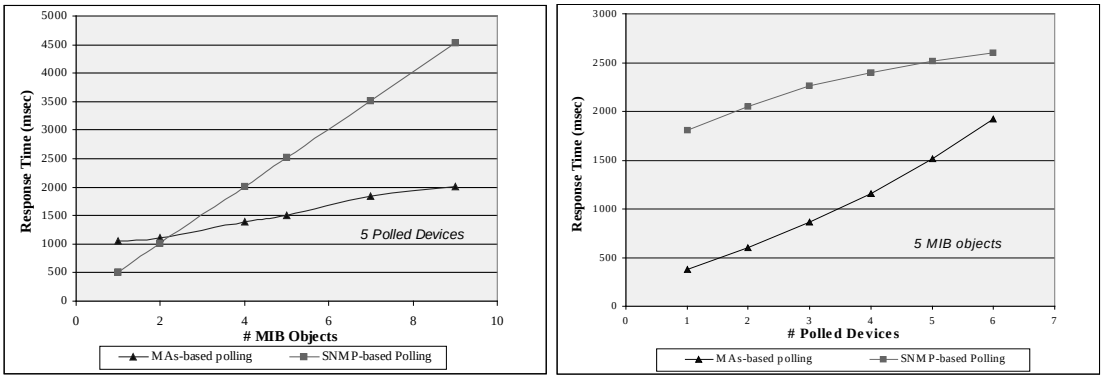

Figure 6: Response times as a function of: (a) MIB objects (for 5 polled devices), (b) Polled devices (for 5 MIB objects).

Thus, there is an upper bound on the number of devices that can be polled whilst maintaining lower response times over the SNMP-based polling scheme. However, the infrastructure's scalability can easily be improved by introducing a partitioned NMS structure with the managed network segmented into several domains, each with dedicated MAs.

II) Network Traffic Overhead

With the traditional centralised approach, assuming an average request/response size of 100 bytes including data and header information, and that polling of n devices for k operational variables is applied, the transmitted data would be $200*n*k$ bytes per polling interval. For MA-based polling the size of the compressed MA state information is on average 205 bytes (for a bytecode of 2.72 Kb), resulting in an overhead of $205*(n+1)$ bytes per polling interval, as the k variables are aggregated into one.

Thus, when large networks (as $n>>$) are considered, the management data are, in approximation, reduced by a factor of k. In addition the traffic is more evenly distributed and not concentrated at the manager host.

5. CONCLUSIONS

The design and implementation of an infrastructure that exploits the capabilities of MAs in distributed NM has been described. It has been shown that the use of MA objects leads to the sharing of workload between the manager and the agent hosts, thereby reducing bandwidth usage by applying remote data aggregation methods. Security issues have also been addressed through the implementation of the RSA algorithm to provide MA authentication and data encryption. A novel tool that dynamically extends NMS functionality by constructing lightweight MAs, able to carry out specified management tasks has also been introduced.

The functional appropriateness of the presented architecture has been demonstrated with data intensive operations. Empirical results indicate a significant improvement in both response time and bandwidth consumption when compared to the centralised paradigm.

Current work addresses:

- Optimisation of MAs itinerary to minimise polling response time.
- Extensions to MAG functionality, such as constructing MAs able to filter SNMP tables.
- Enhancements to all security aspects of MAs.

REFERENCES

[1] Yemini Y., Goldszmidt G., Yemini S., "Network Management by Delegation", Proceedings of the 2nd International Symposium on Integrated Network Management, April 1991.

[2] Nwana H., Ndumu D., "Introduction to Agent Technology", BT Technology Journal, Vol. 14, No 4, pp. 55-67, 1996.

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

[4] Ku H., Luderer G., Subbiah B., "An Intelligent Mobile Agent Framework for Distributed Network Management", Proceedings of the IEEE Global Telecommunications Conference (Globecom '97), pp. 160-164, 1997.

[5] Susilo G., Bieszczad A. and 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.

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

[7] Sun Microsystems, URL: http://www.javasoft.com/products/jini.

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

[9] Rivest R.L., Shamir A., Adleman L., "A Method for obtaining Digital Signatures and Public-Key Cryptosystems", Communication of the ACM, 21(2), Feb. 1978.

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

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