Running Mobile Agent Code over Simulated Inter-networks: an Extra Gear towards Distributed System Evaluation*

ANTONIO LIOTTA, CARMELO RAGUSA, GEORGE PAVLOU
Center for Communication Systems Research, School of Electronics, Computing and Mathematics
University of Surrey
Guildford, Surrey, GU2 7XH, UK
UNITED KINGDOM
{A.Liotta, C.Ragusa, G.Pavlou}@eim.surrey.ac.uk http://www.ee.surrey.ac.uk/CCSR/Networks/

Abstract: - Mobile Agent (MA) systems are complex software entities whose behavior, performance and effectiveness cannot always be anticipated by the designer. Their evaluation often presents various aspects that require a careful, methodological approach as well as the adoption of suitable tools, needed to identify critical overheads that may impact the overall system performance, stability, validity and scalability. In this paper, we propose a novel approach to evaluating complex mobile agent systems based on a hybrid framework which allows the execution of prototype agent code over simulated internet-works. In this way it is possible to realize arbitrarily complex MA systems and evaluate them over arbitrarily complex inter-networks, relying on full support to physical, link, network and transport layers for fixed and mobile networks. We illustrate the potential of our approach through an example agent system which we have prototyped and assessed over large-scale IP networks.

Key-Words: - Mobile Agent Systems evaluation; mobile agents simulation; large-scale systems; verification and validation of distributed systems; network partitioning algorithm.

1 Introduction

Mobile Agents (MAs) are computational entities that act on behalf of some other software entity, exhibit some degree of autonomy and are particularly featured with migration capability – i.e. they can roam the network and execute in those nodes that can host them. Other properties of an MA include re-activeness, pro-activeness, adaptability and cloning capability. In particular, cloning is the ability of an agent to create and dispatch copies, or 'clones', of itself.

Because of these properties, MA systems are complex distributed software entities whose behavior, performance and effectiveness cannot always be anticipated by the designer. Their evaluation often presents various aspects that require a careful, methodological approach as well as the adoption of suitable tools. A first important question facing the designer is whether or not the MA approach is suitable to address the given system requirements. This can be assessed by looking at architectural aspects but also requires the identification of critical overheads that may impact the overall system performance, stability and validity. Once the MA approach has been found to suit the system, three alternative roads may be pursued towards system realization and evaluation: prototyping, simulation, or a mix of them.

Each of these approaches contributes to unveil different, often complementary, aspects of the MA system. Prototyping has the main advantage of demonstrating the feasibility of the proposed approach as well as providing useful hints for improving the final design/implementation. Measurements of overheads such as delays, code migration time, load incurred by MAs and so forth may be vital to fine tuning the system design and to anticipate the performance of the agent system that is later to be deployed in the large scale. The extent to which this evaluation can be carried out is however limited by the scale of the experimental test-bed.

On the other hand, simulations do provide a better means to assess performance and scalability and have the additional advantage of allowing the assessment of other important aspects such as the correctness, validity, robustness, and stability of the MA system under consideration. Simulations alone, however, often fail to capture important aspects of the real MA system since they must rely on simplified modeling aimed at containing simulation time and trace data.

* The work reported in this paper has formed part of the Software Based System work area of the Core 2 Research Program of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, www.mobilevce.co.uk.

A more effective MA evaluation method will, thus, rely on a mixed experimental and simulation-based approach which inherits the intrinsic benefits of both providing the necessary insight into the MA system. Such an approach requires the adoption of a suitable 'hybrid' framework which supports the prototyping of agent code through specified Application Programming Interfaces (APIs), as well as facilitating the agent system evaluation via computer simulation.

The current approach to MA systems simulation is to create a simplified model of the system and realize software which simulates its behavior. In contrast, what we have done is to design a hybrid simulation framework which is generic – i.e. decoupled from any specific MA system – and addresses the requirements of complex, dynamic, large-scale MA systems. Our system is build on top of the *NS* network simulator [1] which supports the most common physical, link, network, transport and application layer protocols for fixed and mobile communication networks. Having extended *NS* with support for MAs – including agent execution over 'simulated' network nodes, agent migration, agent cloning, agent destruction and so forth – we have created an environment in which an arbitrarily complex MA system can be run over arbitrarily complex inter-networks. This can be used to evaluate the various aspects of agent systems under a variety of conditions such as network topology, size, communication protocols, etc.

After describing the *NS* network simulator and its features, we provide the details of the MA extensions and APIs designed to support the evaluation of agent systems. We illustrate the proposed methodology for MA system evaluation through a simple case study which we have prototyped and assessed over large-scale IP inter-networks. We finally draw the lessons learned from experimenting with our framework in order to provide hints to those who would like to apply it for the evaluation of other MA-based distributed systems.

2 Background: the *NS* Network Simulator

NS is a network, discrete event simulator that runs in a non-real-time fashion and supports the most common physical, link, network, transport and application layer protocols for fixed and mobile communication networks. Over the last few years, its extensive use in the networking research community has contributed to make it a valuable tool for studying, improving, and introducing new protocols.

In addition to fixed networking, *NS* supports wireless and satellite networking, unicast and multicast routing, centralized and hierarchical routing, static and dynamic routing. Extensions to the core *NS* have been provided by researchers worldwide, including support to GPRS, Mobile IPv6, BlueTooth, RSVP, Differentiated Services, MPLS, active networking, IEEE 802.11 for WLAN, multi-hop wireless ad-hoc networking, and Cellular IP (the interested reader may refer to [1] for software and documentation).

NS allows characterizing point-to-point bi-directional links through bandwidth, delay, and queue type. It also allows modeling packet scheduling (i.e. the decision process used to choose which packet should be serviced or dropped) and buffer management (i.e. any particular discipline used to regulate the occupancy of a particular queue). *NS* includes support for several algorithms such as drop-tail (FIFO) queuing, RED buffer management, and different variants of fair queuing. There is also literature on the experimentation of various queuing disciplines for Quality of Service management, active queue management, and stochastic queue management (refer to [1] for documentation).

In addition to the extensive networking support, as summarized above, *NS* enjoys the advantages of an open software tool (the source code is freely available for experimentation) and of an architecture that can be easily extended (Fig.1). The user specifies the simulation scenarios in OTcl, an object oriented extension to Tcl. These are interpreted in the *NS* Kernel that includes the abovementioned networking functionality. For increased efficiency the *NS* core functionality is implemented and executed in C++. In order to ease the use of the simulator, however, most C++ classes are mirrored into OTcl and can be directly invoked by OTcl scripts. The simulator functionality may be simply enhanced by creating new C++ classes, adding new functions to the *NS* library, and mirroring the new code into OTcl classes.

Two important tools come with *NS*, the Network Animator (NAM) [2] and the GT-ITM topology generator [3]. NAM generates graphical animations of trace files generated by *NS* simulations including the relevant network events. The tool is extremely useful for verifying the correctness of the simulations.

Finally, the topology generator is used to automate the process of generating networks for the purpose of assessing the protocols and systems

under scrutiny. GT-ITM generates realistic Internet-like topologies as well as random topologies following a well established methodology [4, 5].

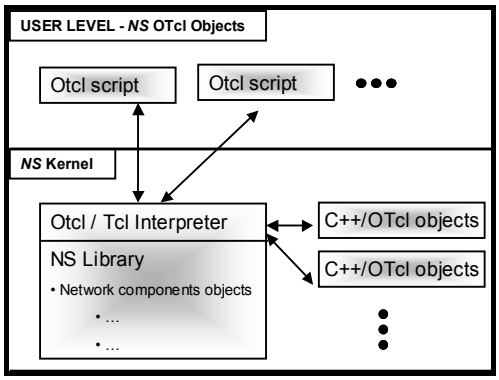

Figure 1. *NS* functional architecture.

Although it provides such extensive support to layers 1 to 3 networking, *NS* is limited in its capability to support application-level simulations. It provides simple traffic generators which model the behavior of applications such as http, ftp, telnet and web applications. However, there is no support to generic applications and particularly to distributed applications and systems. We focus below on our approach to extending *NS* with a virtual execution environment which allows the execution of arbitrary distributed applications based on MAs over simulated networks.

3 MA Extensions for *NS*

The extended *NS* functional architecture with support to MA systems and applications is depicted in Fig.2 in which the MA extensions are highlighted in bold. At *NS* kernel level, we have added a new `MAgent` class as a subclass of the *NS* `Agent` class and mirrored it into an equivalent OTcl class. `MAgent` has all the necessary methods to create, run, stop, clone, migrate and destroy an MA.

These methods are exposed at user level as an Application Programming Interface (API) which is used to write the necessary MA code using the OTcl scripting language. In this way, one can easily write a script which creates one or more agents, each of which incorporates a specific task, building an MA distributed system. Agent creation involves the instantiation and initialization of an MA object and its association to a network node.

The agent code is executed in the MA virtual execution environment that is intertwined with the network simulation environment through the following stratagem, as shown in Fig.3. Agents are allowed to execute their code in real time as far as they do not involve any communication with other distributed entities or agents.

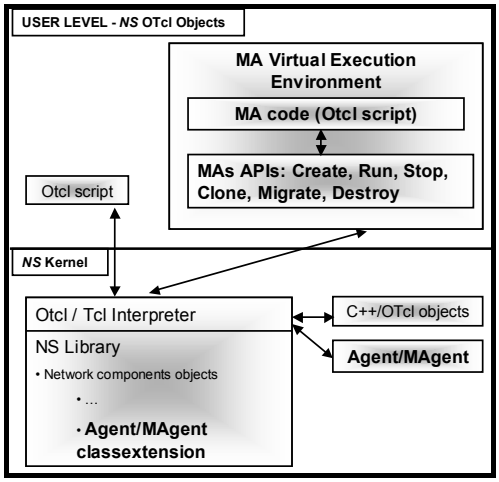

Figure 2. Functional Architecture of *NS* with MA extensions.

In other words agents can process data that is available locally and take decisions that do not incur any network load. However, as soon as agent processing involves network resources – e.g. the agent needs to send or receive packets or wishes to migrate to a different node – the relevant network events are triggered into the network simulator that handles them accordingly.

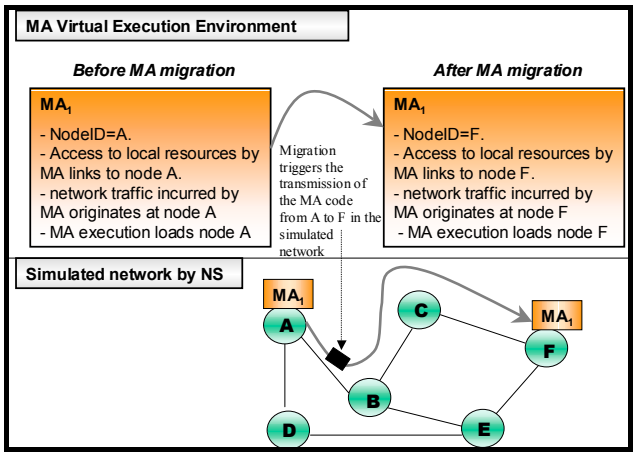

Figure 3. MA execution over a simulated network.

Fig.3 depicts an example of how agent migration is handled by our hybrid simulation environment. As the agent resides at node *A*, the local resources accessible by the agent are those of node *A*. For instance, if the agent task involves processing of the local routing tables, the agent will read and process the routing tables stored at *A*. If the agent wishes to migrate to node *F*, agent execution is suspended and the agent code is serialized for transmission. At this point a packet having size equal to the serialized

agent is scheduled for transmission from *A* to *F* – i.e. a network event is added to the event queue of the simulator. This event is handled by the simulator according to the networking protocols set up by the OTcl script – i.e. layers 1 to 3 protocols.

Finally, as soon as the packet is received by node F an appropriate `packet_received` event is generated by the simulator. This event triggers the MA de-serialization, instantiates the MA in the virtual execution environment, setting up the `NodeID` parameter to *F*. This means that when the agent execution is resumed, its local resources will be those of node *F*. Further details of the proposed MA simulation environment are given in the section below through a case study which we have prototyped and assessed by simulation.

4 Case Study: Near-optimal Network Partitioning based on MAs

The network-partitioning problem can be formulated as follows. Given a distributed application, we need to determine the number of partitions, the nodes forming those partitions, and the center of each partition that minimize the overall traffic and/or response time of the distributed application.

This problem is fundamental to a number of applications, including not only networking problems such as the one of optimally placing *p* servers in a network of *N* nodes but also in the area of transportation theory. The former, studies the optimal location of emergency facilities such as fire stations which results in minimal intervention time. It also deals with the optimal planning of transport infrastructures such as railways, roads and so on.

Because of its importance, network partitioning has been extensively studied since the early '70s under the banners of *p*-center and *p*-median problems, which have both been proved *NP*-complete when striving for optimality [6]. Approximate polynomial algorithms have been proposed but none of them suites the combined requirement of scalability, optimality, and efficiency. Those algorithms are centralized, requiring the network distance matrix – i.e. knowledge of the whole network topology – at a single computational point where the partitioning algorithm is executed. This constraint represents a major impediment at these days in which networks have assumed a worldwide scale, are not under the control of a single operator, and have a highly dynamic nature.

We proposed herein a distributed algorithm which makes use of MAs to find a provably near-optimal solution to the network partitioning problem in linear time. In addition, our approach does not require any knowledge of the network topology and does not rely on the network distance matrix. It assumes, instead, that MAs have a 'read-only' access to the routing tables of the local router.

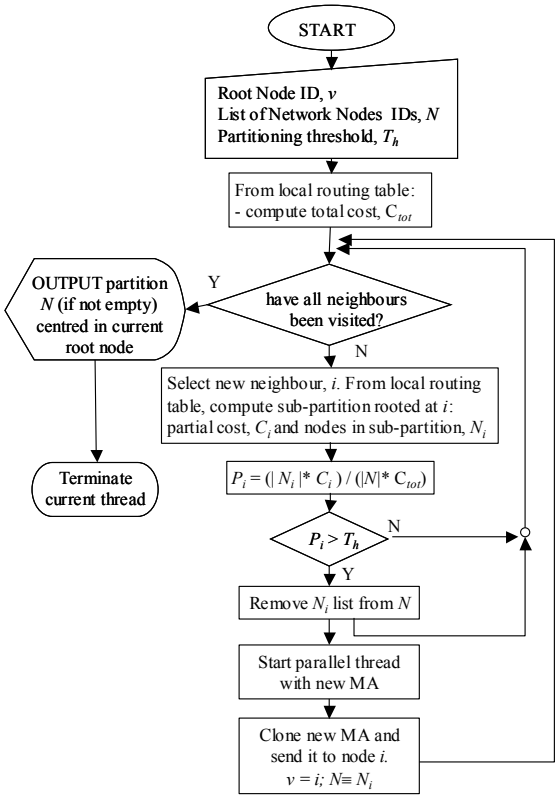

Figure 4. Flow-chart diagram of network partitioning algorithm.

Our algorithm is depicted in the flow-chart diagram of Fig.4. Because the focus of this paper is on the simulation environment rather than on the actual algorithm and due to space restrictions we do not describe the internals of the algorithm in greater detail. However, the interested reader may refer to [7] for a thorough explanation and a comprehensive analysis.

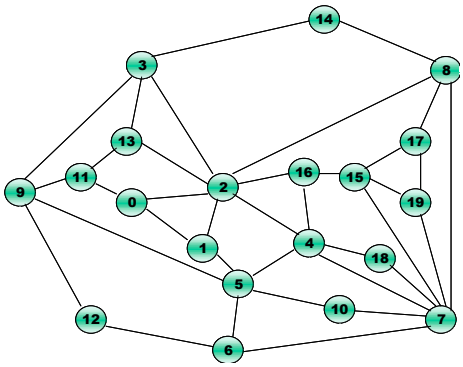

Figure 5. Example network topology.

The algorithm can be illustrated through the example topology depicted in Fig.5. Assuming node 0 as the initial root node, the algorithm starts off by creating an MA at this node (initially, the list of nodes includes all the nodes). The agent computes the total cost, C_{tot} required to reach all the nodes from the local routing table (i.e. the one stored at node 0). Similarly, it computes the partial costs associated to each of the neighbor nodes (nodes 1, 2, and 11) and compiles the lists of nodes reachable through each of the neighbors (N_1, N_2 ,N_{11}).

In other words the agent starts building the distribution tree shown in Fig. 6, with associated partial costs. The agent computes then a probability function, P_i for each neighbor to make a decision as to how to build an initial sub-partition. This process results in establishing two partitions, {1, 5, 6, 9, 10, 11, 12} and {2, 3, 4, 7, 8, 13, 14, 15, 16, 17, 18, 19}, and in cloning a new agent for the former one. The new MA is rooted at node 2 and starts a parallel thread which repeats the whole process and ends up further sub-partitioning the network as in Fig.6.

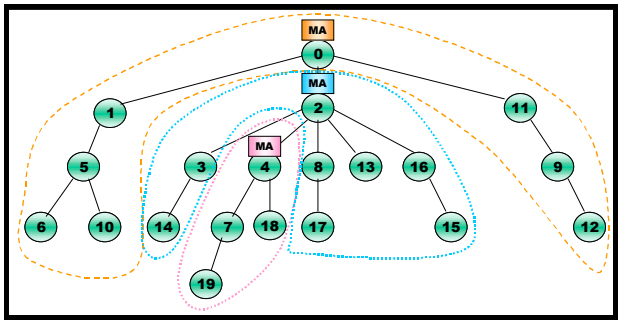

Figure 6. Network distribution tree and final partitions.

The above algorithm has been evaluated through simulation using our hybrid environment. Each agent incorporates the same partitioning algorithm and executes in the virtual execution environment with links to the node where the agent actually sits. The agent code is parametric with respect to the routing tables – i.e. it can only access the local routing table – and list of nodes in its sub-partition. The output of the algorithm is incorporated in the final agents. Each agent contains a disjoint list of nodes (the sub-partition) while the agent node represents the center of the sub-partition.

Having implemented the MA-based network-partitioning algorithm it was possible to run it under a variety of network conditions for the purposes of validation and assessment. A comprehensive evaluation of the algorithm's *scalability* with respect to number of network nodes and network diameter is reported in [7] which applies the algorithm to configure a distributed monitoring system. *Performance* parameters include the *traffic* incurred by the monitoring system and its *response* time. Both of them have been found to increase linearly with network size, an interesting result that confirms the scalability of the proposed algorithm.

Simulations were also essential to study the *optimality* of our algorithm. The algorithm is optimal if it computes a set of centers (i.e. the *p*-centers) that minimizes the total distance between each center and the nodes in its respective partition. Optimality has been studied by measuring two important distance metrics, "total hop distance" and "maximum weighted distance" under different network conditions (topology, size, average node degree, etc) and by comparing the results with those achieved with a provable near-optimal algorithm – i.e. the *Lagrangian* algorithm [6]. It should be mentioned that the former algorithm, despite providing near-optimal partitions, is not scalable because it is centralized and relies on the network distance matrix.

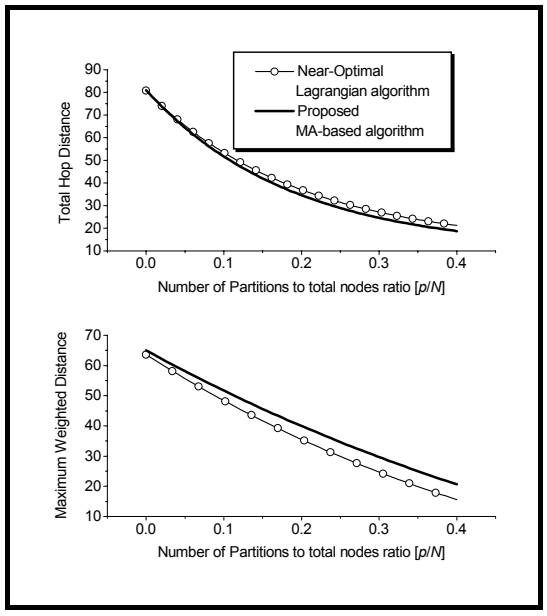

Figure 7. Simulation results.

Simulation results are depicted in Fig.7 which proves the near-optimality of our algorithm with regard to total hop distance (the MA curve is under the *Lagrangian* curve which is near-optimal). Maximum weighted distance is not provably near-optimal but is, however, relatively close to it.

5 Applicability of MA Simulator

Sections 3 and 4 suggest an evaluation methodology which does not significantly differ from the typical simulation-based system performance evaluation

described, for instance, in [8] (part V, p.391-504). Simulations design involves the appropriate choice of *metrics*, *parameters*, *factors*, and *workload*. It entails: the *validation* of the simulation model against the real system; the *repetition* of simulations over randomly generated network topologies having similar topological features; the performance of *continuity*, *degeneracy*, and *consistency* tests; and the *statistical analysis* of simulation traces.

Our simulation environment poses, however, a peculiar issue arising from its hybrid nature. In fact, it allows running the agent system in real time while network events are simulated. This generates the paradox of a system where data processing happens virtually instantaneously – i.e. the real-time clock – whereas network-bound events progress at the pace of the simulator – i.e. the simulator clock. Our MA virtual execution environment provides a loose synchronization between MA processing and network events by pausing MA execution during the generation of network events (Fig.3).

This approach suits those MA distributed systems whose processing is not tightly bound to networking events, such as the case study presented above. Each MA bases its cloning/partitioning decision based on the local routing table which is not assumed to change during MA processing. This assumption is reasonable because MA processing time is orders of magnitude smaller than that of networking events.

More generally, the MA simulator successfully captures the behavior of MA systems in which MA processing is relatively fast with respect to network events or is loosely coupled with them. We are currently working towards releasing this constraint in order to extend the applicability of the MA simulator to systems that are tightly coupled with networking events.

6 Conclusions

As MA distributed systems assume an increasingly important role in diverse areas of communications, management and service provisioning, tools for their evaluation become important. Simulation-based assessment has traditionally been limited to *ad hoc* implementations, often aimed at capturing particular aspects of the system. In this article, we have described a general-purpose MA simulation environment which allows the realization of MA code that can run over simulated networks. Our main design requirement was the ability to assess performance and scalability of complex MA systems over realistic networking environments. For this reason we have based our system on the *NS* network simulator, a well established environment which can handle networks of the order of thousand of nodes and supports the most common networking protocols.

We have realized, however, a prototype that can be used to validate new MA systems, study their behavior under a variety of conditions, and facilitate software re-use. We have illustrated our hybrid approach via a case study in which the MA code is only loosely coupled with the network simulator and can be easily ported to a real MA system.

Clearly, more work is needed to refine the MA models, add new MA functionality, and improve the synchronization between real-time and simulator clocks. It is through the lessons learned from new, additional cases studies that the limits of the simulator will be identified and stretched even further.

Acknowledgments:
The work reported in this paper has formed part of the WA1 area of the Core 2 Research Programme of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, www.mobilevce.com, whose funding support, including that of EPSRC, is gratefully acknowledged. More detailed technical reports on this research are available to Industrial Members of Mobile VCE.

References:

[1] K. Fall, K. Varadhan, *The NS Manual*, UC Berkeley/LBNL, (www.isi.edu/nsnam/ns/).

[2] J. Mehringer, *The Network Animator* (NAM), University of Southern California, Information Sciences Institute (www.isi.edu/nsnam/nam/).

[3] Ellen Zegura, *The GT-ITM Topology Generator*, (www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html).

[4] E.W. Zegura, K.L. Calvert, M.J. Donahoo, A Quantitative Comparison of Graph-based Models for Internet Topology, *IEEE/ACM Transactions on Networking*, 1997.

[5] K.L. Calvert, M.B. Doar, E.W. Zegura, Modeling Internet Topology, *IEEE Comm. Mag.*, June, 1997.

[6] M.S. Daskin, *Network and Discrete Location*, Wiley, 1995.

[7] A. Liotta, *Towards Flexible and Scalable Distributed Monitoring with Mobile Agents*, PhD Thesis, Dept. of Computer Science, University College London, London, UK, 2001.

[8] R. Jain, *The Art of Computer Systems Performance Analysis*, Wiley, 1991.