

Agent Transport Simulation for Dynamic Peer-to-Peer Networks

Evan A. Sultanik
Max D. Peysakhov
and
William C. Regli

Technical Report DU-CS-04-02
Department of Computer Science
Drexel University
Philadelphia, PA 19104
September 2004

Agent Transport Simulation for Dynamic Peer-to-Peer Networks

Evan A. Sultanik Max D. Peysakhov William C. Regli
eas28@cs.drexel.edu umpeysak@cs.drexel.edu regli@drexel.edu

Abstract

This paper introduces *MATES* (the Macro Agent Transport Event-based Simulator). *MATES* is intended for efficient comparison of application layer agent algorithms. We further motivate the need for such simulator as well as explain its architecture. We also present the implementation details of *MATES*. This paper is then concluded by presenting *MATES*' limitations, future work, and conclusions.

1 Introduction

In developing the Secure Wireless Agent Testbed, *SWAT* [6], we have experienced firsthand numerous problems that arise from working with agents on live wireless networks. Hardware failures and system misconfigurations can cost valuable research time. Although producing agent based technologies for a fully functional wireless network has always been our goal, we have recognized the need for a more controlled simulated environment in which to compare algorithms and perform more controlled experiments.

To address this need we have developed *MATES*, the Macro Agent Transport Event-based Simulator — an application layer simulator created to investigate the behavior of distributed agent based systems over mobile ad hoc networks (MANETs). Given such a goal, *MATES* was implemented *not* to simulate any specific agent framework, but rather to provide a generic, easily expandable environment for agent-based system testing.

The rest of this paper is organized as follows: we present our motivation for developing *MATES* in section 2, followed by the formal description of the simulator design in section 3. Section 4 presents all of the implementation details for *MATES*. We present the limitations of the current implementation, as well as mention possible improvements in section 6. Section 7 concludes this paper.

2 Motivation

Various network simulators already exist, such as the now ubiquitous NS-2 [8]. However, the purpose of these simulators is to model real-world networks such that low-layer processes, such as routing algorithms, can be compared. Modeling a system at such a resolution has proven to create a fair amount of overhead. Therefore, *MATES* was created to approximate the low-layer processes to divert more resources to agent interaction. This concession of accuracy mitigates the problem of efficiency.

The purpose of *MATES* is primarily to model the conditions of dynamic, peer-to-peer networks such that algorithms for distributed computing in the *application layer* can be compared. While this is technically possible using simulators such as NS, doing so imposes extra configuration, computational complexity, and experimental “noise” that can skew algorithm comparison.

Another potential advantage of *MATES* is ease of use. One can easily implement a new set of tests by simply extending provided agents and overriding/modifying their functionality.

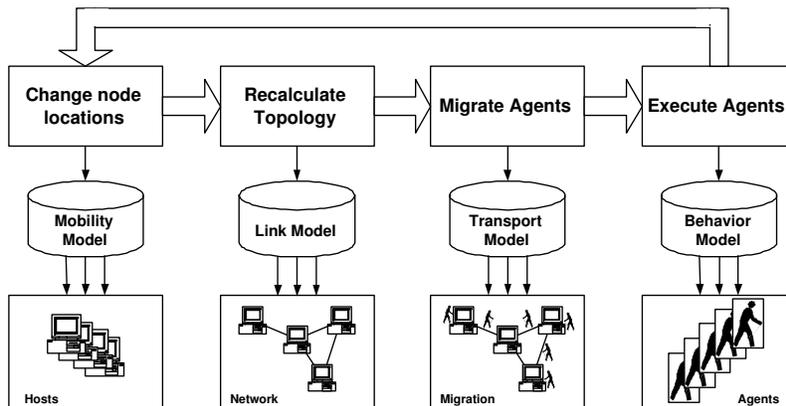


Figure 1: The simulator cycle.

3 Formalization

MATES is based upon four core models used for simulation: the *Host Mobility*, *Link Connectivity*, *Transport*, and *Agent Behavior*. Each model can be thought of as conglomerating and approximating an associated “block” of the OSI model [9]. During every cycle of the simulator each of the models is successively applied to the domain, as in Figure 3.

3.1 Iterations vs. Scheduling

Time in MATES is modeled using *iterations*, as opposed to “simulated time.” Therefore, events in MATES are not scheduled, per se. Instead, the simulator itself goes through iterations, during which each host may iterate each of its agents.

There are multiple reasons why this approach was taken. Firstly, it is assumed that the simulator will be used more for *comparing* algorithmic approaches in a controlled environment than for actually predicting real-world runtime. Second, it is not clear how CPU latency could be modeled using a scheduling paradigm. For example, consider simulating a mobile ad hoc network of PDAs. The bottleneck in this simulation might not be the network; instead it might be the devices’ CPUs. Therefore, scheduling an event to take place requires a priori knowledge of CPU usage. This problem is further complicated if the hosts in the network are non-homogeneous. With the iterative approach, each host can be given a (not necessarily constant) number of iterations that it may allot to each of its agents during each *simulator* iteration.

3.2 Hosts and Agents

MATES models two primary entities: *hosts* and *agents*. Unlike some other agent architectures such as EMAA [3], MATES does not provide a thread of execution for servers. Instead, static agents (i.e. agents that never migrate) can be used.

Let H be the set of hosts in a simulation, and let A be the set of agents. We define a function g that maps each host to a set of agents currently on that host:

$$g : h \in H \mapsto \{a \in A\}$$

Note that while an agent is in transit it does not exist on any hosts. Therefore:

$$\left(\sum_i^{|H|} |g(h_i)| \right) \leq |A|$$

Each agent can query its current host for the following percepts:

- Handles to the *other* agents at the current host;
- Geographic location of the current host; and
- Addresses of the neighbor hosts.

3.3 Host Mobility Model

Every host has a *mobility model* that dictates the way in which it moves. In every iteration of the simulation, each host's mobility model can dictate its location anywhere within the bounds of the simulated environment (which are a simulation parameter). In most cases, mobility models will move the host to a position adjacent to its current, however this is not a requirement.

3.4 Link Connectivity Model

The simulation itself has a *link connectivity model* that defines the conditions under which two hosts have a connection. This allows for simulation of both static and ad hoc networks. In the former case, the link connectivity model is basically a lookup table for connections. In the latter, link connectivity is a function of hosts' locations.

3.5 Data Transport Model

The simulation also has a *data transport model* that defines the amount of time required for an entity to be sent over a specific link. Here, "time" is actually referring to simulator iterations. This will usually be a function of the link quality and the size of the entity.

3.6 Routing

Hooks for routing are not provided in MATES; agents are assumed to route themselves (in the *Active Networking* paradigm [7]). However, it is possible to implement a routing protocol in the simulator. This can be accomplished by having a static agent on each host responsible for carrying out the routing tasks. When an agent needs to know the next hop in a route, it will query the host for a handle to the "routing agent," and query that agent for the route table.

4 Implementation

The current version of MATES has been implemented in Java. Java was chosen primarily for the ease of its class polymorphism. Also, many of the current leading agent architectures are implemented in Java [3, 2].

Our implementation of MATES has default host mobility, link connectivity, and data transport models, however each of these can be overridden.

The default host mobility model initializes each host with a random heading (0° to 360°). During each iteration, each host's heading is randomly chosen by either maintaining in its current heading (with probability 0.6), rotating 45° clockwise (with probability 0.2), or rotating 45° counter-clockwise (with probability 0.2). Hosts move forward one meter in their respective directions. When the host mobility model is determining a host's next movement, it is passed a handle to the link connectivity model. This enables the host mobility model to move hosts to preserve a certain topology. For example, in some instances one might never want the topology to be disconnected. Therefore, the host mobility model can predetermine if a specific host-repositioning will disconnect the network.

The default link connectivity model emulates the *exact connectivity* model for ad hoc network graph generation [1]; connections are determined by the Euclidean distance between hosts. Each host has a default radio range of 300 meters.

The default data transport model dictates that transit times are calculated with an inverse linear relationship to link quality. Therefore, agent transit times have an exponential relationship to the Euclidean

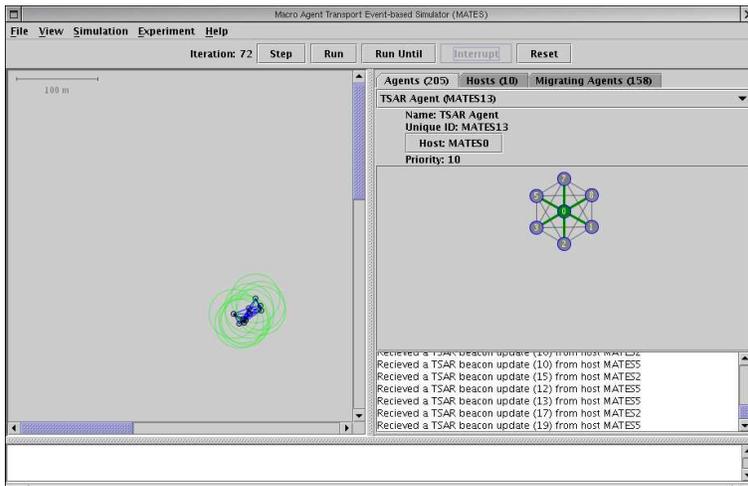


Figure 2: Screen of MATES, simulating a dynamic network of 10 hosts.

distance between hosts. By default, the maximum transit time is 1 iteration; all traffic takes the same amount of time to transmit. However, each agent may implement an interface such that it can dynamically define its own transit time.

A screen shot of our implementation of MATES is provided in Figure 2.

5 Example

One of the first ideas tested in the simulator was an ecology-inspired approach to agent balancing [5]. The goal was to balance the number of agents on the network according to the number of hosts.

The host mobility model for these experiments was identical for all hosts. Host locations were assigned in the beginning of the experiment and remained unchanged for the duration of the experiment. The link model assumed all connections between the nodes to be of constant, equal strength regardless of the nodes’ physical location.

The transport model for agents assumed that the time required to transmit the agent over the specific link, t , is $t \in [1, n]$. Delay is linearly distributed between 1 and n iterations corresponding to link with 100% and 0% quality, respectively. In this manner, an agent traveling from host h_1 to h_2 would be removed from host h_1 and then introduced on host h_2 t iterations later.

The control flow of an ecology-based agent’s behavioral model is shown in Figure 3. According to this control flow diagram, an agent first decreases its internal food bank by $f_a(t)$ for each second that elapsed since the last decrement. Then, the agent completes the task and collects all food points associated with that task. Based on its current food resources, the agent may decide to die or to reproduce. Lastly, the agent migrates to another random host looking for food.

MATES provided us with an environment to perform the initial set of experiments.

6 Limitations and Future Work

The major limitation of MATES, in its current implementation, is its iterative representation of time. This requires that all computation is atomic. Furthermore, all user-created simulation components (i.e. agents) must agree on a mapping from iterations to time.

Implementation of MATES with “simulated time,” as opposed to iterations, might be beneficial to the transparency of the simulator. This could be accomplished by using a system like Java in Simulated Time

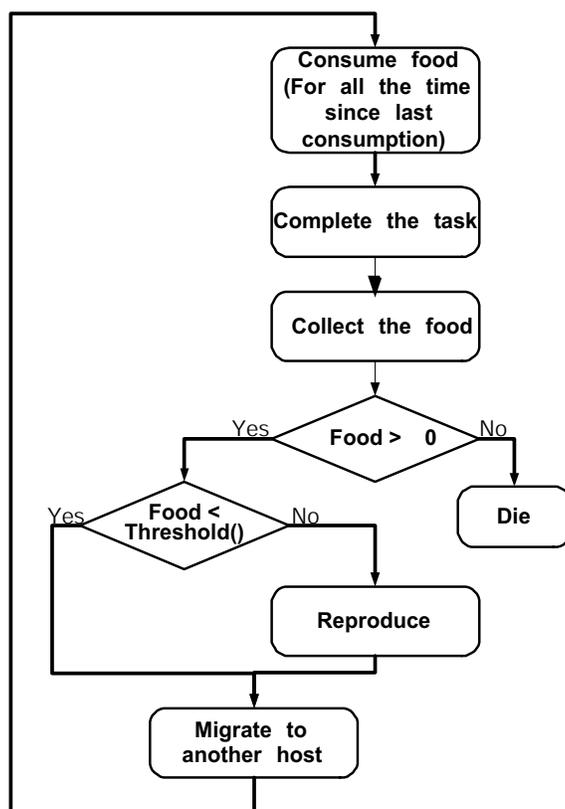


Figure 3: Behavior of the ecology-based agent.

(JiST) [1], however simulating computational bottlenecks on hosts of a non-homogeneous network might be unintuitive.

MATES was not designed for large networks, and as such does not make an effort to aggregate or parallelize homogenous entities. With that said, MATES has exponential computational complexity with respect to the number of agents and hosts, with memory usage scaling linearly.

7 Conclusions

Ad hoc networks and agency are both heavily researched areas of computer science. As these types of networks become more prevalent, efficient distributed algorithms will be required; the need for further research in this area has already been established [4]. Therefore, a system for testing and comparing these algorithms before deployment is a necessity. We believe MATES, or a system like it, is a step toward achieving this.

References

- [1] Christopher L. Barrett, Madhav V. Marathe, D. Charles Engelhart, and Anand Sivasubramaniam. Approximate connectivity graph generation in mobile ad hoc radio networks. In *Proceedings of the 36th Annual Simulation Symposium*, page 81. IEEE Computer Society, 2003.
- [2] BBN Technologies. Cougaar architecture document. <http://docs.cougaar.org>, February 2003.

- [3] R. P. Lentini, G. P. Rao, J. N. Thies, and J. Kay. Emaa: An extendable mobile agent architecture. In *AAAI Workshop on Software Tools for Developing Agents*, July 1998.
- [4] Nikos Migas, William J. Buchanan, and Kevin A. McCartney. Mobile agents for routing, topology discovery, and automatic network reconfiguration in ad-hoc networks. In *Proceedings of the 10th IEEE Conference and Workshop on the Engineering of Computer-Based Systems*, pages 200–206, 2003.
- [5] Max Peysakhov, Vincent A. Cicirello, and William C. Regli. Ecologically inspired agent control model. In *Proceedings of Formal Approaches to Agent-Based Systems III*, April 2004.
- [6] Evan Sultanik, Donovan Artz, Gustave Anderson, Moshe Kam, William Regli, Max Peysakhov, Jonathan Sevy, Nadya Belov, Nicholas Morizio, and Andrew Mroczkowski. Secure mobile agents on ad hoc wireless networks. In *The Fifteenth Innovative Applications of Artificial Intelligence Conference*. American Association for Artificial Intelligence, August 2003.
- [7] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, 1997.
- [8] UCB/USC/LBNL/VINT. Network simulator (NS) version 2. <http://www.isi.edu/nsnam/ns/>, February 2003.
- [9] H. Zimmerman. OSI reference model - the ISO model of architecture for open system interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980.