Comparing the Performance of Message Delivery Methods for Mobile Agents

Andrei Olaru(✉), Dragoş Petrescu, and Adina Magda Florea

University Politehnica of Bucharest, 060042 Bucharest, Romania
{andrei.olaru,adina.florea}@cs.upb.ro, dragos.petrescu@stud.acs.pub.ro

Abstract. Deploying a large number of mobile agents in scenarios where agents migrate frequently and/or exchange messages frequently requires methods for message delivery that are adequate to these specific situations. Deciding on which message delivery model to use, and whether a newly developed model is better than existing ones, may be difficult without an experimental testbed for comparison. This paper presents a framework for the comparison of message delivery models dedicated to mobile agent systems. The framework allows the generation of large, difficult scenarios, in which different methods may be evaluated side-by-side, revealing trade-offs between success rate, delivery time, and resource consumption. The architecture of the framework is designed to quickly integrate new models and to allow the direct deployment of a model implementation in real-life applications. As validation, we have integrated the implementation of several well-known delivery models and made comparisons between these models, from different points of view.

Keywords: Multi-agent systems · Mobile agents · Models and abstractions for MAS

1 Introduction

Mobile agents enable computation to be performed on a remote machine, by an autonomous entity that is able to travel between different hosts, thus avoiding both the difficulties of remote procedure calls and the high bandwidth required to move, from one machine to another, data instead of code. Beyond current applications of mobile agents in high-performance computing [2], wireless sensor networks [6,14], and fog computing [4,16], Future use includes large-scale wireless sensor networks and smart city infrastructures, which require that the protocols for messaging between agents scale up with the number of agents, nodes, and messages, and with the frequency of agent migration.

While they migrate, mobile agents need to be able to receive messages, from either fixed agents or other mobile agents. A *message delivery model* (sometimes called a *message delivery protocol* in the literature) is a set of rules and methods

This research was supported by grant PN-III-P1-1.2-PCCDI-2017-0734.

describing how to deliver a message sent from one agent to another that is identified by its name, in a distributed deployment, considering that the destination agent is mobile and is able to migrate between different hosts (or nodes). When mobile agents decide dynamically where to move next, it is challenging to create a *message delivery model* such that messages reach a rapidly moving agent in a timely manner. In the case of a large number of mobile agents that change host frequently, trade-offs exist between latency, reliability, performance, and network load. While many message delivery models have been surveyed from a qualitative point a view [15,17], choosing the appropriate model also requires a comparison of experimental results.

This paper introduces a framework for the comparison of message delivery models for multi-agent systems featuring mobile agents. We present the architecture of the framework, together with tools for analyzing the outcomes of simulated large-scale experiments. The result is an environment which developers can use to evaluate the advantages and trade-offs of either pre-implemented or newly developed models quickly and efficiently, using numerical comparisons for relevant criteria. The framework is implemented in Java and it is open-source.[1]

Based on characteristics such as number of agents and nodes, processing power of nodes, latency of the network, and frequency with which agents send messages or with which agents migrate, we generate a large number of scenarios for the simulation of the given conditions.

We have implemented several models for message delivery. All the implementations are available to the framework through the same API and can transferred to real-life MAS frameworks, such as the FLASH-MAS framework [12], or the popular Jade framework, as `MTP` instances. The implemented models have been evaluated on the same scenarios and evaluated according to the same metrics – message delivery success rate, mean delivery time, and network load. These metrics quantify the criteria presented by Virmani and by Rawat [15,17], and are also inspired by the work of Deugo [8]. Based on the results, we were able to make quantitative comparisons between the models and observe which model is more appropriate for each situation, whether we are dealing with a high rate of migration, or a large number of messages, or a resource-constrained network.

The next section presents work related to the subject of this paper. Section 3 introduces the architecture of the framework and its main features. Section 4 covers the comparison between several existing message delivery models, together with experimental results and discussion. The last section draws the conclusions.

2 Related Work

Existing models for message delivery in mobile agent systems have been previously surveyed and compared. Deugo [8] compares several classic delivery models from a theoretical point of view, without actual experiments. Virmani [17] and Rawat et al. [15] make qualitative comparisons of the state-of the art models

[1] The implementation is freely available as a Github repository at https://github.com/dragospetrescu/mobile_agents_system_simulator.

at the time. However, no means of quantitative comparison are offered. In the qualitative analyses, the features that are evaluated are generally a subset of the following: solution to the tracking problem (when an agent moves after the message is sent, but before the message reaches it), guaranteed message delivery, support for asynchronous communication, delivery in reasonable time, and transparency of location. However, without any quantization of these criteria, one cannot evaluate the trade-offs between features, such as whether a not-so-perfect success rate is a fair trade-off for other advantages, such as a good delivery time.

Message delivery models for communication in mobile agent systems are generally built around several well known schemata:

- the *centralized* solution, in which one server is tracking the whereabouts of all agents and forwards their messages accordingly; this is improved by the *home server scheme* where different servers are assigned to different partitions of the agent set, offering a more balanced solution than centralization [18];
- *blackboard* solutions, in which agents need to visit or contact the blackboard explicitly in order to get their messages [3,5];
- *forwarding proxy* solutions, in which each host remembers the next location to which an agent migrated, and messages will be forwarded along the path of the agent [7]; the *shadow protocol* combines the proxy model with the home server model by using proxies but agents regularly send updates of their location to their home server [1]; a combination of forwarding proxies and location servers is used by MEFS [9].

Mobile agents are currently used in several application areas, the most relevant being distributed computing (including here HPC and fog computing) and wireless sensor networks. In distributed computing, agent-based applications generally use centralized messaging or centralized directories [4]. However, experiments are only performed using a relatively small number of agents, a case for which the centralized server solution works fine. If the applications were to scale up to larger numbers of agents (e.g. in the thousands), the central server would become a serious bottleneck. In wireless sensor networks, mobile agents are used to gather information from WSN nodes. Some works only use mobility along a pre-calculated itinerary, with no communication [14]. In the cases where communication is needed, centralized communication methods are used, many times using Jade [6,13]. This works for small setups or for when the number of messages is low, but is not adequate to city-scale WSNs.

Research also exists in the field of distributed computing regarding distributed messaging [10], however these are made to support only fixed message receivers, which are not able to migrate through the network. This makes the problem that we address specific to the field of mobile multi-agent systems.

3 Framework Architecture

We see the implementation of a message delivery model as a system distributed across all hosts, which is able to pick a message from one agent in the system

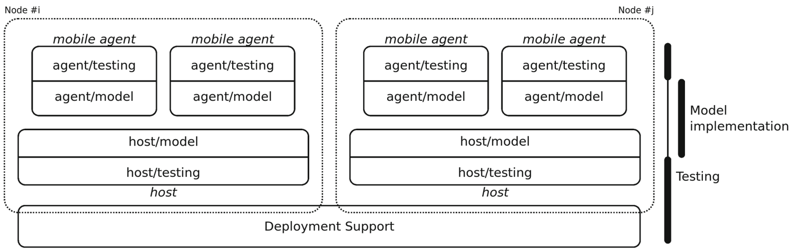

Fig. 1. The structure of elements in a deployment with two hosts, each on one machine, and 4 agents, two for each host.

and deliver that message to a mobile agent which is its destination, wherever that agent may be located at the current time.

In a distributed mobile multi-agent system, there exist 3 types of elements: mobile agents, hosts (or nodes), and what we call *deployment support*, which is able to actually send data through the network from one host to another, and is able to provide information about the network.

The architecture of the framework was built around the following principles:

– there should be a clear separation between the implementation of the message delivery model and the implementation of application-specific or framework-specific components of the multi-agent system;
– one should be able to change the message delivery model without changing the structure of the agents, making the application agnostic with respect to the delivery model that is used;
– one should be able to transfer the implementation of the delivery model from the framework to an actual MAS deployment, making the delivery model agnostic with respect to whether it runs in a simulation or in real-life.

This results in a modular structure which makes the implemented components be reusable across situations and between simulated and actual deployment.

For both the agents and the hosts we have integrated the separation described above, resulting in the following elements result (see also Fig. 1):

– the delivery *model* implementation; this implementation should be deployable in a real MAS application framework;
  • the *agent/model* segment is the part of a mobile agent which is model-specific and is able to communicate with the corresponding *host/model* segment; it stores any information which model-specific but must travel together with the agent;
  • the *host/model* segment contains all functionality that is dependent of the model, but is fixed to the machine, e.g. the registration of agents located on the host, or the routing of messages to or from the agents on the host;
– the *testing* components, including everything outside of the delivery model implementation, and which is used to evaluate the model;

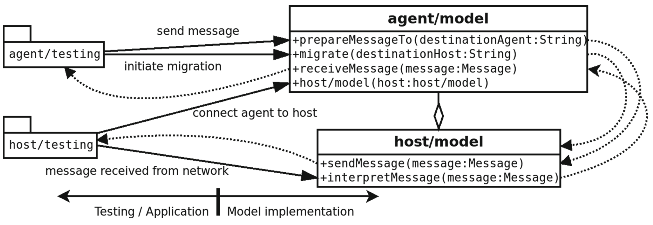
agent/testing
send message
initiate migration
agent/model
+prepareMessageTo(destinationAgent:String)
+migrate(destinationHost:String)
+receiveMessage(message:Message)
+host/model(host:host/model)
connect agent to host
host/testing
host/model
+sendMessage(message:Message)
+interpretMessage(message:Message)
message received from network
Testing / Application
Model implementation

Fig. 2. A UML class diagram, also describing the interactions between the model implementation and the testing components.

- the *agent/testing* segment is the part of a mobile agent which participates in evaluating the message delivery model; the agent/testing segment generates and initiates the sending of messages, or initiates the migration to other hosts, all according to the settings in the evaluation scenario; the agent/testing segment is homologous to an application-specific agent in a real deployed application.
- the *host/testing* segment contains all functionality that is fixed to the host, but is independent of the delivery model, e.g. packing and unpacking of mobile agents when they migrate from or to the host;

– the *deployment support* contains all the elements that simulate a real deployment, such as the capacity to delivery messages between machines;

The resulting structure of the framework is layered. Messages are transmitted as follows:

1. according to the evaluation scenario, the *agent/testing* segment of a mobile agent generates a message and passes it to the *agent/model* segment of the same mobile agent;
2. the *agent/model* implementation performs model-specific processing on the message and passes it to the local *host/model* segment of the host;
3. the *host/model* decides the network identifier of the host at the next hop and passes the message to the *host/testing* segment of the host, which passes it to *deployment support*;
4. at every intermediate hop (if required by the model), the message is passed to the local *host/model*, which decides on the next hop;
5. on the host which is the destination of the message, the *host/model* passes the message to the *agent/model* segment of the appropriate agent;
6. the *agent/model* segment passes the message to the *agent/testing* segment of the same mobile agent, which updates the statistics related to delivery success rate and delivery time.

Similarly, when an agent wishes to migrate, it informs the local *host/testing*, which packs the agent as a message and passes it to the local *host/model*, which routes it the same as with any message; when this special message arrives on the destination host, the local *host/testing* will unpack and deploy the agent and the local *host/model* will register the migration.

There may be other messages that travel through the system, which are not initiated by the *testing* components, but by *model* components, according to the specific message delivery model. Such messages may be disseminating updates on the location of agents or attempting to resend messages that have not been delivered.

In order for a developer or researcher to evaluate a new model one must only implement two Java classes – one for the *agent/model* part and one for the *host/model* part of the model implementation. The interfaces which these classes must implement contain a minimum of required methods: the `agent/model` must contain a method to create and send a message, a method to receive a message from the `host/model`, and a method to initiate the migration to another host; the *host/model* must contain a method to send a message to another host (via *deployment support*) and a method to interpret a message coming from another host. The exact communication between the *agent* and the *host* parts of the model implementation is not imposed. These relationships are presented in Fig. 2.

3.1 Experimental Scenarios

The proposed framework supports the automatic generation of evaluation scenarios with the purpose of estimating the performance of the various message delivery models in real life.

Time in an evaluation scenario is discrete, quantified by means of *time units.* In a time unit, each node is able to process a given amount of messages, and on each direct connection between nodes a message travels for a given length. For easier interpretation of results, the scenarios presented in this paper simulated a complete network with edges of varying length, but other network layouts may be specified in the configuration of experiments. Some delivery models require the network to be divided into regions. This is also done by the scenario generation component. While messages could have different sizes, larger sizes can be simulated by several unit-sized messages.

For each delivery model, about 1000 scenarios were generated, considering combinations of the following parameters:

- the number of nodes in the deployment – with values of 10, 50, 100;
- the number of agents in the scenario – with values of 2, 10, 100, and 200;
- the probability of an agent migrating to another node in a given time unit – with values of EXTREMELY LOW, LOW, NORMAL, HIGH, and EXTREMELY HIGH, each value assigned to a probability between 1 in 1000 and 1 in 10;
- the probability of an agent sending a message to another agent in a given time unit – with values of EXTREMELY LOW, LOW, NORMAL, HIGH, and EXTREMELY HIGH, each value assigned to a probability between 1 in 100 and 1 (the highest probability is when an agent sends one message in each time unit);
- the "CPU" power of hosts, specifying how many messages a host can process in a time unit – with values of EXTREMELY LOW, LOW, NORMAL, HIGH, and EXTREMELY HIGH, each value assigned to a number of 1 to 50 messages processed in every time unit;

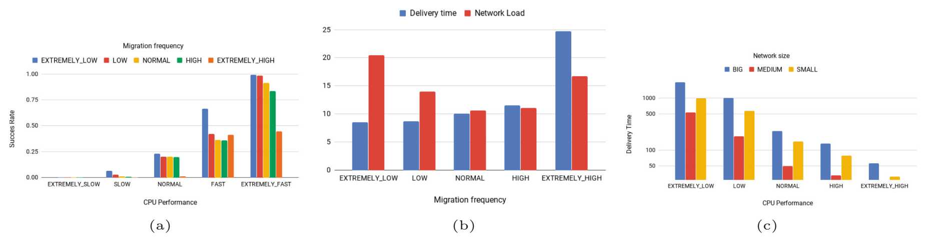

Fig. 3. (a) The success rate of Central Server Schema, depending on CPU power and migration frequency. (b) The delivery time (in time units) and the network load (in number of messages on the network in a time unit) for the Shadow Protocol, for various migration frequencies. (c) The performance of RAMDP, in terms of delivery time (on a log scale), depending on CPU power and network size.

The framework is used via a command-line interface offering arguments specifying the JSON files describing the hosts, the agents and the network, as well as values for migration frequency, message frequency, processing power, and the duration of the simulation. The configuration files for hosts and agents may contain per-host/per-agent values for processing power and for migration and messaging frequency. The framework is able to automatically generate all of the necessary JSON files, randomly, based on several parameters. Using files, however, as opposed to generating the scenario at every execution, allows for improved repeatability of experiments.

The results of one evaluation scenario are returned after the specified number of time units, as a tuple of 4 values which we use as comparison criteria, as detailed in the next section.

4 Comparison of Message Delivery Models

In order to demonstrate how our framework is able to handle comparison between message delivery models, we have implemented several such models and compared the experimental results of their evaluation.

In the comparison, we focused on how well various models handled difficult scenarios, characterized by limited processing power, increased number of messages, or increased probability of migration. For each evaluation scenario, we have used average parameters for all but a chosen characteristic of the scenario, and we have varied that characteristic in order to evaluate its impact on the performance of the model. As comparison criteria we use 4 values which the framework computes for each scenario or for an entire batch of scenarios.

The **message delivery rate** is the ratio of messages successfully delivered during the entire simulation, over the total number of sent messages:

$$Delivery\ rate = \frac{number\ of\ messages\ which\ have\ been\ delivered}{total\ number\ of\ sent\ messages}$$

The **mean message delivery time** is average number of time units it takes to deliver a message:

$$Time = \frac{\sum_i^{message} message\ delivery\ time\ for\ message\ i}{total\ number\ of\ messages}$$

The **mean network load** is the average number of messages which are in transit over the network, over the entire time of the simulation:

$$Load = \frac{\sum_i^{steps} number\ of\ messages\ in\ transit\ at\ time\ unit\ i}{number\ of\ steps}$$

The mean time during which messages that do not get to be delivered stay in the network or on the hosts, before being discarded, is computed as:

$$Time_{failures} = \frac{\sum_{failed\ messages} time\ spent\ in\ transit}{number\ of\ failed\ messages}$$

In order to validate the presented framework, we have implemented several known message delivery models from the literature. For each model, we have run 1000 scenarios with various configurations of the parameters. Each experiment ran for 100,000 time units, of which 30,000 time units were left for messages to be delivered, with no new messages being sent. We have grouped message delivery models into two categories [8]:

- *asynchronous* models, where it is the message that travels through the network, trying to reach its destination agent (leading to the chasing problem); such models are CS, FP, HSS, MDP, and MEFS (in some cases);
- *synchronous* models, where it is the destination agent that needs to synchronize with the source of the message or with the host where the message is currently stored and, once synchronized, the message is delivered immediately, so that messages spend almost no time traveling through the network; such models are RAMDP, Blackboard, and MEFS (in some cases).

For **asynchronous models**, we have analyzed how the success rate, the network load, and the delivery time are influenced by migration frequency and message frequency. Figure 4 presents these results, from which we can derive some useful insights into the situations various models are adequate for. In this comparison, the number of agents was large and CPU power has been relatively limited, in order to observe the efficiency of the models.

The **Central Server Scheme (CS)** model is characterized by the existence of a particular, central, server, which knows the locations of all agents. When agent migrate, they notify the central server. Every message that is sent between agents passes through the central server, which forwards it to the host where the destination is located. The central server is the bottleneck of the system and the performance of the system relies on the performance of the central server machine (as seen in Fig. 3 (a)). While the migration frequency has a moderate impact on performance, increasing the number of messages decreases performance abruptly (see Fig. 4 (b)), because all messages queue on the central server and don't get to be delivered. For situations where the number of messages is reasonable and reliability is not required at 100%, CS is well suited.

The **Home Server Scheme (HSS)** improves CS by distributing the central registry to several servers, with each agent being assigned to a specific home server [18]. Home servers know the distribution of agents among servers. HSS generally has good performance, but delivery time increases when there is a large number of agents, because, as with CS, performance is limited by the performance of individual nodes.

In the **Forwarding Proxy Protocol (FP)**, an agent leaving a host leaves a proxy on the host, pointing to its next location [7]. A message must 'chase' an agent in order to reach it. FP is the only completely decentralized model, among the models compared. FP has very good success rate even when the number of messages is very large. Its advantage comes at the cost of long delivery times and high network usage, even when the migration frequency or the number of messages are moderate. However, if reliability is needed, FP is the best choice. Other models based on FP add home servers, which need CPU performance in order to route messages.

The **Shadow Protocol** combines HSS and FP features, with agents using proxies but periodically updating their location on their home server (the 'shadow') [1]. The shadow forwards the message to the last location that it knows and from there the message chases the agent via proxies. Performance of the Shadow Protocol is better than FP when migration frequency increases, but is worse when the number of messages is large. The results in Fig. 3 (b) show how a lack of migration is related to higher network load, as more frequent migration leads to more frequent refresh of the information in the shadow, and therefore better performance. Being pseudo-centralized, performance of the Shadow Protocol can benefit from increased CPU performance for home servers. If the number of messages is not very high, Shadow offers good all-around performance.

The **Message Efficient Forwarding Schema** also combines centralized server features with forwarding proxies, introducing more reliability by placing forwarding proxies on hosts where it migrates from, and also changing its registration from the old host to the new host in order to increase efficiency [9]. MEFS has similar results to FP.

Message Delivery Protocol (MDP) uses a hierarchical structure to track the location of agents and to route messages in the system [11]. The structure, however, needs to be created specifically for each particular deployment. MDP is quite reliable when there are many messages, has good delivery times and does not load the network. It never achieves perfect reliability, but it is a good trade-off if short delivery times and low network usage are required.

We can observe that, when CPU power is limited, no asynchronous model handles a large number of messages too well, except maybe FP, which comes with other disadvantages for less stressful scenarios. For the case of very frequent messaging, MDP is a good choice.

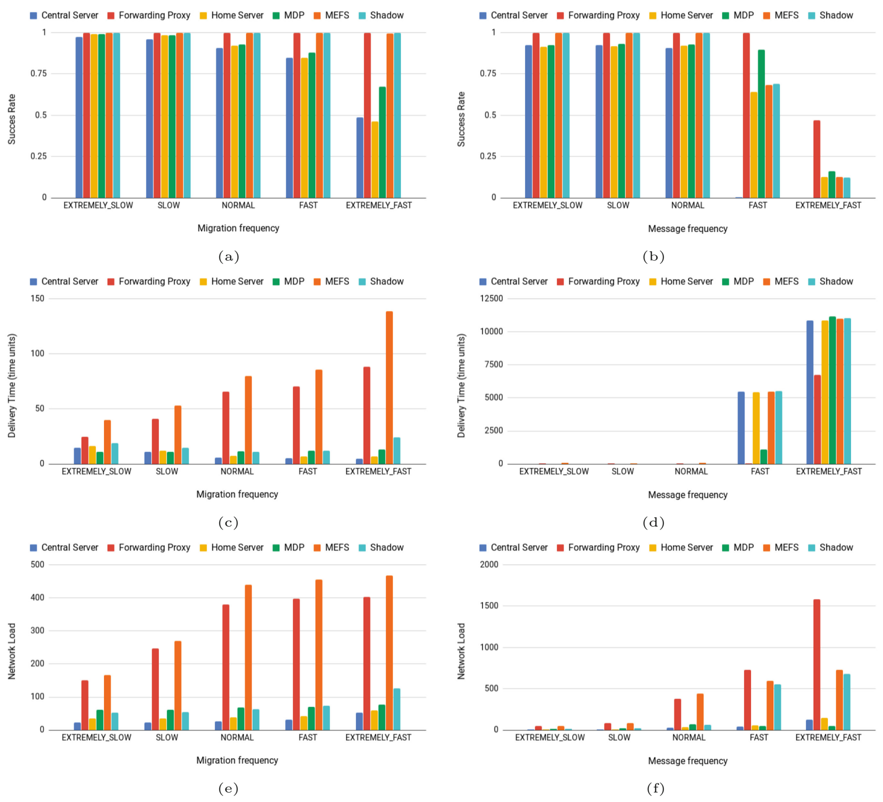

Fig. 4. Comparison of asynchronous models, for 100 nodes and 200 agents. On the left, migration probability varies between 1‰ and 10%, with messaging probability of 10%. On the right, messaging probability varies between 1% and 100%, with migration probability of 1%.

For **synchronous** models, we have analyzed how the success rate, the network load, and the delivery time are influenced by migration frequency. For these models, we have not presented a comparison where the messaging frequency varies, because their performance is not influenced by the number of messages – an agent retrieves all its messages received until that point at once. For the same reason, except for MEFS, which is a hybrid model, the network load is also quite low, as messages don't chase agents, but just wait on a host until an agent retrieves them.

In the **Blackboard** model, every node hosts a blackboard where agents leave messages [3]. An agent which is the destination of a message needs to move to the host where the message is stored in order to receive it. Lacking a specific action from the part of the agents to visit all hosts, the success rate of message delivery is low, and only increases when agents migrate frequently, having the opportunity to visit more hosts.

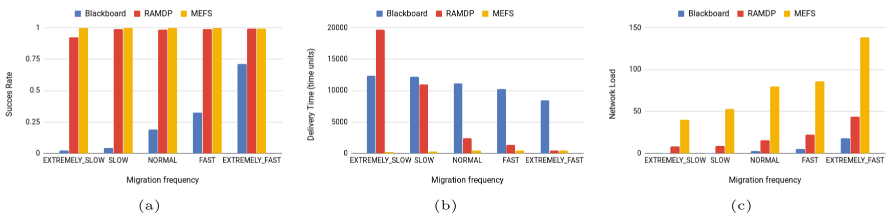

Fig. 5. Comparison of synchronous models, for 100 nodes and 200 agents, when migration probability varies between 1‰ and 10%, and messaging probability is 10%.

The **Reliable Asynchronous Message Delivery Protocol (RAMDP)** groups agents in regions and each region has a blackboard for messages [5]. Whenever an agent migrates, it informs the region and in return it receives its messages. As with the blackboard model, performance increases when agents migrate frequently, also because an agent receives messages only after it migrates and informs the region server. The design of the regions is important, as is visible in the case displayed in Fig. 3 (c), where a network of medium size offers the best performance because the number of regions is more adequate to the number of agents and nodes. RAMFS offers good success rate, especially when the frequency of migration is higher. As message retrieval is related to agent migration, agents that don't migrate don't get to receive their messages very often (Fig. 5).

5 Conclusion and Future Work

We have presented a framework for the experimental evaluation of message delivery models in a variety of scenarios, allowing users to compare the performance of various models and to ascertain which model is adequate for a specific situation. We have integrated the implementation of several delivery models and compared their performance in stressful scenarios. We were able to draw several conclusions related to the suitability of the models for various usage situations.

The framework does not currently support evaluating the robustness of a model when faced with network failure, faulty implementation, or agents that crash unexpectedly, nor the impact of the adoption of open systems, in which agents are able to enter and leave freely. This is a part of future work.

We intend to extend the framework with further metrics, for instance for analyzing CPU consumption, as opposed to only limiting CPU usage. We intend to develop the tools needed to analyze hosts individually, or by regions, as opposed to evaluating metrics across all hosts.

References

1. Baumann, J., Rothermel, K.: The shadow approach: an orphan detection protocol for mobile agents. Pers. Ubiquit. Comput. **2**(2), 100–108 (1998)

2. Benchara, F.Z., Youssfi, M., Bouattane, O., Ouajji, H.: A new scalable, distributed, fuzzy c-means algorithm-based mobile agents scheme for HPC: SPMD application. Computers **5**(3), 14 (2016)
3. Cabri, G., Leonardi, L., Zambonelli, F.: Mobile-agent coordination models for Internet applications. Computer **33**(2), 82–89 (2000)
4. Chang, C., Srirama, S.N., Buyya, R.: Indie Fog: an efficient Fog-computing infrastructure for the Internet of Things. IEEE Comput. **50**(9), 92–98 (2017)
5. Choi, S., Kim, H., Byun, E., Hwang, C., Baik, M.: Reliable asynchronous message delivery for mobile agents. IEEE Internet Comput. **10**(6), 16–25 (2006)
6. Derakhshan, F., Yousefi, S.: A review on the applications of multiagent systems in wireless sensor networks. Int. J. Distrib. Sens. Netw. **15**(5), 1550147719850767 (2019)
7. Desbiens, J., Lavoie, M., Renaud, F.: Communication and tracking infrastructure of a mobile agent system. In: Proceedings of the Thirty-First Hawaii International Conference on System Sciences, vol. 7, pp. 54–63. IEEE (1998)
8. Deugo, D.: Mobile agent messaging models. In: Fifth International Symposium on Autonomous Decentralized Systems, ISADS 2001, Dallas, Texas, USA, 26–28 March 2001, pp. 278–286. IEEE Computer Society (2001)
9. Jingyang, Z., Zhiyong, J., Daoxu, C.: Designing reliable communication protocols for mobile agents. In: 2003 23rd International Conference on Distributed Computing Systems Workshops, Proceedings, pp. 484–487. IEEE (2003)
10. John, V., Liu, X.: A survey of distributed message broker queues. CoRR abs/1704.00411 (2017). http://arxiv.org/abs/1704.00411
11. Lazar, S., Weerakoon, I., Sidhu, D.: A scalable location tracking and message delivery scheme for mobile agents. In: 7th Workshop on Enabling Technologies (WETICE 1998), Infrastructure for Collaborative Enterprises, CAUSA, Proceedings, 17–19 June 1998, Palo Alto, pp. 243–249. IEEE Computer Society (1998)
12. Olaru, A., Sorici, A., Florea, A.M.: A flexible and lightweight agent deployment architecture. In: 22nd International Conference on Control Systems and Computer Science, Bucharest, Romania, 28–30, pp. 251–258. IEEE (2019)
13. Outtagarts, A.: Mobile agent-based applications: a survey. Int. J. Comput. Sci. Netw. Secur. **9**(11), 331–339 (2009)
14. Qadori, H.Q., Zulkarnain, Z.A., Hanapi, Z.M., Subramaniam, S.: Multi-mobile agent itinerary planning algorithms for data gathering in wireless sensor networks: a review paper. Int. J. Distrib. Sens. Netw. **13**(1), 1550147716684841 (2017)
15. Rawat, A., Sushil, R., Sharm, L.: Mobile agent communication protocols: a comparative study. In: Jain, L.C., Behera, H.S., Mandal, J.K., Mohapatra, D.P. (eds.) Computational Intelligence in Data Mining - Volume 1. SIST, vol. 31, pp. 131–141. Springer, New Delhi (2015). https://doi.org/10.1007/978-81-322-2205-7_13
16. Roman, R., López, J., Mambo, M.: Mobile edge computing, Fog et al.: a survey and analysis of security threats and challenges. Future Gener. Comput. Syst. 78, 680–698 (2018)
17. Virmani, C.: A comparison of communication protocols for mobile agents. Int. J. Adv. Technol. **3**(2), 114–122 (2012)
18. Wojciechowski, P.T.: Algorithms for location-independent communication between mobile agents. In: Proceedings of AISB 2001 Symposium on Software Mobility and Adaptive Behaviour (2001)