A Platform for the Evaluation of Distributed Reputation Algorithms

V. Agate, A. De Paola, G. Lo Re, M. Morana

In Proceedings of the 22nd IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2018)

Publisher: ACM / IEEE

A Platform for the Evaluation of Distributed Reputation Algorithms

Vincenzo Agate, Alessandra De Paola, Giuseppe Lo Re, and Marco Morana
{firstname.lastname}@unipa.it
Università degli Studi di Palermo
Viale delle Scienze, ed.6, 90128, Palermo, Italy

***Abstract*—In distributed environments, where unknown entities cooperate to achieve complex goals, intelligent techniques for estimating agents' truthfulness are required. Distributed Reputation Management Systems (RMSs) allow to accomplish this task without the need for a central entity that may represent a bottleneck and a single point of failure. The design of a distributed RMS is a challenging task due to a multitude of factors that could impact on its performances. In order to support the researcher in evaluating the RMS robustness against security attacks since its beginning design phase, in this work we present a distributed simulation environment that allows to model both the agent's behaviors and the logic of the RMS itself. Moreover, in order to compare at simulation time the performance of the designed distributed RMS with a baseline obtained by an ideal RMS, we introduce an omniscient process called *truth-holder* which owns a global knowledge all involved entities. The effectiveness of our platform was proved by a set of experiments aimed at measuring the vulnerability of a RMS to a common set of security attacks.**

***Index Terms*—Agent-based simulation, Multi-agent systems, Distributed Reputation Management Systems**

I. Introduction

Complex distributed environments consist of a great number of autonomous entities that continuously exchange information in order to achieve a common goal. In a completely distributed architecture, these agents can take advantage of the lack of a central authority to adopt malicious, e.g., selfish, behaviors that may compromise the whole community. In this scenario, Reputation Management Systems (RMSs) allow cooperating agents to estimate the reliability of other nodes of the network before establishing actual interactions.

The adoption of a distributed RMS prevents single points of failure and represents a scalable solution because of the elimination of performance bottlenecks. However, this choice raises significant challenges for the designers. The aim of this work is to provide a software platform, designed also as a distributed application, that allows to evaluate the behavior of distributed RMSs before their real deployment, thus helping in the main challenges related to the design phase. First of all, evaluating the accuracy of reputation estimation and the time required to convergence is not a trivial task. Some RMSs are based on a sound mathematical formulation, which allows to theoretically evaluate them [1], but the underlying assumptions are not always valid and represent a strong constraint for designing new systems. The second issue concerns the evaluation of the RMS robustness toward malicious users. This point is crucial since the proper functioning of a distributed RMS strictly relies on the truthfulness of user's feedbacks. Such dependence makes RMSs sensitive to security attacks aimed to alterate the system capability of correctly estimating the agents' behavior. Moreover, the impact of an attack is hard to estimate since it depends on the features of the considered RMS [2]. Thus, a simulation environment that models the RMS logic and the agents' behavior can provide an invaluable support to foresee the impact of different design choices.

As summarized in [3], the most desirable characteristics of a simulation environment for RMSs should include i) the opportunity of defining abstract RMS models that can be easily implemented to represent specific systems, and that maintain the independence from a specific application; ii) the availability of well-defined metrics to evaluate the RMS's accuracy and its vulnerability to security attacks; iii) a high flexibility in varying the set of security attacks, the aspects of the RMS to be analyzed, and the simulation scenarios; iv) the possibility of performing large-scale simulations.

According to such guidelines, the proposed platform aims to be generic and not tailored to a specific application. Thus, the evaluation enabled by our tool is focused exclusively on RMS policies, agents' behavior, and security attacks, paying no attention to RMS implementation details and specific communication protocols. A set of high-level interfaces allows to disregard some low level details (e.g., implementing the agent communications, driving the simulations), so to let the researcher focus on higher level tasks, such as defining new reputation algorithms, or selecting the specific features to produce the desired resistance to security attacks. Moreover, our platform allows to compare the performance of the designed distributed RMS with a baseline obtained by an ideal, *omniscient*, RMS. This oracle, called *truth-holder*, knows the real outcome of each interaction and is able to apply the same reputation estimation algorithm while neglecting the effect of possible biased or malicious feedbacks.

The remainder of the paper is organized as follows. Section II discusses the related work. Section III provides a description of the platform architecture. Section IV describes the distributed simulation environment underlying our system. Section V presents both an assessment of the simulation tool itself, and its application to evaluate the impact of a set of

978-1-5386-5048-6/18/$31.00

security attacks on RMSs characterized by different policies. Finally, Section VI discusses our conclusions and indicates some future work.

II. Related Work

Some testbeds and simulators have been proposed in the literature with the aim of enabling the evaluation of distributed RMSs. Most of these solutions satisfy only a subset of the requirements identified in [3]. In particular, their common limitation is to be specific for a particular application, not allowing for an easy comparison among RMS policies designed for different domains.

The simulator described in [4] is defined as a competition between different strategies that are compared by evaluating the utility obtained by each agent at the end of the simulation. Such approach enables a useful comparison in Multi Agent Systems (MASs), but does not allow to evaluate the accuracy of the reputation estimation, nor its capability of discouraging malicious behaviors.

Some testbeds have been proposed to assess the RMS resistance to security attacks, such as [5] and [6]. The authors of [5] propose TREET, a simulation environment tailored for a marketplace scenario that allows to measure the resistance of a RMS to different types of attacks, such as reputation lag, proliferation, and value imbalance. TREET agents can dynamically join or leave a simulation, but the specific pattern of events is randomly generated and cannot be customized by the designers. It is worth noting that security attacks considered by TREET are meaningful in a marketplace scenario only, and cannot be applied to a generic RMS. On the contrary, the authors of [6] present a testbed which supports some generic security attacks, such as *slandering* and *promoting*. Such a tool requires to model the RMS as a sequence of transformations of a graph that represents both transactions among agents and their mutual trust. The main limitation of this environment is that the adopted model does not allow to simulate agent interactions nor to model complex agent behaviors. All these solutions force the RMS designers to meet specific constraints while modeling the behavior of the RMS.

These limitations are common to other works. ATB [7], for instance, mainly focuses on the decision making mechanism and neglects other relevant components of a RMS. TRMSim-WSN [8] is intended only to model RMSs over wireless sensor networks. The framework presented in [9] is a high flexible solution allowing the designers to define both new RMSs and security attacks. However, the performances of the RMS can be evaluated in terms of *hit rate* only.

The work proposed here aims to overcome the limitations of these simulators by proposing a generic RMS simulation platform. The core of the distributed multi-agent architecture we adopted has been briefly described in [10] and [11], while an early evaluation module including some metrics for a quantitative assessment of the RMS vulnerabilities to security attacks was presented in [2].

These preliminary versions do not allow to perform a comprehensive evaluation of the RMS policies, since the metrics adopted were specifically dependent on a given class of attacks. In this paper, we introduce an overall error metrics that allows to evaluate the RMS accuracy by highlighting the error caused by the distributed nature of the adopted policy. Moreover, in comparison with previous versions of our work, the platform presented here allows the definition of complex agent behaviors obtained by combining the different ways an agent may act as *resource* and *feedback* provider.

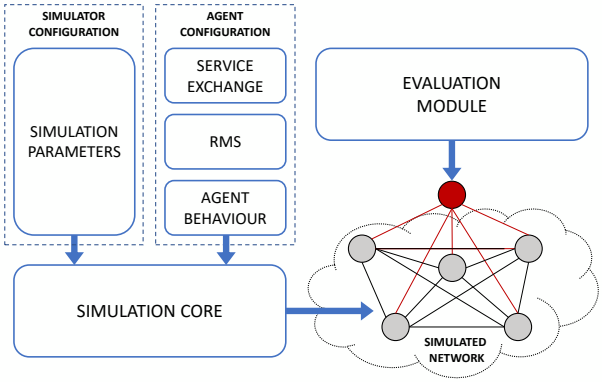


Fig. 1: Logical overview of the simulation platform. Users can specify simulation parameters, agent characteristics, and algorithms used by the RMS. The simulated network is analyzed through a customizable evaluation module.

III. Platform Architecture

The simulation platform is built according to a two-level architecture, where the upper level is responsible for modeling the RMS and the simulation scenario, and the lower level implements the communication and control primitives needed for actually driving the simulation. At the topmost layer, the RMS is modeled as a fully distributed system in which autonomous agents interact in order to exchange services. Such a behavior is implemented in the lowest layer through a distributed environment where each agent is mapped on a different process, and where processes communicate with each other by exchanging messages.

The platform structure and its working mechanisms are developed with the aim of hosting a generic RMS. According to the available interfaces, designers can re-define some system features in order to model a novel distributed application and its RMS, or they can combine existing solutions already available in the platform.

As shown in in Fig. 1, the simulator can be adapted to model different RMSs by specifying the *agent configuration*. This phase allows to detail the service request/response policies, the algorithms behind the RMS, and the different behaviors that each agent can follow over time. Once such configuration is completed, it is possible to perform the *simulation configuration* by specifying a set of available parameters. In this phase it's possible to specify the topology of the reputation network that lists the set of neighbours each agent can interact with. Moreover, users can specify the behavior each agent has

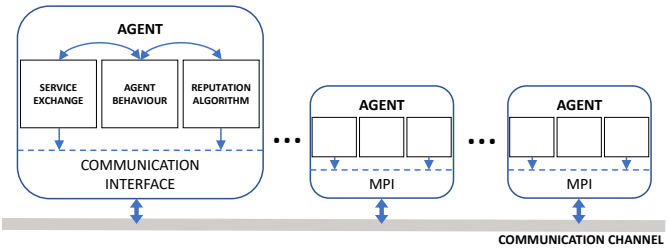


Fig. 2: The distributed multi-agent simulation scheme. Agents are mapped to system processes that communicate through the MPI protocol.

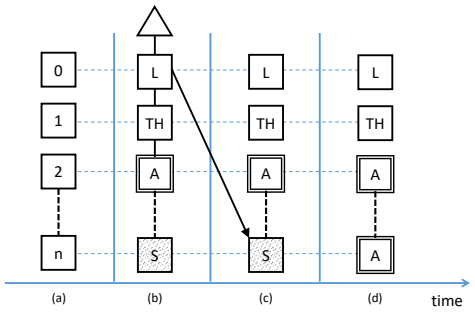


Fig. 3: Parallel processes managed by the simulation platform: (a) the MPI environment starts a set of identical processes; (b) by reading the simulation parameters, each process adopts its own behavior. (c) MPI messages sent by the leading process cause some other processes to change their status (d).

to follow during the simulation, and the possible sequence of actions to perform. Finally, the *Evaluation Module*, which analyzes the RMSs performance at the end of the simulation, can be customized by defining any ad-hoc evaluation metrics.

During the simulation, the agents interact with each other in order to provide/receive services and obtain a reward according to the synchronous, time-discrete model proposed in [12]. The role of the RMS is to apply an incentive mechanism that makes the reward proportional to the cooperativeness of each agent, computed by taking into account the whole community of agents. As result, agents are able to select a service provider according to the policy established by the RMS, e.g., by selecting the provider which corresponds to the highest expected utility.

The simulation evolves through a set of rounds during which all the agents cyclically perform the same sequence of steps. The behavior of a single agent within a round is defined by implementing some functions inherited from an abstract *agent class* provided with the simulation library. More specifically, the designers can specify:

- the service exchange logic, which rules the sending of service requests and replies (according to the RMS policy);
- the set of RMS algorithms, which specifies how to spread agents' opinion to their neighbourhood and how to compute the reputation values;
- the agent behavior, which models its cooperativeness during the service exchange.

With respect to service provisioning, the agent behavior can be fully *cooperative*, fully *selfish*, or partially cooperative, with a degree that can be specified as parameter. As regards the contribution to the RMS, the agent behavior can be *honest*, *slander* (if false negative feedbacks are provided) or *promoter* (in case of false positive feedbacks). The adoption of the *composition over inheritance* principle allows to easily define new agents whose behavior is a combination of these two behavioral dimensions. Moreover, new behaviors can also be defined by inheriting the available abstract classes.

The *Evaluation Module* allows to analyse the RMS's performance with different levels of detail. For instance, the simulator allows to evaluate the average reputation estimated by the RMS over the whole network, but it makes also possible to analyse with a high degree of granularity the trend over time of the reputation of a given agent as estimated by one of its neighbours. Furthermore, the *Evaluation Module* allows to compare such outcomes with those obtained by the *truth-holder* in order to understand whether the errors are due to the reputation algorithm, or to possible bias introduced by false user feedbacks.

IV. The Simulation Environment

One of the most important requirements of a simulation environment for RMS evaluation is to guarantee a high degree of parallelism to support large-scale simulations. To meet this constraint, our simulator takes advantage of a distributed environment where each agent is simulated by a process running on a computer cluster. Fig. 2 highlights the four elements characterizing an agent. Besides the aforementioned components, i.e., *service exchange*, *RMS algorithms*, and *agent behavior*, a *communication interface* is needed to enable messaging over a communication channel. In order to provide the programmers with a standard protocol, communication between processes is based on Message Passing Interface.

The core of the platform we proposed is written in C++ and exploits MPICH, a MPI library available for many UNIX like distributions and Windows OS, to provide the designer with an easy tool to implement his own distributed algorithm.

In order to simulate a dynamic network, the processes started with the simulation kickoff include both *active* agents, and a group of *silent* processes representing agents that could later join the RMS network. While all processes are started as identical copies of the same initial process, after the initialization phase the process behavior is changed according to the simulation parameters, as shown in Fig. 3-b. A process assuming the role of leading process is responsible for the management of the life cycle of other processes. For instance, for simulating a new agent joining the RMS network, the *leading process* wakes up one of the *silent* processes, sending an appropriate message through the MPI communication interface, Fig. 3-c. The process receiving the message will change its status at the next simulation step becoming *active*,

Fig. 3-d. Conversely, when an agent leaves the RMS network, the corresponding process is forced to change its status from *active* to *silent*.

Finally, the *truth-holder*, collects all the transactions occurred between RMS agents, allowing to evaluate the *ground truth* about reputation, thus neglecting the effect of possible biased or malicious feedbacks. The centralized coordination performed by the *leading process* and the *truth-holder* only concerns the management of the processes and the evaluation of the RMS, while service exchanging and communications within the RMS are performed in a totally distributed way.

A. Defining the Service Exchange Model

In distributed applications, the role of a RMS is to provide the user with information needed to select the best agent to interact with, according to the quality of past interactions between providers and the whole community. Thus, agents should be able to provide feedbacks any time a transaction is completed in order to let the RMS compute and spread over the network the new reputation values.

To this aim, in our platform, each agent maintains a list of known providers and advertises the exported services by means of a *service announcement protocol*, i.e., broadcasting its provided services to the whole network, sending information only to the neighbour agents, or to a group of agents by using a hop-by-hop path.

The *service selection* method allows the single agent to implement its final decision, e.g., by selecting the provider with the highest reputation.

Finally, the *service reply* policy specifies whether and how to reply to service requests. This method allows to implement the *incentive mechanism* of the modeled RMS, e.g., it is possible to define a policy according to which a provider replies only to agents whose reputation is above a given threshold, or randomly decides to reply with a probability function of the consumer reputation. Decisions taken by the *service reply* method, are further influenced by the agent's *behavior as service provider*, as detailed in section IV-C.

B. Defining the RMS algorithms

The main step for setting up the simulation environment consists in the definition of the RMS algorithm.

Regardless of the specific reputation representation and the reputation management algorithm, any RMS [10] includes four common parts. Generally, each agent locally computes a first estimation of the reputation of other agents by exploiting its direct interactions with the community. This *local trust* can not be used to predict the behavior of entities that have not been previously seen. Thus, the *local trust evaluation* is supported by two more components, namely the *gossip protocol*, which allows agents to exchange information about the reputation of their neighbours, and the *information fusion* mechanism, that is used to merge local with gossiped reputation. The fourth component is the *incentive mechanism* used by the RMS to discourage antisocial behavior by rewarding trustworthy agents and limiting malicious ones.

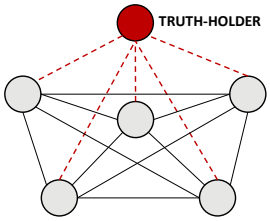


Fig. 4: The role of the *truth-holder* process.

In order to support the designers in the definition of specific policies, the simulator provides the *Reputation Interface* which allows to represent reputation as a scalar, as a discrete value (e.g., *good*, *medium*, *bad*), as a vector containing different parameters (e.g., type of behavior, cooperativeness, level of engagement), or as a generic object.

C. Defining the Agent behavior

The distributed nature of RMSs makes them vulnerable to several types of security attacks, as described in [13], [14].

In most cases, the worst type of attacks comes from the inner of the distributed system, by authorized users that exploit vulnerable system services. In such a scenario, the attacker is usually allowed to cooperate with other malicious agents in order to compromise the system integrity.

In general terms, an attack requires an agent or a group of agents to adopt a specific behavior as *service provider* or *RMS member*.

Promoting [15] and *slandering* [16], for instance, are two examples of attacks that require a malicious behavior of a group of agents *as RMS members*. Such attacks are performed by spreading poisoned feedbacks in order to maliciously alterate the RMS outcome. Both attacks can be neutralized by working on the *information fusion* mechanism, allowing the system to resist to a wild diffusion of fake information.

Whitewashing [17] and *traitor* attacks [18], on the other hand, imply the malicious behavior of an agent *as service provider*. Such attacks are performed by alternating cooperative and selfish behaviors, in order to abuse system resources.

Our simulation environment includes a set of behavioral patterns the designer can combine to describe the behavior of single agents. While configuring a specific simulation scenario, it is possible to choose how many agents should exhibit a behavior b, where b can be either atomic, e.g., *slander*, or obtained by composing n atomic behaviors, $b = [b_1, ..., b_n]$. For example, when planning a complex collusion attack, it might be desirable that an agent A contributes both to rise the reputation of an agent, and to reduce that of another one. In such case, its behavior could be expressed as $b_A = [b_{slander}, b_{promoter}]$ to let a specific agent perform both a slandering and a promoting attacks with different targets.

D. Evaluation Module

In order to measure the robustness of a RMS against different type of attacks, the simulation platform provides the

designers with an objective measure of the error between the actual reputation values and those estimated by the RMS.

As discussed in the previous Sections, the mechanisms implemented by a RMS to estimate the reputation of a given agent can be maliciously biased by forged information sent to counterparts. In order to evaluate the impact of false feedbacks on the RMS under analysis, it is useful to consider the reputation hypothetically estimated by an ideal omniscient RMS that knowns the true outcome of each transaction, and that can not be influenced by false feedbacks.

Such an ideal RMS is implemented through a *truth-holder* process (see Fig. 4), external to the agents network, that, at each simulation step, is responsible for collecting the real outcomes of agent interactions in order to build the ground-truth reputation R_i^* for each agent i. It is worth noting that the *truth-holder* is totally transparent to the RMS, and it only aims to provide a centralized tool to compute the error associated with the reputation estimated by the RMS.

In order to distinguish the bias introduced by forged feedbacks, it is convenient that the reputation aggregation algorithm adopted by the *truth-holder* is the same used by the network agents; thus, it must be redefined by the designers in order to meet the behavior of the RMS under analysis.

The *average reputation* of the agent i at time t is computed by the *Evaluation Module* by considering the reputation values individually estimated by all the network agents:

$$\overline{r_i}(t) = \frac{\sum_{j \in N} r_{ji}(t)}{|N|}. \quad (1)$$

where r_{ji} are the reputation values computed by the RMS for all the agents j that received a service from i, and N is the number of agents involved in the simulation.

Thus, the *single node error* can be obtained step-by-step as the difference between the *ground truth* and the *average reputation* of the agent i:

$$e_i(t) = |R_i^*(t) - \overline{r_i}(t)|. \quad (2)$$

This error can be considered as the instant measure of how much the reputation values computed by the RMS diverge from the actual behavior of the agents. The mean value of these errors computed over the whole network represents the *average system error*:

$$\overline{E}(t) = \frac{\sum_{i \in N} e_i(t)}{|N|}. \quad (3)$$

This value can be used to evaluate the performance of the RMS since the first simulation steps. For instance, in order to reduce the simulation time, if the average error of the RMS is above a threshold fixed by the designer, the current simulation could be stopped before its completion.

V. Experimental Analysis

In this Section, we first illustrate how to set up the simulation platform to model the particular RMS to be analyzed. Then we present a set of experiments aimed to assess both the performances of the simulator itself, and the robustness of the RMS to a common set of security attacks. These experiments have been performed using a cluster of virtual machines, managed by ESXi, a Vmware hypervisor.

A. Configuring the RMS under analysis

As regards the Service Exchange Model, we consider a RMS where all agents provide the same single service; nevertheless, the number of available services and their distribution among providers can be specified as simulation parameters. Each agent sends the list of the exported services using the *service announcement protocol*. In our case, the single service is announced only to the direct neighbours (defined according to the topology provided as input of the simulation) of the agent. Using the *service selection* method, an agent can select the service to be consumed. In order to make the RMS analysis independent from a specific decision making strategy, we implemented a dummy method that sends a service request to each neighbour agent without performing any specific choice. In this way, the whole neighbourhood is uniformly explored, avoiding the bias potentially introduced by a specific policy, e.g., select the provider with the highest reputation.

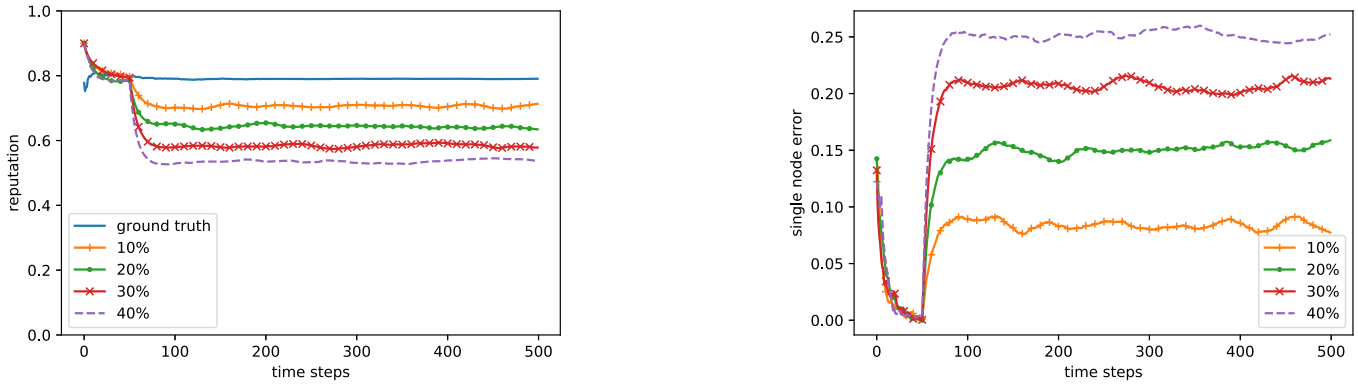
The *service reply* method implements the simple *incentive mechanism* proposed in [19], which consists in providing a random reply with a probability proportional to the requester's reputation. The *behavior as service provider* method enforces such probability proportionally to the agent's cooperativeness (a cooperative agent has a cooperativeness degree equal to 1, while a totally selfish agent has a cooperativeness degree equal to 0). Finally, an agent rates each successful interaction with 1, or 0 if the service has not been provided.

The considered RMS consists of four components. The *local trust evaluation*, inspired to [1], computes the *local trust* of a given provider as the number of satisfactory transactions over the total number of requests sent in the last time interval. A weighting factor α is used to specify how the local trust affects the overall reputation that agents own about their peers. The chosen *gossip protocol* establishes that information is exchanged with neighbour agents only. The *information fusion* mechanism is based on [20], which states that gossiped information is weighted with reputation of the gossiper agents, whilst the impact of received opinions on the overall reputation is weighted by means of a parameter, β. α and β, together with the initial default reputation value, are declared as *varying parameters* so that they can be automatically tuned by the simulator in different simulation runs.

The reputation values are represented as scalar in the range $[0, 1]$, with 0.1 steps. In our case study the *truth-holder* computes the ground-truth reputation of the agent i, i.e., $R_i^*(t)$, according to the local trust evaluation algorithm:

$$R_i^*(t) = \frac{\sum_{j \in N} sat_{ji}^*(t)}{\sum_{j \in N} (sat_{ji}^*(t) + unsat_{ji}^*(t))}, \quad (4)$$

where $sat_{ji}^*(t)$ and $unsat_{ji}^*(t)$ are the number of transactions that i satisfied, or unsatisfied, by providing a service to j,



(a) (b)

Fig. 5: Reputation (a) and average system error (b) of the RMS considered as case study during a slandering attack while varying the percentage of malicious agent involved.

during the time window t. This value, calculated for each of the N agents involved in the simulation and updated at each simulation step t, can be computed considering all the transactions occurred from the beginning of the simulation, or a subset within a specified time window.

B. Resistance to attacks of increasing severity

The first set of experiments aimed to evaluate the robustness of a specific RMS to security attacks based on the injection of forged feedbacks.

The simulated network is fully connected and is composed of 100 agents that interact to exchange a single service. In order to speed up the reputation estimation process, at each time step each agent interacts with all the others. For each agent, the *behavior as service provider* is obtained by setting a cooperative degree equals to 0.8.

For our purpose, we consider here the slandering attacks. During such attacks, in order to reduce the reputation of the victim agent, a variable percentage of malicious agents (i.e., 10%, 20%, 30% and 40%), uses the gossip protocol to disseminate forged feedbacks within the community. The duration of the simulations are set to 500 time steps and the attacks start after 50 time steps.

Fig. 5a compares the reputation computed by the *truth-holder* and the average reputation estimated by the distributed RMS, while varying the percentage of malicious agents. As we can observe, the reputation estimated by the RMS moves away from ground truth as much as the percentage of malicious agents increases. Specularly, Fig. 5b shows the average system error of the considered RMS, evaluated as the average difference between the reputation value estimated by the RMS and the corresponding ground truth. Results suggest that the error made by this specific RMS grows quite proportionally to the percentage of implicated agents. Thus, it does not exhibit an amplification of the false negative opinions, thanks to the inclusion of the direct experience (i.e., local trust) in the information fusion mechanism.

C. Comparisons among different RMS policies

The second set of experiments shows how the proposed platform allows to compare different RMS policies by evaluating their accuracy and resistance to forged feedbacks, given the same network configuration. The reputation network for this set consists of 300 agents, where 30% of them are implicated to perform a slandering attack against a 10% of cooperative agents. This experiment compares three policies obtained by varying the parameter β of the RMS considered as case study. The first policy uses only direct experience to estimate agents' reputation, i.e., $\beta = 0$. The second policy is characterized by a good balance between local trust and gossip information, and is obtained by setting $\beta = 0.2$. The third policy relies excessively on gossip information, with a weight $\beta = 0.8$. Fig. 6a compares the reputation computed by the *truth-holder* and the average reputation estimated by these policies, while Fig. 6b shows the corresponding average system errors. As expected, smaller weights to gossip information reduce the vulnerabilities to attacks characterized by the injection of false information. In the borderline policy where gossiped information are not considered at all, the estimated reputation corresponds to the ground truth, thus obtaining an average error which quickly goes to zero.

The same analysis can be performed considering promoting attacks, as shown in Fig. 6c and 6d. In this case, using the same network configuration described above, promoting attacks are performed to raise the reputation of a group of target agents whose real cooperativeness is 0.2. As shown before, policies that limit the weight of gossiped information are characterized by greater resistance.

D. System scalability

Some tests were run to verify the capability of our platform of measuring the actual performance of a given RMS, regardless of the size of the reputation network. In particular, we considered a slandering attack launched by a set of implicated agents (the 20% of the network) against other agents (the 10%

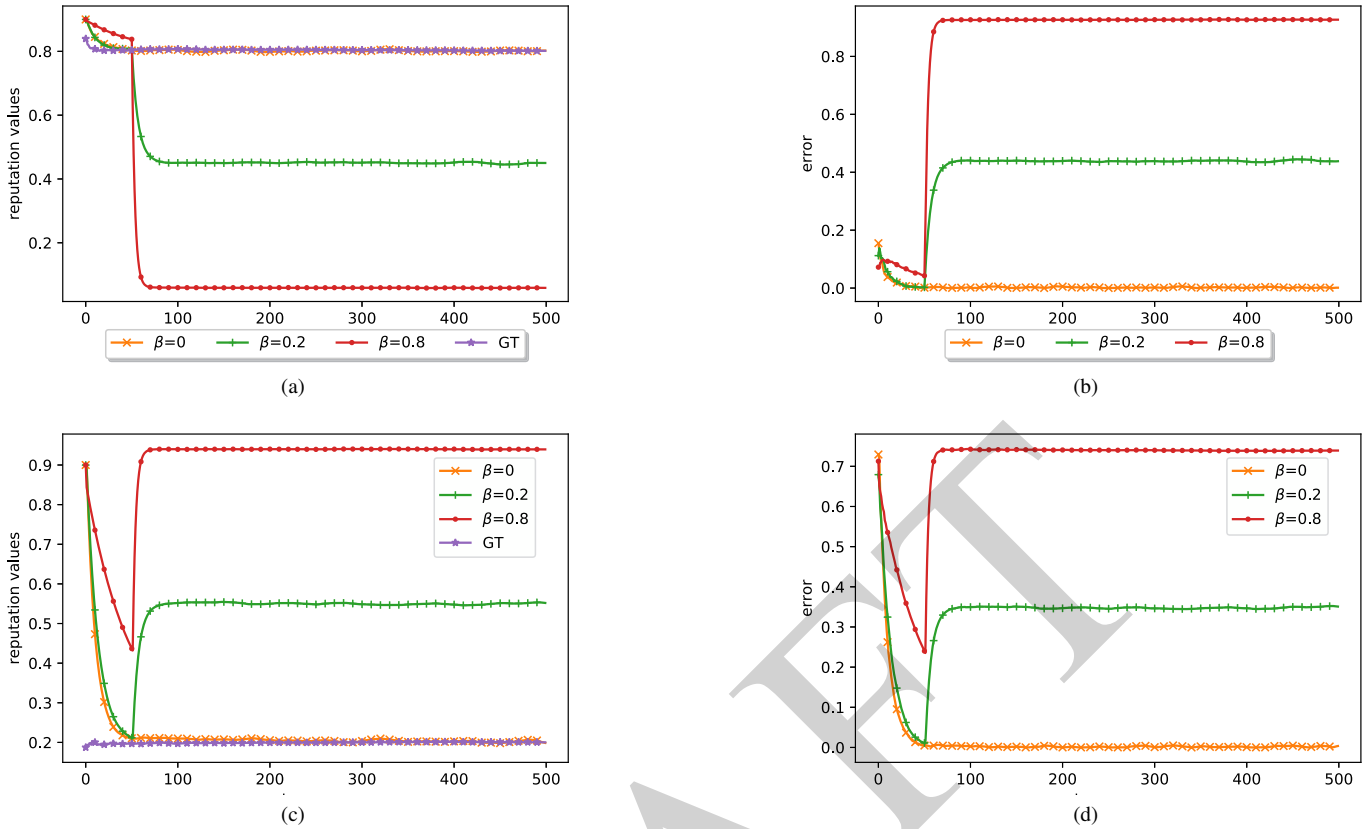


Fig. 6: Reputation (a, c) and average system error (b, d) measured while simulating slandering (a, b) and promoting (c, d) attacks performed on a network of 300 nodes with 30% of implicated agents. The different curves are obtained while varying the parameter β which weights gossip information with respect to the local trust. The *GT* curve shows the reputation computed by the *truth-holder*.

of the network) that have a cooperation degree of 0.8. Fig. 7a compares the reputation computed by the *truth-holder* and the average reputation estimated by the RMS on networks of 100, 200, 300, 400, and 500 nodes. Fig. 7b shows that the corresponding average system errors are quite similar, so proving that the diagnostic capability of the simulator is not dependent on the size of the network.

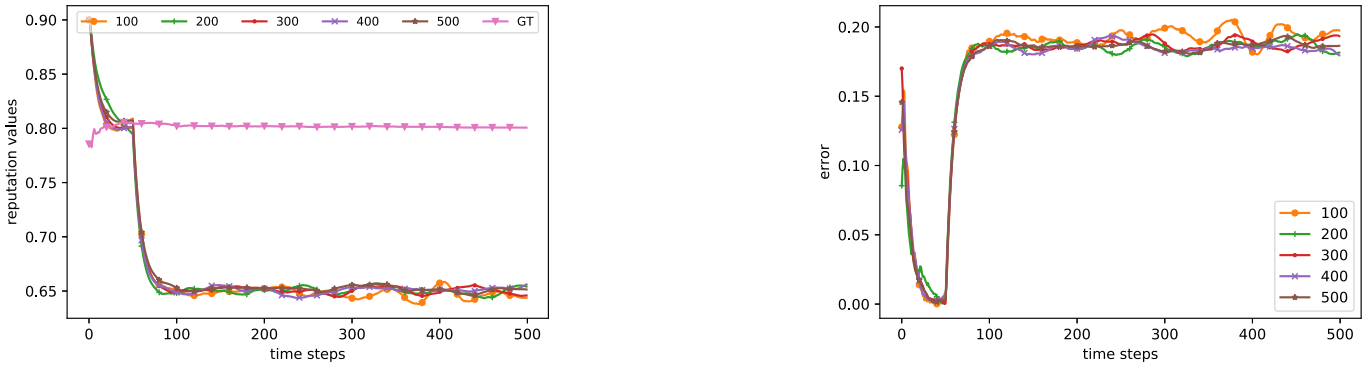
Other tests have been performed to measure how the simulation time depends on the number of deployed computational nodes. In particular, three experiments were run considering clusters of 4, 8, and 16 single-core nodes (SCNs). Results from Fig 8 show that the number of SCNs deeply impacts on the number of agents the platform can simulate. In particular, the current implementation of the simulator is not able to support more than 300 agents when using 4 SCNs, while in order to simulate a network of 500 agents at least 16 SCNs are needed. This is mainly due to the inter-process communication routines needed to support the simulation, according to which every agent exchanges message with all other agents in the network, regardless of the network topology. Nevertheless, for any number of SCN, the simulation time exhibits a quadratic growth, which is quite reasonable to simulate reputation networks of significant dimensions.

VI. Conclusions and Future Work

Distributed environments where autonomous agents act cooperatively need a Reputation Management System to estimate the reliability of unknown agents before starting a service exchange. In this work we presented a comprehensive simulation platform that can be used by developers to assess a distributed RMS since the design phase. The proposed solution allows to define new RMSs by exploiting a number of existing interfaces, and to manage large-scale simulations where agents can adopt complex behaviors that can be easily implemented by inheriting basic behavioral patterns. Different security attacks that can be defined by modifying the agent's behavior, and the robustness of the analyzed RMS to these attacks can be evaluated through to a customizable evaluation module.

The effectiveness of the proposed solution has been proved by presenting a set of experiments which show how our platform allows to assess the vulnerability of a RMS to a common set of security attacks.

As future work, we plan to exploit the simulation tool to evaluate the RMS under different conditions, e.g., by defining



(a) (b)

Fig. 7: Reputation (a) and average system error (b) measured while simulating a slandering attack in which the 20% of the network is implicated and all the agents have a cooperativeness degree of 0.8. The different curves are obtained in networks composed of 100, 200, 300, 400, and 500 nodes. The *GT* curve shows the reputation computed by the *truth-holder*.

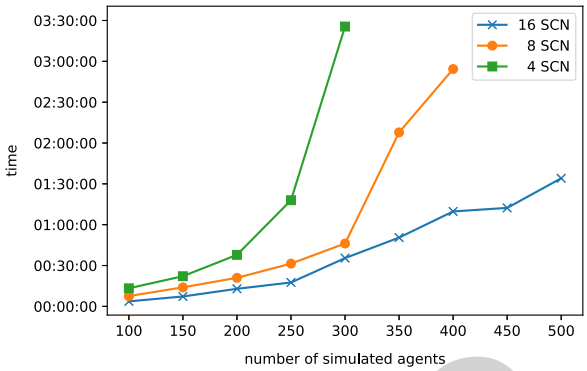


Fig. 8: Simulation time using 4, 8, and 16 single-core nodes.

other decision making strategies to send a service request and to select the service provider, or using more complex network topologies. We are also working on the release of an open source version of the platform including some popular RMSs and common attacks proposed in the literature.

References

[1] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigentrust algorithm for reputation management in P2P networks," in *Proc. of the 12th Int. Conf. on World Wide Web*, 2003, pp. 640–651.

[2] V. Agate, A. De Paola, G. Lo Re, and M. Morana, "Vulnerability Evaluation of Distributed Reputation Management Systems," in *InfQ 2016 - New Frontiers in Quantitative Methods in Informatics*, 2016, pp. 1–8.

[3] E. Koutrouli and A. Tsalgatidou, "Reputation systems evaluation survey," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 35, 2016.

[4] K. K. Fullam, T. B. Klos, G. Muller, J. Sabater, A. Schlosser, Z. Topol, K. S. Barber, J. S. Rosenschein, L. Vercouter, and M. Voss, "A specification of the agent reputation and trust (ART) testbed: experimentation and competition for trust in agent societies," in *Proc. of the 4th Int. joint Conf. on Autonomous agents and multiagent systems*, 2005, pp. 512–518.

[5] R. Kerr and R. Cohen, "Treet: the trust and reputation experimentation and evaluation testbed," *Electronic Commerce Research*, vol. 10, no. 3, pp. 271–290, 2010.

[6] P. Chandrasekaran and B. Esfandiari, "Toward a testbed for evaluating computational trust models: experiments and analysis," *J. of Trust Management*, vol. 2, no. 1, pp. 1–27, 2015.

[7] D. Jelenc, R. Hermoso, J. Sabater-Mir, and D. Trček, "Decision making matters: A better way to evaluate trust models," *Knowledge-Based Systems*, vol. 52, pp. 147–164, 2013.

[8] F. G. Mármol and G. M. Pérez, "TRMSim-WSN, trust and reputation models simulator for wireless sensor networks," in *Proc of the IEEE Int. Conf. on Communications (ICC'09)*. IEEE, 2009, pp. 1–5.

[9] A. G. West, S. Kannan, I. Lee, and O. Sokolsky, "An evaluation framework for reputation management systems," *Trust Modeling and Management in Digital Environments: From Social Concept to System Development*, pp. 282–308, 2010.

[10] V. Agate, A. De Paola, G. Lo Re, and M. Morana, "A simulation framework for evaluating distributed reputation management systems," in *Proc. of the 13th Int. Conf. on Distributed Computing and Artificial Intelligence*, 2016, pp. 247–254.

[11] V. Agate, A. De Paola, S. Gaglio, G. Lo Re, and M. Morana, "A framework for parallel assessment of reputation management systems," in *Proc. of the Int. Conf. on Computer Systems and Technologies (CompSysTech)*, june 2016.

[12] N. A. Lynch, *Distributed algorithms*. Morgan Kaufmann, 1996.

[13] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Computing Surveys (CSUR)*, vol. 42, no. 1, p. 1, 2009.

[14] Y. Sun and Y. Liu, "Security of online reputation systems: The evolution of attacks and defenses," *IEEE Signal Processing Mag.*, vol. 29, no. 2, pp. 87–97, 2012.

[15] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li, "An empirical study of collusion behavior in the maze p2p file-sharing system," in *Proc. of the 27th Int. Conf. on Distributed Computing Systems (ICDCS'07)*. IEEE, 2007, pp. 56–56.

[16] S. Ba and P. A. Pavlou, "Evidence of the effect of trust building technology in electronic markets: Price premiums and buyer behavior," *MIS quarterly*, pp. 243–268, 2002.

[17] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and whitewashing in peer-to-peer systems," in *Proc. of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*. ACM, 2004, pp. 228–236.

[18] S. Marti and H. Garcia-Molina, "Taxonomy of trust: Categorizing p2p reputation systems," *Computer Networks*, vol. 50, no. 4, pp. 472–484, 2006.

[19] C. Crapanzano, F. Milazzo, A. De Paola, and G. Lo Re, "Reputation management for distributed service-oriented architectures," in *Proc. of the 2010 Fourth IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshop (SASOW)*, 2010, pp. 160–165.

[20] A. De Paola and A. Tamburo, "Reputation Management in Distributed Systems," in *Proc. of the 3rd Int. Symp. on Communications, Control and Signal Processing (ISCCSP)*, 2008, pp. 666–670.