Scheduling Reputation Maintenance in Agent-based Communities Using Game Theory

Mohamed Amine M'hamdi
Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
Email: m_mhamd@encs.concordia.ca

Jamal Bentahar
Concordia Institute of Information System Engineering, Concordia University, Montreal, Canada
Email: bentahar@ciise.concordia.ca

***Abstract*—In agent-based systems, agents can be organized within groups, called communities, where members are providing similar or complementary services. Agent-based communities of web services is an example of such systems. Managing reputation of each agent and of the whole community is a key issue towards securing this type of systems, where a controller agent is designed to observe and check the behavior of each member to update and maintain the system's reputation. Scheduling the maintenance activity by deciding about the moment where the check has to be done is still an open problem. Because it is highly expensive, maintenance cannot be done every moment or based on small history of agents' behaviors. We propose in this paper a scheduling algorithm that helps the controller agent improve the quality of the reputation mechanism, which increases the trust value of users toward the community. The proposed algorithm is based on a class of games called Bayesian Stackelberg. Our Bayesian Stackelberg game is designed between the controller agent and community members, for example agent-based web services. We simulate and compare the efficiency of our algorithm with other stochastic techniques, namely uniform, normal and Poisson distributions. This research draws the lines for future work in the subject of optimizing reputation mechanisms through maintenance in different time intervals.**

***Index Terms*—Trust, Multi-Agent Systems, Game Theory,**

I. Introduction

A. Motivations

Trust and reputation have gained tremendous attention in online systems [1], [2]. Using multi-agents technology, there have been many efforts to improve the reputation of these systems using different probabilistic and logic based techniques [3], [4], [5], [6]. In these systems, agents having similar or complementary functionalities can be grouped together into communities called *agent-based communities*[1]. We define an agent-based community as a virtual organization of autonomous rational agents having incentives to interact with each other, share information and expertise, and collaborate [7]. Within a community, agents can collaborate to offer services to end-users or to other services outside the community. Each community is managed by a *controller*, an autonomous agent that is responsible for attracting agents to the community, observing and monitoring the behavior of each member, and dismissing bad and malicious members [8], [9]. Two techniques can be used to settle a community: call-in (agent to controller) and call-out (controller to agent). However, the results presented in this paper are independent from which technique is used. The advantages of having communities are multiple, for instance increasing the collaboration between the members, increasing their visibility and exposure towards the users, managing security centrally, etc. [9]. Agents providing services to the users have incentives to join and stay in communities enjoying good reputation, because this allows them to get more requests from the users and thus generate more profits [10].

One of the efforts in addressing communities reputation is establishment of a maintenance based trust mechanism that allows the controller agent to update the reputation of the members by assigning either rewards or penalties upon each maintenance activity [11]. The term maintenance in this paper refers to the update activity performed by the controller to adjust the aggregated reputation value of each autonomous agent in a community, and should not be confused with the maintenance in the software life cycle. The controller agent can decide to dismiss an agent from the community if its reputation or performance are bad. Reputation assessment as mentioned in [12] relies heavily on different parameters that are gathered about each agent in the community. However, these parameters are assumed to be reliable, which is not necessarily the case. One of the principal causes of questioning the unreliability of the parameters provided is the fact that in open multi-agent systems, agents are rational in their behaviors and could have incentives to perform collusion, which can be defined as malicious acts performed by agents with other agents in order to take advantage of the system vulnerability and mislead the controller agent with fake feedback[2].

[1] In this paper, community and agent-based community are used interchangeably.

[2] Here we assume a feedback system where agents send their feedback to evaluate services provided by other agents.

B. Problem Definition and Contributions

Under direct (where agents are communicating directly with each other) and indirect (where agents are referring to other agents to get information about a given agent) trust evaluation, it has been shown that agents have good assessment of trust when they update their belief sets and when the controller agent checks their behaviors frequently, especially in dynamic environments with many fickle agents and where agents can join and leave the system dynamically [12]. This periodic maintenance based on periodic check is an important aspect in evaluating trust and reputation as it helps alleviate collusion and discourage agents to act maliciously where the controller agent updates its trust belief about each agent in the community based on the agents recent behavior. This maintenance allows increasing the reliability of the reputation mechanism. However, autonomous agents can take advantage of predictable maintenance phases of the controller agent (i.e. predictable check moments). Thus, the problem is how to help the controller agent choose moments in time during a certain interval in order to perform the behavior check by monitoring the agents behavior, and so the maintenance update to reduce malicious acts as much as possible. The key problem is then making the maintenance update phases hard to predict by the agents.

In fact, in open multi-agent systems, agents are autonomous and rational in their behaviors. Consequently, they can decide when it is in their best interest to act maliciously, or collude, in order to mislead the controller and stay in the community. In other words, driven by their motivations, malicious and bad members aim at staying in the community and generating profits for services they provide by deciding to deceive the controller agent with fake feedback. To address this problem, two constraints need to be emphasized. The first one is that the controller agent is not willing to perform the monitoring and check at every single moment. This will require many resources and cause overhead in the system and lack of efficiency in terms of overall performance as each web service should be monitored all the time. The second constraint is related to the number of feedback needed to perform the update; it is inefficient to perform maintenance and reputation adjustment based on very few feedback. Furthermore, it is not recommended for the controller agent to perform one single maintenance check in a large time interval. This will give more motivations to agents to collude in that time interval and will not help the controller agent minimize the collusion scenarios in the community.

In this paper, considering these two constraints, we present a game theoretic approach that will serve as algorithm for the controller to decide when the check should be done during a certain time interval. For instance, if the check moment is decided by the algorithm to be t_x, the period during which the agents behaviors are monitored and checked by the controller is $[t_x - \alpha/2; t_x + \alpha/2]$ where the variable α, fixed by the system designer, represents the size of the time window to be investigated relative to the log file where the feedback are recorded. By fixing α, only the size of the checking interval is fixed, but not the interval itself, which is decided by the algorithm so that it is hard for the agents to predict it. Fixing α allows the controller to have a larger view of the agents behavior and to put this behavior in a more general context, which should be the same for all agents to guarantee a form of fairness. In the paper, we also experimentally analyze and compare the results of this algorithm (with different values of α) with other algorithms following three different probabilistic distributions: uniform, normal, and Poisson, and this in both static and dynamic environments.

C. Paper Organization

The rest of the paper is organized as follows. In Section II, relevant related work are discussed. In Section III, a game theoretic analysis of the problem we are addressing is presented. The overall controller agent's scheduling problem and our game theoretic framework are described. In Section IV, experimental and simulation settings are introduced. Section V shows the simulation results. Finally, Section VI concludes the paper.

II. Related work

Since our research focuses on improving the quality of reputation mechanisms in a community of autonomous agents, it is important to discuss the strategies players can commit to. In [13], the authors studied the problem of prisoners dilemma game where each player cooperates or defects. Nash equilibrium analysis shows that the steady state of each player is to defect. The authors proved that it is possible to reach Pareto optimality in which each player is cooperating using conditional joint action learning over several rounds. The limitation of this work is that the game payoff structure has to satisfy certain conditions, which are not always true in open multi-agent systems. If we try to apply this methodology to our game of interest, we will be ending up with collude strategy profile of a rational agent and perform maintenance check of the controller agent at nearly every moment. Besides, during each round, each player commits to a pure strategy profile, which is not the case in a time interval since on the one hand agents may collude in some moments and not in others, and on the other hand, the controller agent is performing a maintenance check in some moments and not in others. This corresponds to a mixed strategy profile of both players.

When dealing with collusion, which can be defined as fake feedback resulting from malicious act between service providers and service consumers in order to alter the truthful performance, it has been proved in [14] that when truth telling strategy is encouraged by providing incentives, the proposed trust model outperforms competitive models in the literature in terms of cumulative utility gained and good selection percentage. This improvement is partially achieved thanks to the reputation assessment

process that makes use of other agents opinions about service providers. In terms of fickle selection percentage, the proposed model performs better than the other models since it allows agents to update their belief sets regularly and allows customer agents to be flexible in their decision making process. This trust framework relies heavily on aggregation of feedback, which are reputation parameters assumed to be reliable. Furthermore, this work does not explain what regularly updates mean in terms of deciding when the update should be performed.

To address the same collusion problem, the authors in [15] considered the use of incentive-compatibility, which means rational agents find it in their best interest to report the truth, which encourages agents to act truthfully and avoid colluding. The paper showed that under specific constraints, it is possible to compute payment mechanisms in order to provide the right amount to reviewers whenever they are asked to rate a given service in both cooperative and non-cooperative settings. The main result is that achieving collusion resistance as the only strategy equilibrium is possible under specific circumstances such as specific number of feedback given that the payment amount has to satisfy certain criteria. However, the work does not consider mixed strategies, which means agents can still collude if they adopt efficient mixed strategies. Furthermore, the paper experimentally proved that the relative cost of the mechanism increases exponentially as colluding fractions increase in partial coordination scenarios.

In [12], a retrospect trust adjustment has been investigated to help agents assess other agents based on three dimensions: direct trust assessment, indirect trust assessment based on trustworthy agents, and indirect trust assessment based on referee agents. Contributions of this research lie in mutual interactions between agents and update of their trust beliefs based on the final results in order to assess the credibility of the trustee agent in the so called maintenance phase. However, although the work is being investigated on the optimization part of the maintenance phase in order to make it more adaptable to different situations, the problem of determining the suitable moment to perform the update has not been addressed. In [16], the authors suggested that user perception is not enough to compute service reputation, but how trustworthy the provider has been in satisfying the service level agreement should be accounted for, which has been measured through a metric called verity. However, monitoring and updating this verity has not been investigated.

In terms of managing quality of service (QoS) in communities, several proposals have investigated the possibility to implement a managerial block towards monitoring services. In [17], a multi and cooperative broker architecture for service selection has been proposed[3]. Each broker manages services in its domain and shares information about these services with other brokers. Furthermore, the monitoring policy proposed in [18], which suggests separating monitoring from management activities by dedicating a specific community to host monitors, has allowed online detection of violations (i.e. web services are not operating as expected). This has also reduced the overhead of other individual communitiess managers. A tuning of this separation strategy for QoS improvement is presented in [19] where the managerial community implementation has been extended to handle selection, communication, monitoring, adaptation, and load balancing on a periodical basis or upon request. The authors showed that QoS attributes such as response time and availability are getting improved with this managerial community implementation. QoS management has also been analyzed in [20] where a new architecture providing advanced management functionalities has been developed. These functionalities include extending the service description with QoS-centered annotations, including a validation process to test the service interface and the level of QoS that can be provided, supporting QoS negotiation, and monitoring the provided QoS.

All these proposals have proved that trust frameworks work well when previous behaviors are checked and maintenance is performed periodically. However, the problem is in the maintenance phase, which is related to learning and predictability. Because agents have the opportunity to study the controller behavior over time before joining the community and decide what strategy to adopt, they have better chance to take advantage of a predictable controller agent scheduling time if it follows certain pattern. This will give the system higher degree of vulnerability and higher risk of providing fake feedback. Hence, reputation will be less reliable and the trust value of users towards that community will be dramatically minimized. Therefore, our problem can be formalized as a problem of designing a scheduling algorithm allowing the controller agent to plan monitoring, check, and trust update activities, so that the pattern becomes hard to predict.

III. Game Theory Analysis

This section presents first the general overview of the problem through the flow chart of the controllers algorithm to schedule the maintenance activity. Next, we explain in details the game theoretic framework between the controller agent and community members. In fact, the controller agent is facing two challenges. First, this controller needs to commit to a schedule before the other agents in the community do. Because the strategy the controller commits to can be observed by the community members, the game we are modeling is a Stackelberg game where the controller is the leader and community members are the followers [21], [22]. Second, the controller agent is performing maintenance of different community members having different types (i.e. different probabilities of acting maliciously). This implies that we are modeling a Bayesian Stackelberg game [23], [24], [21], [22].

[3]A broker can be seen as a community.

A. Flow Chart of the Scheduling Problem

As previously mentioned, the controller agent is a specialized and devoted autonomous agent to observe and monitor the behaviors of community members. The monitoring scheme we use in this paper includes two main activities. The first one, similar to the monitoring process described in [19] (except that the monitoring is not done whenever serving users requests), is about assessing the performance of the agent and comparing the QoS revealed before joining the community against the QoS provided during the period of check. The agents performance is assessed using some statistics extracted from the log file managed by the controller, such as the number of served requests during the period of check, portions of high/low load, number of delegated requests, and number of queued requests from clients. The second monitoring activity is based on assessing the correlation between the QoS provided and the feedback received during the monitoring period. For instance, if the feedback rates do not reflect the actual provided QoS (e.g. bad or normal QoS with very high rates), the controller should take actions to penalize the monitored agent. This is done by investigating the log file by the controller agent, which contains feedback values sent by users or other agents relative to each agent in the community about the service this agent is providing.

Figure 1 depicts the scheduling problem flow chart, which is composed of five main steps. The first step is observing the history of the log file (history file) to get for instance the number of requests made for each agent as well as the obtained QoS, and this is during specific moments (arrow 1). The second step is constructing the Bayesian Stackelberg game from remarks and investigation of the history file (arrow 2). This step includes determining the payoff matrices R and C of the two players according to their possible strategies. The third step lies in formalizing the scheduling problem as an optimization problem using the payoff matrices defined in the second step and applying a multiple-integer linear programming technique called DOBSS [25] in order to solve that Bayesian Stackelberg game, which considers probability distributions of the types of the community members (arrow 3). The fourth step is about selecting among the probability distributions of the controllers pure strategies those that are different than zero, which correspond to optimal schedule (arrow 4). Then the controller follows these moments to perform its maintenance (arrow 5). The details of the second, third, and fourth steps are provided in this section.

B. Game Theoretic Framework

Two players are considered in our community-based multi-agent system: the controller agent and community members. The strategy profile of the controller agent is either to perform a maintenance check in a specific moment during a given time interval or not. The strategy profile of each member is to act truthfully or maliciously,

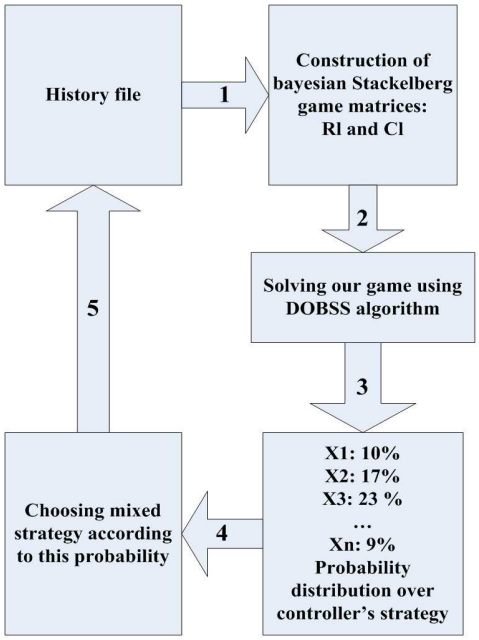

Fig. 1. Flow chart diagram of the controller agent's scheduling problem.

which corresponds to colluding, in a specific moment of time.

Our objective is to allow the controller agent to schedule moments in time for maintenance update. It is crucial to mention three factors that determine the constraints of our solution.

- The first one is that the controller agent that takes the feedback file under surveillance as defined in [11], is not willing to investigate this file at every moment in time because of limited resources and capacities. For example, during 100 minutes of activities during which feedback submissions are performed at every minute, the controller agent does not have the capacity to perform the investigation and maintenance at every single minute. On the one hand, there are cases where communities might not have activities for a period of time. On the other hand, in a busy environment, it is not wise to schedule very few periodic maintenances since the chances of collusion are very high.
- The second factor is the type of community members. In open multi-agent systems, agents differ in there strategies in terms of when to perform the collusion as well as how damaging the fake feedback is compared to the real reputation parameter. For example, we may have agents that collude 2% of the time and others may be willing to collude 90% of a time interval. Even if we have two colluding agents with the same percentage, say 20%, one agent may be willing to collude at moments 1 and 2 while the other at moments 8 and 9 in an interval of 10 moments. Also, we may have agents that collude with service consumers, as defined in [11], which are agents that continuously seek for services pro-

vided by some other agents, by providing very high feedback that raises great degree of suspicion like 10/10 or a feedback that meets the average like 7/10, which is hard to detect. Community members are also assumed to be rational in their behaviors [26]; their objective is to maximize their own payoffs.
- The third one is that the controller agent needs to avoid choosing the same moment of check during subsequent time intervals. The predictability issue has a major drawback of making the learning process of the controllers strategy overtime an easy task for malicious agents. Hence, those agents will have better opportunity to schedule their collusion in order to maximize their benefits, which results in tremendous damage to the communitys reputation.

To be focussed on the schedule problem, we assume that when the check is performed, the controller has the capacity to detect malicious acts if performed by the checked member with full accuracy using the monitoring scheme. The game between the controller and community members can then be formalized as follows, which corresponds to the phase of constructing the Bayesian Stackelberg game matrices R and C in Figure 1:

- **Malicious act penalized**: this is the case where the controller agent performs the maintenance (the check) and penalizes the malicious agent by dismissing it from the community. The controller agent gets a positive payoff $+\pi$ as we assume that the users are paying the controller for its work, which is making the community secure. This payoff already includes the cost of performing the maintenance. The community member gets a major penalty: $-(N+L)P$, where N being the average number of possible requests the agent gets if not fired, L the expected increase in the number of requests the agent can get if not penalized, and P the reward of that agent for serving each request. $N+L$ is then the expected number of requests the member can get if not fired and $-(N+L)P$ represents the loss undergone by the member because of being fired from the community.
- **Malicious act ignored**: this is the worst case for the controller agent in which a community member is colluding but the controller did not prevent it by performing the maintenance check. This corresponds to the controller payoff of $-\pi$ (as users will not pay the controller since the system is not secure) and a reward $+(N+L)P$ for the community member (L corresponds to the increase resulting from collusion to promote the member's reputation).
- **Good performance ignored**: this is the case where the controller agent is not performing maintenance and community member is not colluding, but performing well in terms of its reputation increase. This corresponds to the payoff $+\lambda$ for the controller, which is less than $+\pi$ (as the community is secure and users are happy, but no update has been made to justify a higher payoff), and a payoff of $+(N+L)P$ for the member as the member is benefiting from the increase L.
- **Good performance recognized**: this is the case where the controller agent is performing maintenance and community member is not colluding, but performing well in terms of its reputation increase. This corresponds to the payoff $+\pi$ for the controller (as the community is secure and updated and users are happy), and a payoff of $+(N+L)P$ for the member.
- **Bad performance penalized**: this is the case where the controller agent is penalizing the member by dismissing it from the community not for collusion, but for bad performance. The controller payoff is $+\pi$ since an effort has been done to make the community more reputable, which increases the quality of service for the users. The corresponding payoff of the member is $-NP$ because of being fired, which means requests are lost.
- **Bad performance ignored**: this is the case where the controller agent is not doing the maintenance and the member is not colluding, but doing bad in terms of reputation. The controller payoff is $-\pi$ and the corresponding payoff of the member is $+NP$.

Table I shows the payoff structure for our game theoretic framework at a given moment in time with the column player being the controller agent and the row player a community member.

	Perform maintenance	Ignore maintenance
Collude	$(-(N+L)P, +\pi)$	$(+(N+L)P, -\pi)$
Not collude with good performance	$(+(N+L)P, +\pi)$	$(+(N+L)P, +\lambda)$
Not collude with bad performance	$(-NP, +\pi)$	$(+NP, -\pi)$

TABLE I
STRATEGY PROFILES AND PAYOFF STRUCTURE OF OUR GAME

C. Finding the Optimal Schedule

As pointed out earlier, community members differ in their collusion strategies in terms of collusion time. For instance, in a time interval of five moments, agents A and B with 20% collusion degree may not necessarily collude at the same moment; agent A may collude at moment 2 while agent B may collude at moment 3.

Our solution concept lies in allowing the controller agent (i.e. the leader) to find an optimal scheduling algorithm to commit to, given that the community members may know this strategy when choosing their own strategies. We build on the frameworks developed in [27], [21], [22], and [28] to define our mathematical apparatus. Notice that these proposals have been defined in a different context (airport security) using different parameters and different mathematical formulation than the ones we use here.

First, we present this mathematical reasoning with one community member type and then generalize the solution to many types. Let us now introduce the problem parameters. During a given interval of time, there is a

given number of discrete moments where the controller has to play. In each moment, the controller has two strategies: perform the check (strategy 1) or not (strategy 0). Similarly, there is a given number of discrete moments where the member has to play by choosing between acting truthfully (strategy 1) and acting maliciously (strategy 0). Let x and y be the pure strategy vectors of the controller and community member respectively. x_i is the strategy chosen by the controller at moment i and y_j is the strategy chosen by the member at moment j. Thus, $x_i \in \{0,1\}$ where $x_i = 1$ (resp. $x_i = 0$) means strategy 1 (resp. 0) is played by the controller at moment i. $y_j \in \{0,1\}$ has the same meaning for the member. Let I be the index set of acting moments of the controller agent; J be the index set of acting moments of the member; and R and C be the payoff matrices such that R_{ij} is the controller agents reward and C_{ij} is the members reward where the controller agent and member are respectively playing strategies x_i and y_j . $|I|$ and $|J|$ are the cardinalities of I and J respectively, where $|J|$ is function of $|I|$ and α (see Section I-B). In this paper, we use the following linear function:

$$|J| = (\alpha + 1)|I| \tag{1}$$

By fixing the strategy vector of the controller (i.e fixing x), the community member has to solve the optimization problem (2) to find its optimal response to x.

$$\begin{aligned} \arg\max_{y} \quad & \sum_{j\in J}\sum_{i\in I} C_{ij}x_i y_j \\ s.t. \quad & \sum_{j\in J} y_j \leq |J| \\ & y_j \in \{0,1\} \qquad j \in J \end{aligned} \tag{2}$$

The objective function is maximizing the member's payoff, and the constraints make feasible any member's mixed strategy. When solving this problem, it is obvious that for a given j, if $\sum_{i\in I} C_{ij} > 0$ then $y_j = 1$. Thus, we obtain:

$$\begin{aligned} \sum_{i\in I} C_{ij} > 0 \quad &\Rightarrow \quad y_j = 1 \\ \sum_{i\in I} C_{ij} \leq 0 \quad &\Rightarrow \quad y_j = 0 \end{aligned}$$

Consequently, we obtain the following equation:

$$\sum_{i\in I} C_{ij}y_j = \max(0, \sum_{i\in I} C_{ij}) \quad j \in J \tag{3}$$

Given a fixed strategy vector y of the community member, The controller must choose its own strategy x that is best response to y. Associated with the strategy of performing check (strategy 1) at moment i is a cost t_i, which is proportional to the payoff. The maintenance cost reflects, in some extent, the systems scalability and amount of computational resources needed to perform the check. Thus, the optimization problem the controller has to solve is formalized as follows:

$$\begin{aligned} \arg\max_{x} \quad & \sum_{i\in I}\sum_{j\in J} R_{ij}(1-t_i)x_i y_j \\ s.t. \quad & \sum_{i\in I} x_i \leq |I| \\ & \sum_{i\in I} x_i > 0 \\ & x_i \in \{0,1\} \qquad i \in I \\ & 0 < t_i < 1 \qquad i \in I \end{aligned} \tag{4}$$

The objective function maximizes the controller's payoff by maximizing the reward R_{ij} and minimizing the cost t_i. The second constraint forces the controller to have at least one check moment. From this problem, we obtain the general problem the controller has to solve (i.e. without fixing the member's strategy y) as follows:

$$\begin{aligned} \arg\max_{x,y} \quad & \sum_{i\in I}\sum_{j\in J} R_{ij}(1-t_i)x_i y_j \\ s.t. \quad & \sum_{i\in I} x_i \leq |I| \\ & \sum_{i\in I} x_i > 0 \\ & x_i \in \{0,1\} \qquad i \in I \\ & \sum_{j\in J} y_j \leq |J| \\ & y_j \in \{0,1\} \qquad j \in J \\ & 0 < t_i < 1 \qquad i \in I \end{aligned} \tag{5}$$

The problem with the optimization problem (5) is that it does not consider the optimal solution y of the member. This mens, we need to add a constraint forcing the obtained solution for the controller's strategy vector x to be associated with the optimal solution for the member's strategy vector y. This is archived by integrating the Equation 3 as a constraint. Therefore, the controller's problem becomes:

$$\begin{aligned} \arg\max_{x,y} \quad & \sum_{i\in I}\sum_{j\in J} R_{ij}(1-t_i)x_i y_j \\ s.t. \quad & \sum_{i\in I} x_i \leq |I| \\ & \sum_{i\in I} x_i > 0 \\ & x_i \in \{0,1\} \qquad i \in I \\ & \sum_{i\in I} C_{ij}y_j = \max(0, \sum_{i\in I} C_{ij}) \quad j \in J \end{aligned} \tag{6}$$

$$\sum_{j\in J} y_j \leq |J|$$
$$y_j \in \{0,1\} \qquad j \in J$$
$$0 < t_i < 1 \qquad i \in I$$

The first, second, and third constraints enforce a feasible mixed strategy for the controller, and the fifth and sixth constraints enforce a feasible pure strategy for the community member.

In order to generalize this problem to consider multiple member types, we need to consider the following additional parameters, which makes the game Bayesian:

- p^l: the probability of encountering a member of type l;
- R^l and C^l: the payoff matrices such that R^l_{ij} and C^l_{ij} are the rewards associated with the strategies x_i and y_j of the controller agent and community member of type l respectively;
- y^l_j: the pure strategy of the member of type l at moment j;
- L: the set of members types.

Hence, the problem (6) becomes:

$$\begin{aligned}
\arg\max_{x,y} \quad & \sum_{i\in I}\sum_{l\in L}\sum_{j\in J} p^l R^l_{ij}(1-t_i)x_i y^l_j \\
s.t. \quad & \sum_{i\in I} x_i \leq |I| \\
& \sum_{i\in I} x_i > 0 \\
& x_i \in \{0,1\} \qquad i \in I \quad (7)\\
& \sum_{i\in I} C^l_{ij} y^l_j = \max(0, \sum_{i\in I} C^l_{ij}) \qquad j \in J \\
& \sum_{j\in J} y^l_j \leq |J| \\
& y^l_j \in \{0,1\} \qquad j \in J \\
& 0 < t_i < 1 \qquad i \in I
\end{aligned}$$

Notice that this optimization problem is a quadratic programming problem since we have two unknown integers to solve $x_i y^l_j$. We can make the program linear using the following change of variables: $x_i y^l_j = z^l_{ij}$. Consequently, we obtain the following equivalent problem:

$$\begin{aligned}
\arg\max_{x,y} \quad & \sum_{i\in I}\sum_{l\in L}\sum_{j\in J} p^l R^l_{ij}(1-t_i)z^l_{ij} \\
s.t. \quad & 0 \leq \sum_{i\in I}\sum_{j\in J} z^l_{ij} \leq |I||J| \\
& 0 < \sum_{i\in I} z^l_{ij} \leq y^l_j |I| \qquad j \in J \\
& 0 \leq \sum_{j\in J} z^l_{ij} \leq |J| \qquad i \in I \\
& z^l_{ij} \in \{0,1\} \qquad (i,j) \in I \times J \quad (8)\\
& \sum_{i\in I} C^l_{ij} y^l_j = \max(0, \sum_{i\in I} C^l_{ij}) \qquad j \in J \\
& \sum_{j\in J} y^l_j \leq |J| \\
& y^l_j \in \{0,1\} \qquad j \in J \\
& 0 < t_i < 1 \qquad i \in I
\end{aligned}$$

It is worth mentioning that the parameter p^l (the probability of encountering a member of type l) encapsulates the probability of receiving requests from that member. Because we are solving multiple-integer linear programming (with multiple members), the overall requests distribution is considered in the problem 8. Furthermore, The equivalence of the problems (7) and (8) is straightforward by construction as the objective function of (8) is a direct transformation of the objective function of (7) and each constraint in (8) is directly obtained from a constraint in (7) using the change of variables: $x_i y^l_j = z^l_{ij}$. The objective function maximizes the payoff of the controller agent against different community members having different types. The output of this program is the mixed strategy for the controller, which dictates the strategy to choose at each moment (i.e. performing the check or not) given that the community members are playing the best strategies (i.e. best responses). Thus, the controllers strategy vector we obtain is the optimal schedule we are seeking, which corresponds to stage four in the controllers flow chart (Figure 1) explained earlier. To show how the dynamism is captured in this optimization problem, let us consider the following example with $|I| = |J| = 5$ (i.e. acting over 5 moments). Assuming that the solution the algorithm provides for x is (1; 0; 0; 1; 1); this means the controller should perform the check at moments t_1; t_4, and t_5. This will correspond to the best strategies of the member. Consequently, most likely the member will play honest all the time or collude during times where the check is performed (as the controller is playing assuming that the member is maximizing its payoff). In fact, the following theorem holds.

Theorem 1: The optimal solution given by solving the problem 8 is Pareto optimal.

Proof: Solving problem 8 results in an optimal solution for the controller in terms of payoff given that the community member is maximizing its payoff. Consequently, any change in the controllers strategy will not make this controller better off without making the member worse off (the controller can gain more by detecting a collusion only if the member gains less by being detected as malicious). Reversely, any change in the member strategy can make this member better off (malicious action not detected) only if it makes the controller worse off. Thus, there is no Pareto improvement that can be made, so we are done. ■

Because it is based on solving a linear optimization problem, the periodic monitoring approach we propose in this paper is cost-effective compared to other approaches where monitoring is done whenever an agent is providing

a service. In the next section, we will show how our approach outperforms other periodic approaches where the monitoring moments are randomly chosen or selected according to given probabilistic distributions.

IV. Simulation settings

The simulation consists of interactions between users and community members providing services, and according to a certain schedule the controller agent performs a maintenance update, which includes updating the belief set of the controller agent about each agent in the community. Each agent is checked for the performance stored in a reputation feedback file during a given interval and gets penalized by leaving the community in case the average reputation calculated is less than a given threshold defined by the controller agent or greater than another predefined threshold. Therefore, the agent is being penalized for performance reasons in the first case and for collusion in the second. For the sake of simplicity but without losing generality, requests are being judged by user agents based on response time only as reputation parameter.

At every moment, certain members may collude once. The number of requests received by each member is directly proportional to the maintenance results. Once a maintenance update is performed, each agent is being rewarded with an increase in terms of requests that can be received during that time interval. For example, given that agents A and B are ranked first and second, respectively, in terms of performance, A will be given 1 more request than B during that time interval.

To compare our approach with other stochastic techniques, the simulation of the controller's behavior is implemented in five forms. The first form is the fixed check time; the controller agent chooses fixed check moments in time at every round and it gets repeated for many rounds. The second form is a uniform check time; the controller agent chooses check moments in time using a uniform random distribution at the beginning of every round. The third form is a normal check time; the controller agent chooses moments in time to perform the check using a normal random distribution at the beginning of every round. The fourth form is a Poisson check time; the controller agent chooses check moments in time using a Poisson random distribution at the beginning of every round. The fifth form is the adoption of a mixed strategy that is based on our game theoretic framework of the Bayesian Stackelberg game; the controller agent chooses check moments in time based on the solution provided by the decomposed optimized Bayesian Stackelberg solver, or DOBSS [25] of the optimization problem 8.

Our simulation was conducted on two main settings: static settings and dynamic settings. To make the system realistic, different member types having different collusion degrees are considered. Table II shows the distribution of the collusion degree. For instance, 16% of the population have probability between 0.7 and 1 to collude.

A. Static Settings

In this experiment, we analyzed the behavior of the five algorithms in a static environment of 50 community members and 100 users. This experiment analyzes two facts: the first one is the starting moment in time the controller agent -under different check time algorithms- is able to reduce the fake feedback percentage of 5% or less; the second one is which algorithm allows the controller agent to get the minimum percentage value of collusion at the end of the simulation.

Collusion degree distribution interval	Distribution percentage
0	20
]0, 20[	12
[20, 50[	34
[50, 70[	18
[70, 100]	16

TABLE II
Distribution of the collusion degree in our static simulation

B. Dynamic Settings

In this experiment, we analyzed the behavior of the five algorithms in a dynamic environment, an environment where agents join and leave the community dynamically. Two facts are investigated: the first one is how the controller agent -under different check time algorithms- can be able to cope with the environment change; the second one is how the controller agent can minimize the percentage of collusion or fake feedback at the end of the simulation.

V. Simulation Results

Our results are discussed according to two environment types: static and dynamic

A. Static Settings

According to the results shown in Figure 2 (in the five graphs the X axis represents moments in time while the Y axis represents fake feedback percentage), two conclusions can be inferred. The first one is that our game theoretic approach with the DOBSS algorithm manages to reduce the number of collusion to less than 5% earlier than the other algorithms. Precisely, our algorithm achieves this percentage at time moment 40 while fixed, normal, Poisson and uniform achieve this percentage at moments 45, 53, 68, and 45 respectively. The second conclusion is that our approach manages to have the smallest number of collusion at the last moment of the simulation compared to the other algorithms.

B. Dynamic Settings

As in the static settings, our simulation consists of analyzing, over 10 different runs of each algorithm, the percentage of fake feedback in the community. Figure 3 shows the percentage of collusion at each moment in time

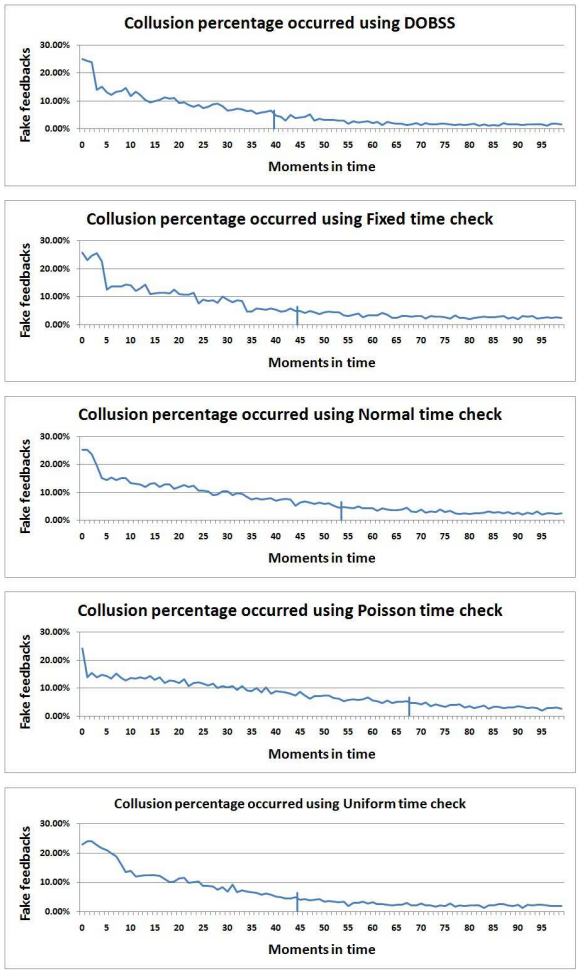

Fig. 2. Comparative results between the scheduling algorithms in a static environment

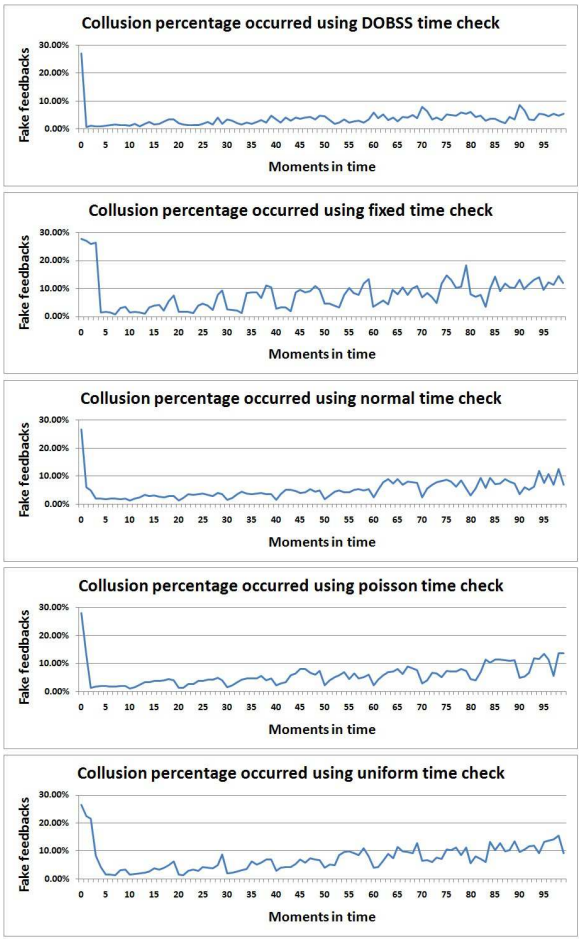

Fig. 3. Comparative results between the scheduling algorithms in a dynamic environment

of these scheduling algorithms (once again in the five graphs the X axis represents moments in time while the Y axis represents fake feedback percentage).

Before pointing out the results from our dynamic simulation, it is important to explain the sudden drop in the very first moments across the five algorithms. As mentioned in the previous section, at the beginning of the simulation community members receive the same number of requests. Once the maintenance is performed, the controller agent, regardless of the scheduling algorithm, rewards the highly ranked members with an increase in terms of requests per interval. Therefore, since the malicious agents performed collusion to mislead the controller agent before the maintenance, they get more requests after the maintenance has been performed. Hence, the number of truthful feedback increases after the maintenance phase while the number of fake feedback is nearly the same. This explains the sudden drop in terms of fake feedback percentage after the first maintenance process.

The graphs of Figure 3 show that our approach is more stable in terms of reducing the collusion percentage despite the environments dynamism. It also manages to get the minimum number of collusion, especially at the last moments starting from the moment in time 89 till the end of the simulation. It also shows that the controller agent under DOBSS algorithm was able to react quicker than the rest of the algorithms concerning penalizing the malicious members before they have chance to collude.

VI. Conclusion

Trust and reputation have gained significant major research curiosity in the field of community-based autonomous and multi agent systems. Many attempts towards the improvement of trust mechanisms have been made. However, the problem of scheduling the maintenance activity has not been addressed. In this paper, we provided a solution by modeling the interaction between agents through game theoretic foundation. In this scenario where a controller agent is supervising the reputation mechanism, we investigated the periodic maintenance performed by this controller agent in order to update the belief set about each agent. Our contribution is divided into two parts. First, we introduced a game theoretic model between the controller agent and community members with respect to the phenomenon of collusion and formalized the scheduling problem as an optimization problem. Second, we analyzed the efficiency of our approach against two metrics and compared this efficiency with four other stochastic techniques.

As future work, we are planning to investigate the possibility of extending the scheduling quality through three factors. The first one is to automatically determine the size of the interval time, not just the moments where the controller agent should perform its maintenance operation. The second one is to consider the possibility of initiating the collusion from the user and not the community member, which implies including users in our game framework for the controller agent to make better decisions. The third one is to consider different probabilities of the controllers accuracy, which implies using more parameters and introducing sophisticated solving techniques.

REFERENCES

[1] H. Chen, "Personalitys influence on the relationship between online word-of-mouth and consumers trust in shopping website," *Journal of Software*, vol. 6, no. 2, pp. 265–272, 2011.

[2] D. Cheng, J. Han, and Y. SongIs, "Value sufficient? empirical research on the impact of value and trust on intention," *Journal of Software*, vol. 6, no. 1, pp. 124–131, 2011.

[3] P. Matt, M. Morge, and F. Toni, "Combining statistics and arguments to compute trust," *In Proceedings of the 9th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 209–216, 2010.

[4] S. Parsons, Y. Tang, E. Sklar, K. Cai, and P. McBurney, "Argumentation-based reasoning in agents with varying degrees of trust," *In Proceedings of the 10th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 879–886, 2011.

[5] M. Singh, "Trust as dependence: a logical approach," *In Proceedings of the 10th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 863–870, 2011.

[6] G. Vogiatzis, I. MacGillivray, and M. Chli, "A probabilistic model for trust and reputation," *In Proceedings of the 9th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 225–232, 2010.

[7] J. Bentahar, Z. Maamar, W. Wan, D. Benslimane, P. Thiran, and S. Subramanian, "Agent-based communities of web services: An argumentation-driven approach," *Service Oriented Computing and Applications*, vol. 2, no. 4, pp. 219–238, 2008.

[8] B. Khosfavifar, J. Bentahar, A. Moazin, and P. Thiran, "Analyzing communities of web services using incentives," *International Journal of Web Services Research*, vol. 7, no. 3, pp. 30–51, 2010.

[9] Z. Maamar, S. Subramanian, Ph.Thiran, D. Benslimane, and J. Bentahar, "An approach to engineer communities of web services: concepts, architecture, operation, and deployment," *International Journal of E-Business Research*, vol. 5, no. 4, pp. 1–21, 2009.

[10] B. Khosravifar, M. Alishahi, J. Bentahar, and P. Thiran, "A game theoretic approach for analyzing the efficiency of web services in collaborative networks," *In Proceedings of the 8th IEEE International Conference on Services Computing (SCC)*, pp. 168–175, 2011.

[11] B. Khosfavifar, J. Bentahar, A. Moazin, and P. Thiran, "On the reputation of agent-based web services," *In Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pp. 1352–1357, 2010.

[12] B. Khosravifar, M. Gomrokchi, J. Bentahar, and P. Thiran, "Maintenance-based trust for multi-agent systems," *In Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1017–1024, 2009.

[13] D. Banerjee and S. Sen, "Reaching pareto optimality in prisoner's dilemma using conditional joint action learning," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 15, no. 1, pp. 91–108, 2007.

[14] J. Bentahar, B. Khosravifar, and M. Gomrokchi, "Social network-based trust for agent-based services," *In Proceedings of the International Conference on Advanced Information Networking and Applications Workshops*, pp. 298–303, 2009.

[15] R. Jurca and B. Faltings, "Collusion-resistant, incentive-compatible feedback payments," *In Proceedings of the 8th ACM Conference on Electronic Commerce (EC)*, pp. 200–209, 2007.

[16] S. Kalepu, S. Krishnaswamy, and S. Loke, "Verity: a qos metric for selecting web services and providers," *Fourth International Conference on Web Information Systems Engineering Workshops, IEEE Computer Society*, pp. 131–139, 2004.

[17] M. Serhani, E. Badidi, A. Benharref, and M. Salem, "A cooperative approach for qos-aware web services' selection," *In Proceedings of the International Conference on Computer and Communication Engineering (ICCCE)*, pp. 1084–1088, 2008.

[18] A. Benharref, M. Serhani, S. Bouktif, and J. Bentahar, "A new approach for quality enforcement in communities of web services," *In Proceedings of the 8th IEEE International Conference on Services Computing (SCC)*, pp. 472–479, 2011.

[19] M. Serhani and A. Benharref, "Enforcing quality of service within web services communities," *In the Journal of Software*, vol. 6, no. 2, 2011.

[20] M. Serhani, R. Dssouli, H. Sahraoui, A. Benharref, and M. Badidi, "Vaqos: Architecture for end-to-end qos management of value added web services," *International journal of intelligent informationn technologies, IGI-Global*, vol. 2, pp. 37–56, 2006.

[21] P. Paruchuri, J. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus, "Playing games for security: an efficient exact algorithm for solving bayesian stackelberg game," *In Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 895–902, 2008.

[22] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus, "Security in multiagent systems by policy randomization," *In Proceedings of the 5th International Conference of Autonomous Agents and Multiagent Systems*, pp. 273–280, 2006.

[23] V. Conitzer and T. Sandholm, "Computing the optimal strategy to commit to," *In Proceedings of the 7th ACM Conference on Electronic Commerce*, pp. 82–90, 2006.

[24] D. Fudenberg and J. Tirole, *Game theory*. MIT Press, 1991.

[25] J. Pita, M. Jain, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus, "Using game theory for los angeles airport security," *AI Magazine*, vol. 30, no. 1, pp. 43–57, 2009.

[26] A. Rubinstein, *Modeling Bounded Rationality*. MIT Press, 1998.

[27] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, M. Tambe, and F. Ordonez, "Computing optimal randomized resource allocations for massive security games," *In Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 689–696, 2009.

[28] J. Pita, M. Jain, J. Marecki, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Parachuri, and S. Kraus, "Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport," *In Proceedings of the 7th International Conference of Autonomous Agents and Multiagent Systems, Industry and Applications Track*, pp. 125–132, 2008.

Mohamed Amine M'hamdi is currently a M.Sc. candidate at Concordia University, Montreal, Canada. He received his B.Sc. degree in computer science from Al Akhawayn University, Ifrane, Morocco, in 2008. His research interests include trust and reputation in multi-agent systems, game theory, and agent computing.

Jamal Bentahar is currently an Associate Professor at Concordia University, Montreal, Canada. He received his Ph.D. in computer science and software engineering from Laval University, Quebec, Canada in 2005 and a M.Sc. in software engineering from ENSIAS, Morocco, in 2000. His research interests include intelligent agents and multi-agent systems, trust and reputation, argumentation, formal logics, model checking, and service computing.