

A Robust Audit Mechanism to Prevent Malicious Behaviors in Multi-Robot Systems*

MyungJoo Ham and Gul Agha

Open Systems Laboratory, Department of Computer Science
University of Illinois at Urbana-Champaign
{ham1, agha}@cs.uiuc.edu

Abstract

Market-based mechanisms can be used to coordinate self-interested multi-robot systems in fully distributed environments, where by self-interested we mean that each robot agent attempts to maximize a payoff function which accounts for both the resources consumed and the contribution made by the robot. In previous work, we have studied the effect of various market rules and bidding strategies on the global performance of the multi-robot system. However, rather than use a central monitoring and enforcement mechanisms, we rely on agents to self-report their actions. This assumes that the agents act honestly. In this paper, we drop the honesty assumption, raising the possibility that agents may exaggerate their contribution in order to increase their payoff. To address the problem of such malicious behavior, we propose an audit mechanism to maintain the integrity of reported payoffs. Our algorithm extends previous work on preventing free-riding in peer-to-peer networks. Specifically, we consider locality and mobility in multi-robot systems. We show that our approach efficiently detects malicious behaviors with a high probability.

1 Introduction

Physical multi-agent (multi-robot) systems that require fully-distributed coordination strategies have been proposed for applications such as surveillance [14], search and rescue [8], mine sweeping [7], and space exploration [2]. As the scale of such multi-agent systems grows, a centralized approach is not feasible. We have previously studied a decentralized market-based approach for fully-distributed coordination of self-interested robots (or agents). Specifically, we analyzed the effect of various market rules (different auction mechanisms along with a task swapping mecha-

nism) and bidding strategies on performance as measured by time to complete a given task, distance traveled by robots, communication overhead, auction delays, and imbalance of workload between robots [6, 5, 10]. Although robot agents are assumed to be self-interested, these studies also implicitly assume that the agents act honestly.

In this paper, we drop the honesty assumption. This introduces the possibility that a robot agent may make false claims. For example, if there are multiple organizations that operate robot agents sharing a mission and there may be inter-organization cooperations with pay-offs, a *cheating* agent or a team of cheating agents may claim payoffs without serving other agents, provide counterfeit payoffs to receive other agents' services, or damage the performance of the system by providing false information (e.g., by reporting bogus tasks). In this paper, we try to mitigate such cheating in the targeted multi-robot systems.

We observe that this problem is similar to the widespread problem of free-riding in peer-to-peer (p2p) networks (particularly in pure-p2p networks which do not have central controls: about 90% peers in Gnutella are found to be free-riders [1]). It turns out that preventing such deception is not easy without central controls: even if the peers measure the amount of contribution and consumption, some peers may be able to deceive the system [4].

We have previously proposed ARA audit mechanism to prevent free-riding in pure-p2p networks [4]. ARA is fully distributed, has a low overhead, and detects all but "minimal" cheating. ARA works by providing a credit system to measure the contribution and consumption of each peer and the capacity of being served is determined by the credit value. A peer can be required to contribute in order to be served (free-riding may be prevented). ARA also provides an audit mechanism to stochastically verify credit values in order to prevent cheating that is intended to falsely inflate credit values.

Both pure-p2p systems and multi-robot systems consist of fully-distributed and self-interested autonomous agents. Because of this similarity, it is a reasonable conjecture that

*This work is partially funded by the National Science Foundation under grants CNS 05-09321 and CMS 06-00433.

an audit mechanism for pure-p2p systems may be used to audit robots' activities in multi-robot systems. However, there are some key differences between the two systems:

- how to measure services and resources is not as obvious in multi-robot systems as it is in p2p networks;
- robot agents have limited communication ranges and there is no network overlay as in p2p which enables arbitrary pairs of agents to communicate; and,
- robot agents are mobile.

Because of the above differences, we cannot directly use ARA for multi-robot systems. In this paper, we develop a *location-* and *mobility-*aware variant of ARA with a payoff function. Moreover, because multi-robot systems usually use smaller computing devices than normal p2p users who are on PCs, we ensure that the overhead required by our audit scheme is smaller.

Note that our algorithm is probabilistic. Although there is a possibility that some cheating is not detected by ARA, we assume that if the probability of detection is sufficiently high and a suitable penalty is imposed (e.g., being excluded from further participation, not receiving payoffs from the system, etc.), agents will avoid cheating.

We study our algorithm in the context of large-scale multi-robot (physical multi-agent) systems for the *search and rescue* (SR) problem. This is partly motivated by the fact that we have carried out large-scale experiments on such systems [5]. In order to implement the audit mechanism, we add by a credit system that provides payoffs for inter-agent services. These payoffs represent the amount of contribution made by an agent minus the resources of other agents consumed by the agent. The audit mechanism is then used to verify the integrity of the reporting.

The contributions of this paper are as follows. To the best of our knowledge, it is the first study of the problem of detecting cheating in a system of mobile, autonomous physical agents. The algorithm proposed is decentralized. An analysis of detection rate and overhead is provided to show that the algorithm is efficient and effective.

The paper is organized as follows. Section 2 briefly compares our work with other approaches. Section 3 describes our assumptions. Section 4 discusses the design of the credit system, the design of integrity verification mechanisms, and shows how the mechanisms work. Section 5 shows the performance including detection rate and overhead. The last section concludes this paper with a discussion of future work.

2 Related Work

As mentioned in the introduction, we have not found any previous research on how the integrity of self-interested

agents in cooperative systems may be ensured. Several approaches to verify the contributions of peers in p2p networks have been proposed. Some p2p networks including eMule, BitTorrent, eDonkey, Pruna, and others, throttle peer's resource consumption based on an assessment of a peer's contribution. However, assessing a peer's contribution is difficult. The value determined can be compromised, as we show in [4].

An alternate is to use a reputation-based approach, e.g. [12, 9]. However, in these approaches, the system relies on human intervention; i.e., users need to determine the quality of given services and notify the system of the quality. In case of multi-robot systems, we cannot expect continuous human intervention. Other proposed approaches involve measuring the actual amount of contribution and consumption by using a centralized server to count every transaction [15, 3]. However, this approach is not possible in a fully-distributed system. More decentralized approach addresses to detecting cheating include [13, 9]. However, [13] assumes that peers called "brokers" will never be malicious, and [9] assumes that as long as a peer does contribute, the amount of its contribution does not matter.

Although the audit mechanism we propose here is based on ARA which is designed for pure p2p systems [4], as we noted in the introduction, there are differences between a file-sharing p2p system and a physical multi-agent system: these include mobility of agents, the need to define a payoff system, and locality of communication.

We further assume that physical agents cannot change their identity. Thus, unlike in p2p systems, we need not consider the effect of *white-washing*, where an agent may change its identity in order to clear the negative impact of the current identity [11], and may instead assume that the identity of an agent is permanent. Observe that the scale of a file-sharing p2p systems may be in the millions. However, we assume that the scale of our multi-agent system is $O(1000)$ to $O(10000)$. Given this smaller scale, each agent may be able to store some small amount of data for every agent.¹

In the previous work, we have experimented with physical robot agents and software simulations.² In the present work, we do not model individual agent behaviors and search strategies; instead, we model only the effects of our audit mechanism.

¹Note that such storage is not strictly necessary but its absence requires more complex referral mechanism, which we do not have space to describe in this paper.

²available at <http://taskhard.sourceforge.net> and <http://taskhardss.sourceforge.net>.

3 Definitions and Assumptions

In the rest of the paper, we use the terms robot and agent interchangeably. A mobile task t is served when the requisite number (req_t) of robot agents are dedicated to serving the task t and these agents simultaneously approach it. The serving agents are then free to proceed to serve other tasks. A *mission* is completed when every specified task is served.

The identity of each agent is permanent and cannot be forged, for example, because messages are signed by private keys and the corresponding public keys and agent ids are shared with all agents.

3.1 Agent Behaviors

Market-based mechanisms, such as auctions and swapping, described in [5] are used in order to allocate and re-allocate tasks to agents. In order to create a team of agents that serves a task t , an auction, resulting in req_t winners, is executed. The winners form a team and pursue t . If at least one pursuing robot agent successfully approaches the task t , t stops moving. While a robot agent is dedicated to a task (it won an auction for the task), the agent may swap its task with another agent if the service for the task is not yet complete. In order to swap tasks, both robot agents should agree to the swap (this happens only if the swap is beneficial for both) and the swap should be approved by the leaders of the two teams in order to avoid synchronization problems. When req_t robot agents successfully have approached t and stopped near t , the task is served and removed from the field. Then, the robot agents dedicated to t are free to serve other tasks.

We assume that the communication and sensing ranges of each robot agent are bounded. Thus, it is possible that the system may become partitioned into disjoint groups that cannot communicate with each other. We further assume that each robot agent knows the identities of other robot agents in its neighborhood (i.e., those within the communication range). As in previous work on multi-robot coordination [5], we assume that a robot agent sends its neighbors the sensor information that it has. However, robot agents do not relay sensor information that they have received from other agents. Audit-related information is also shared between robot agents that are within communication range.

An innocent robot agent (not cheating) is assumed to report every cheating attempt that it detects. Although reporting cheating attempts requires some resources, the resources required to detect and report cheating is usually significantly less than the resources that may be spent to serve a task using actuators, as well as the network, for cheating agents. The number of cheating agents is assumed to be small.

3.2 Synchronization

The system uses a clock value *period* for timestamping audit-related messages. Although large-scale decentralized physical agents may not have a globally shared clock, we assume that the agents are *loosely* synchronized. Specifically, the local clocks of the agents are within 1 ‘period’:

$$\forall r_1, r_2 \in \mathcal{R}, |t(r_1) - t(r_2)| \leq 1$$

where \mathcal{R} is the set of robot agents in the system and $t(r)$ is the period at robot r , which is an integer local clock value. The duration of one period is defined system-wide and is sufficiently large to cover, for example, rounds of an auction.

Each agent should keep *recent* (m periods) audit-related data so that other agents can query about recent data.

3.3 Interested Agents

Every agent that is or was in the communication range (*neighbors*) of a robot agent r in the recent past (m periods) is defined as an *interested agent* for r . We assume that the interest agent relation is symmetric. Interested agents collaborate to carry out audits (as we describe in § 4). Note that in a multi-robot system carrying out SR tasks, the number of neighbors in communication range is not so large; using all of them in an audit will maximize the probability of detecting cheaters using our method.

3.4 Payoff Function

A payoff function F_p is a mapping from the set of services S (such as process an auction, service a task, etc.) to the set of possible payoff values N :

$$F_p : S \rightarrow N \tag{1}$$

When an agent r_c receives a service s from another agent r_s , the client agent r_c pays $F_p(s)$ to the server agent r_s . Services s may include auctioneer service (auction dealer), swapping for a less-profitable task, taking a task from an agent, and providing fuel/energy to an agent. One exception is when an agent services a task. A servicing agent r_s gets a payoff by servicing a task although the ‘‘task’’ cannot provide the payoff to r_s because tasks are not modelled as agents in the system. If we assign F_p values for servicing tasks that are not originated by agents in the system, the payoff is to be provided by the system; e.g., the operator of the system. Note that even though tasks may be implemented as agents, because tasks in the SR problem do not participate (bid) in the market, we exclude tasks from our audit mechanism.

The value of F_p for a specific service s is constant, which implies that the F_p values are not market-based. Thus, it is

necessary to set F_p values manually before a mission (unless F_p is adaptive). However, we do not discuss how to set (or adapt) F_p values; instead we focus on how to audit the integrity of the reported payoff values.

For every transaction that generates a payoff, the two agents mutually send signed *transaction records*, which describe the transaction detail including the payoff value. Because the two agents know about the transaction details, any forged data in the records can be detected immediately and the receiver can use the records as evidence because they are signed by the sender.

3.5 Credit System

Credit is earned by providing services to other agents (*contribution*) and is spent by using other agents' services (*consumption*). The credit value of an agent represents its ability to consume the resources of other agents. The credit value (C_r) of an agent r at period t is calculated based on its recent service transactions: because transactions of the current period are counted in the next period, C_r is computed over the periods $t - 1, \dots, t - m$.

We define the amount of contribution and consumption for each service type. This is done by specifying a payoff function (F_p) which returns the corresponding contribution and consumption amount for the service.

A credit value of r , C_r , is calculated by the amount of contribution and consumption determined by F_p . Let $T_{I_r}(t)$ be the amount of consumption of r at period t , and $T_{O_r}(t)$ be the amount of contribution by r at t . Then, at period *now*,

$$T_{I_r}(t) = \sum_{i \in I(r,t)} F_p(i), \quad I(r,t): \text{services for } r \text{ at } t \quad (2)$$

$$T_{O_r}(t) = \sum_{i \in O(r,t)} F_p(i), \quad O(r,t): \text{services by } r \text{ at } t \quad (3)$$

$$C_r = \sum_{t=now-m}^{now-1} \{T_{O_r}(t) - T_{I_r}(t)\} \quad (4)$$

During a mission, a robot agent r may prioritize agents according to the credit values of the agents who request r 's services. Expected payoff value of a service may be included in the utility and cost functions that are used in bidding and coordination strategies. We do not discuss such strategies here as they are not germane to the audit scheme (the interested reader may refer to [5]). When a mission is completed, we may distribute a payment according to the credit value of each agent participating in the mission.

3.6 Types of Cheating

An agent or a team of agents may be able to commit cheating for its own benefits. Assume that credit values are given for running auctions, swapping for targets with less

utility and more costs, servicing targets, allowing a member of the servicing robot team to disengage from the shared target, taking the task of the disengaging member, and providing fuels to other agents. With the credits earned, a robot agent may spend the credits by buying fuel from other robot agents, swapping for easier tasks, and relinquishing a difficult task. A cheating agent can claim credits for services that it did not actually provide: e.g., receiving fuel with the falsely claimed credit.

We target the following types of cheating. Cheating is assumed to be beneficial to the robot agents that have committed cheating; otherwise, the actions are altruistic and we do not need to concern ourselves with such actions. We assume that agents cannot forge signatures associated with data because the data is signed by the private key of a sender.

We do not consider the creation of a *bogus task*, whereby a cheating agent reports serving a fake task. This assumption is reasonable because each task has a physical existence, and its identity can be easily verifiable; for example, each robot in [6] has a unique shape.

We consider the following types of cheating:

Exaggerated Credit: The cheating agent sends forged information in order to justify an exaggerated credit value and avoid detection.

Conspiracy: There may be a team of agents that collaborate together in order to commit cheating; e.g., one makes an exaggerated credit and the others provide corroboration to validate the claimed credit. However, if a collaborator gives away credit it is due to another, this is regarded as a credit transfer rather than a conspiracy.

Blame Transfer: A cheating robot agent may blame an innocent robot agent in order to hide its own cheating attempts. Blame transfer also includes creating a false service report: e.g., a cheating agent claim to have provided a service, which has not happened, to another agent.

Omitting Interested Agents: A cheating robot agent may announce fictitious lists of interested agents. A cheating agent that omits some interested agents in order to hide inconsistencies created by sending fictitious information.

Stuffing the List of Interested Agents: A cheating agent may also add agents that have never been neighbors to the list.

4 Methods

Robot agents retain information about interested agents. This information can be queried by other agents. The retained information consists of every action that changes a credit value, which generates transaction records. Robot agents also keep the history of queries and the results so

that the agents can verify the queried agents later and share the list of interested agents.

If a mission has a finite and ‘not-so-long’ length or duration, we can set m to infinity so that robot agents keep the entire history. On the other hand, if a mission may have indefinite length or duration, m should be finite as the retained information may not be bounded. However, as discussed in [4], if the system duration is indefinite, a volatile credit system which forfeits old information can allow the system to evolve; in this case, an agent has no advantage in hoarding credit.

4.1 Managing Lists of Interested Agents

An agent updates its interested agents list in every period and propagates the updated list to its current neighbors. Because robots are mobile, it is possible that some interested agents have now gone out of communication range. In such cases, the updated list is propagated only to the current neighbors.

A robot agent stores the lists of interested agents that it receives from its own interested agents. Because the interested agents list of the neighbor can be partial (recall that agents are mobile and may go out of communication range), an agent keeps each update with its period value. A robot agent also sends queries to its new neighbors, i.e. agents which have approached it recently. Information from the new neighbors is used to empty slots of its lists whenever possible.

When audit-related information is propagated to interested agents, the message is signed with the sender’s private key so that the authenticity of the message can be verified with public keys, which are stored in every agent. The message including the signature is retained; this enables the message signature to be used for verification purposes whenever an inconsistency is detected. The list also includes when and where the interested agents are found (i.e., a *contact report*) so that others may confirm if an interested agent was indeed nearby during the period claimed.

Table 1 shows the three types of data each agent r may have. Let $tr(r \rightarrow x, t)$ be the transaction record of a service by r for x at t . Data propagated by an agent r_x (shown in the second row) is signed and propagated by r_x . Agent r_x propagates audit-related information to its neighbors (interested agents) including r . The third row shows data that can be queried. Agent r_t is an audit target, which is a neighbor of both r_x (audit sample) and r . This data is signed by r_t and propagated to r_x . This can be queried to r_x by r .

4.2 Audit Mechanism

A robot agent audits the integrity of the targeted agent r_t ’s credit value by comparing the transaction records of r_t

Table 1. Audit-related data in a robot agent r

local	a list of agents and their public keys $C_r, tr(r \rightarrow r_x, t), tr(r_x \rightarrow r, t), IA_r$
propagated	$\forall r_x, \{C_{r_x}, tr(r_x \rightarrow r_t, t), tr(r_t \rightarrow r_x, t), IA_{r_x}\}$
queried	$C_{r_t}, tr(r_x \rightarrow r_t, t), tr(r_t \rightarrow r_x, t), IA_{r_t}$

with those from the interested agents of r_t . In order to do so, each agent r propagates the transaction records ($I(r, t)$ and $O(r, t)$ in Eq. (2), (3)) as well as C_r to its interested agents and keeps the transaction records sent from its interested agents. A robot agent r may query to another agent r_a about the transaction records of r_a ’s interested agents, which allows r to check the integrity of r_a ’s information.

The mechanisms rely on the report from third party agents. The inter-agent audit mechanism is stochastic; i.e., an integrity check is executed for randomly chosen agents with randomly chosen samples.

A robot agent r verifies the consistency between the local records. A set $IA_{r'}$ consists of r' ’s interested agents that is stored locally at r . A transaction record element $i(r', t)@r_p$ or $o(r', t)@r_p$ is in the transaction records propagated by r_p : $i(r', t)@r_p \in I(r', t)$ from r_p and $o(r', t)@r_p \in O(r', t)$ from r_p . When an inconsistency is found locally, it is obvious whose fault caused the inconsistency.

The following properties hold for interested agents:

$$\forall r_x \in IA_r, r \in IA_{r_x} \quad (5)$$

$$\forall r_x, r_y \in IA_r, r_x \in IA_{r_y} \leftrightarrow r_y \in IA_{r_x} \quad (6)$$

$$\forall r_x, (i(r_x, t')@r \vee o(r_x, t')@r) \rightarrow r_x \in IA_r \quad (7)$$

$$\forall r_x \in IA_r, r_i, (i(r_i, t')@r_x \vee o(r_i, t')@r_x) \rightarrow r_{it} \in IA_{r_x} \quad (8)$$

Transaction Records:

$$I(r) \text{ aware of } \forall i(r, t')@r_x \text{ and } \forall o(r, t')@r_x \quad (9)$$

$$\forall r_x, r_y, t' \exists t'' i(r_x, t')@r_y \leftrightarrow o(r_y, t'')@r_x \wedge |t' - t''| \leq 1 \quad (10)$$

$$C_{r_t} = \sum_{i(r_x, t')@r_t} F_p(i(r_x, t')@r_t) - \sum_{o(r_x, t')@r_t} F_p(o(r_x, t')@r_t) \quad (11)$$

Eq. (5) and (6) denote that being interested is symmetric; i.e., a is interested in b if and only if b is interested in a . Eq. (7) shows how to construct its own interested agents list. Eq. (8) shows how to construct interested agents lists of a robot’s interested agent. Eq. (9) and (10) ensure that the transactions of r are also recognized by the counterparts of the transactions. Eq. (11) shows how to verify the credit value based on the transaction records.

4.2.1 Credit Audit

The basic audit method, called a *Credit Audit*, involves querying credit values of a targeted agent r_t from sample agents (chosen with probability p_C) of the r_t 's interested agents. Each C_{r_t} value reported by r_t is compared with the C_{r_t} values that were queried to the sample agents. An agent r executes a Credit Audit for every interested agent of r .

To evade detection by a Credit Audit, a cheating agent r_t should exaggerate C_{r_t} evenly for every interested agent. To evade local integrity check Eq. (9)-(11) as well, r_t should modify transaction records $\{I(r_t, t'), O(r_t, t')\}$ for every interested agents of r_t and the transaction records propagated to an interested agent of r_t , r_{i1} , should not be modified if the records are being sent to r_{i1} . Thus, if a transaction record related with r_{i1} is modified and propagated to r_{i2} , the transaction records of r_t propagated to r_{i1} will be inconsistent with those propagated to r_{i2} . We verify the consistency of inter-agent transaction records using an *Transaction Record Audit*, as described below.

4.2.2 Transaction Record Audit

A *Transaction Record Audit* compares the transaction records of a targeted agent r_t with the transaction records received in response to queries from the sampled interested agents of r_t . Because an exhaustive search on the transaction records may incur too much communication overhead, this audit mechanism also picks samples of the records. During each period, an auditing agent r executes this method for the randomly selected targeted agent r_t , which is one of the interested agents of r , with a fixed probability of p_T . Then, r queries sample agents r_s drawn from interested agents of r_t (with a probability of p_{TS}). The queries return the transaction records of services between r and r_t and between r_s and r_t . In order to ensure that C_{r_t} is not modified differently only for r to avoid the Transaction Record Audit, the values of C_{r_t} are also queried from the sample agents r_s .

4.2.3 Interested Agents List Audit

An *Interested Agents List Audit* verifies the list of interested agents of a targeted agent r_t . The list is audited because r_t may omit an agent r_o from the list that is sent to agents other than r_o . Such omission may be done to hide transaction records between r_t and r_o ; if r_t does only one of omitting r_o or hiding a transaction record with r_o , the omission may be detected locally by r_t 's interested agents (inconsistency between the list of interested agents and transaction records) or by the Transaction Record Audit.

An agent r executes Interested Agents List Audit for r_t with a probability of p_I per period. Then, r picks samples with a probability of p_{IS} from r_t 's reported list and queries

Table 2. The probability constants

p_C	<i>Credit Audit</i> is run p_C per interested agent
p_T p_{TS}	<i>Transaction Record Audit</i> is run p_T Sampling rate from the interested agents of the target agent
p_I p_{IS}	<i>Interested Agents List Audit</i> is run p_I Sampling rate from the interested agents of the target agent and the investigating agent's neighbors

if r and the sample is in the list interested agents list of r_t in the sample. At the same time, r also picks sample agents in the communication range that are not in the interested agents list of r_t with the same probability of p_{IS} . An agent r queries these sample agents whether each sample agent is interested in r_t or not. If one of the samples is interested in r_t , there is inconsistency between the interested agents list of r_t for r and for the sample or the sample agent is lying. The second method prevents evading Interested Agents List Audit by partitioning the set of interested agents in order to prevent the agents from different partitions verifying the integrity of the interested agents of r_t .

When an inconsistency is found, whether to blame the sample or r_t is determined by reviewing the interested agents list of r_t signed by r_t that were sent to the sample.

4.3 How Cheating is Detected

We describe how the types of cheating (Section 3.6) are detected. We assume that a cheating agent tries to minimize the possibility of detection.

The probability of detection discussed in the following paragraphs represents the probability of detection by each interested agent. Thus, the larger the list of interested agents for an agent, the higher the detection probability becomes. This implies that an agent that interacts with more agents has a greater chance to be audited. Because an agent with more interactions may have a greater effect on the system, this in turn means that we offer more accurate audits for more important agents. Even if we provide payoffs after a mission completion (which means that we have a change to audit centrally and completely after the mission is completed), we can reduce the effect of cheating attempts in the course of a mission.

4.3.1 Exaggerated Credit

Let an agent r_t be a cheating agent that exaggerates C_{r_t} by modifying transaction records. Then, the cheating attempt is detected by Credit Audit and Transaction Record Audit. If r_t does not modify transaction records, it is always detected by r_t 's interested agents locally with Eq. (11).

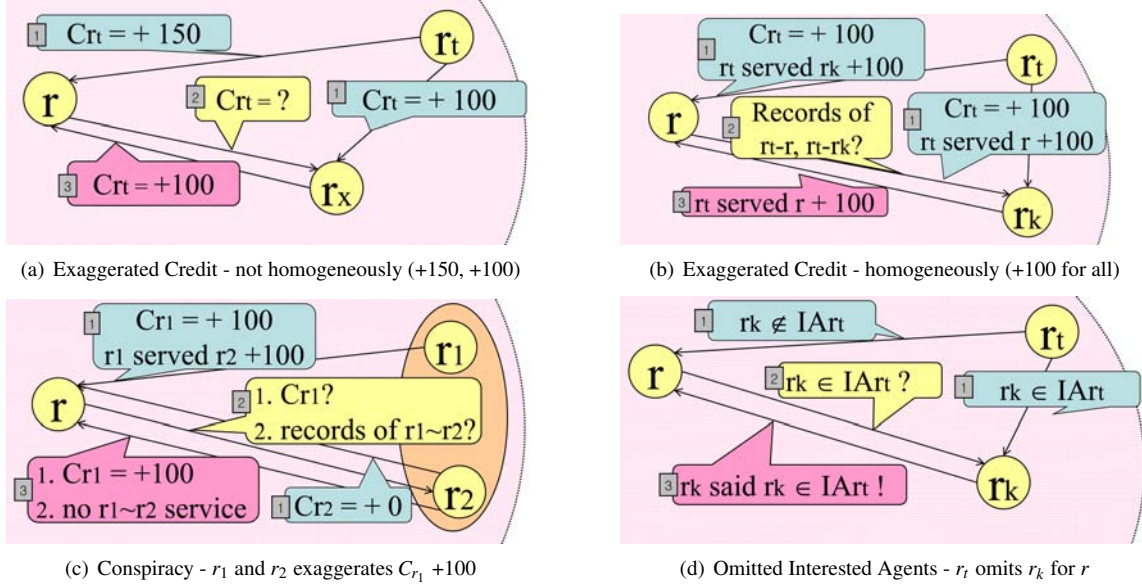


Figure 1. How cheating is detected

If r_t does not exaggerate C_{r_t} homogeneously, the exaggeration is detected by the Credit Audit with some probability: Fig. 1(a). If a transaction record related with r_k is modified and propagated by r_t to exaggerate C_{r_t} homogeneously and avoid being detected locally, it is detected by Transaction Record Audit: Fig. 1(b). If the transaction records are modified homogeneously, r_k always detects the cheating attempts by Eq. (9) and (10). Thus, r_t needs to modify differently for different interested agents to evade the audit mechanism which, because of the inter-agent inconsistency, can in turn be detected by the Transaction Record Audit.

Let us assume that a transaction record of r_t related with r_k is modified and propagated to the interested agent of r_t other than r_k and another transaction record is modified for r_k . Then, the detection probability per period and per interested agent is p_T for r_k and $p_T p_{TS}$ for other agents.

The larger the number of agents (r_k) that are affected by the modified transaction records, the higher the detection probability is; thus, r_t will minimize the number of r_k [4]. This principle—minimize the affected agents and records in order to minimize the probability of detection—applies to other types of cheating as well.

4.3.2 Conspiracy

Conspiracy is mitigated by a Transaction Record Audit. Let r_1 and r_2 be the two collaborators; i.e., r_1 ignores r_2 's cheating attempts and r_2 ignores r_1 's. Assume that r_1 exaggerates C_{r_1} and modifies transaction records related with r_2 . Then, r_2 will support r_1 when another agent queries of r_1 's records. However, when another agent queries of r_2 's

records in order to verify r_2 's credit value, r_2 needs to report honestly if r_2 wants the credit r_2 deserves. Otherwise, r_2 should sacrifice C_{r_2} , which, in turn, means that the conspiracy becomes useless; i.e., C_{r_1} earned by the conspiracy is the same as C_{r_2} wasted by the conspiracy.

In order to make the conspiracy useful for the team of r_1 and r_2 , r_2 should report transaction records differently to different agents, which creates inter-agent inconsistency, which is detected by Transaction Record Audit or Credit Audit executed by a common interested agent of r_1 and r_2 ; the case becomes same as Exaggerated Credit. Thus, the cheating team will make the two sets of interested agents (r_1 's and r_2 's) disjoint.

Such cheating attempts are detected when an auditing agent r interested in r_1 executes Credit Audit on r_1 and Transaction Record Audit on r_1 sampling r_2 (Fig. 1(c)): with probability $p_T \cdot p_{TS}$. Auditing r_2 and sampling r_1 does the same. Therefore, the detection probability is $2p_T \cdot p_{TS} - (p_T \cdot p_{TS})^2$ per period and per agent.

Note that the size of the team needs to be minimized to evade detection because the larger the team, the higher the probability of being sampled. Besides, unlike p2p systems, where a peer can reject service requests or choose servicing peers in order to control its interested peers, agents cannot choose their interested agents easily because interested agents are determined by the physical locations.

4.3.3 Blame Transfer

A report of a cheating attempt requires a proof: a pair of inconsistent records signed by the accused agent. Because

an agent cannot forge a signature, Blame Transfer is easily detected by checking the signature. Blame Transfer only makes it easier to detect cheating agents. Note that a bogus or false service report also requires a signature for the service transactions.

4.3.4 Omitting Interested Agents

The last type of cheating is mitigated by Interested Agent Audit. Assume that a cheating agent r_i , which has an auditing agent r as one of its interested agents, omits r_k from its interested agents list that is propagated to r . It is detected if r executes Interested Agent Audit for r_i sampling an agent that has received r_i 's interested agents list including r_k (Fig. 1(d)); the probability is $p_I \cdot p_{IS}$ per period and per agent for agents other than r_k and p_I for r_k .

In order to minimize the probability of detection, r_i should propagate the interested agents list homogeneously omitted and minimize the probability that r_k is in the interested agent lists propagated to the interested agents other than r_k . Besides, if r_k has a service transaction with r_i , it can be detected either by Transaction Record Audit (if r_i modifies transaction records) or by Eq. (8) locally (if r_i does not modify transaction records). To minimize the possibility of detection, r_i needs to omit only one agent; otherwise r_i increases the number of possible auditing targets and samples.

4.3.5 Stuffing the List of Interested Agents

Adding interested agents only increases the overhead and cannot decrease detection probability. Such cheating attempts fail to reduce the detection probability because it only increases the number of audits; the chance to pick a specific sample does not change.

However, if a bogus interested agent r_k of a cheating agent r_i is away from r_i and r_i 's interested agents so that r_i 's interested agents cannot communicate with r_k , r_i may create a bogus transaction record related with r_k . Such cheating attempts are detected if one of r_i 's interested agents approaches to r_k before m expires. Besides, we can detect bogus agents based on the contact report (Section 4.1).

4.4 Mobility of Robot Agents

Except in Section 4.3.5, we have not discussed the effect of agent mobility on the detection probability. However, since the communication range is bounded, agent mobility creates disconnections and reconnections between agents. Thus, the probability that a mobile agent which was once in the communication range travels out of the range and the probability that such agent approaches into the range again have a significant effect.

In physical experiments ([6]) and software simulations ([5]), we observe that agents usually form a number of

groups and such groups are maintained over some time. This observation implies that p_{out} is much smaller than the cases when the agents are assumed to move randomly.

When an agent r_k moves out of the range from r , r cannot sample r_k for an audit. If such audit trial is stored for later execution and is executed when the audit becomes possible, we can increase the detection probability without increasing communication overhead; i.e., such audit messages would have been sent already if a run-away event did not occur. Thus, such delayed audit will decrease the effect of mobility on the detection probability.

Assume that a cheating agent r_i is physically located between r_A and r_B and (r_i, r_A) and (r_i, r_B) are located within the communication range while (r_A, r_B) are not within the range. In such case, r_i may avoid detection by partitioning transaction records or interested agents list. However, the cheating attempt can be detected if one or more of the following conditions are met:

- another agent r_C is a neighbor of both r_A and r_B .
- the distance between r_A and r_B becomes closer.
- another innocent agent approaches r_A and r_B .

We do not further discuss the mobility issue. However, we assume that the negative effect on the detection probability is not so significant. Because the system is large-scale, it is not likely that an agent has a small number of interested agents; thus, the effect of mobility can be easily mitigated.

4.5 Blacklist Maintenance and Penalization of Cheating

When an inconsistency is discovered, the auditing agent has the proofs; e.g., the two contradicting transaction records, lists of interested agents, or credit values signed by the same agent, a proof of a service and its contradiction transaction record, a misbehavior report without proper signs, and others. A misbehavior report includes such proofs and is sent to every neighbor and propagated to every agent. A blacklist is kept by each agent.

5 Performance Analysis

We study the performance (detection probability and communication overhead) analytically by building a model in Maple on Windows XP. We use the following parameters in the model. The duration of a period is 1 minute and m is 30. The average number of interested agents is 30. The communication bandwidth is 11Mbps assuming 802.11b networks, which was used in [6]. The size of a signature is 20 bytes and that for a data element is 4 bytes. We assume that the probability of sampling failure due to

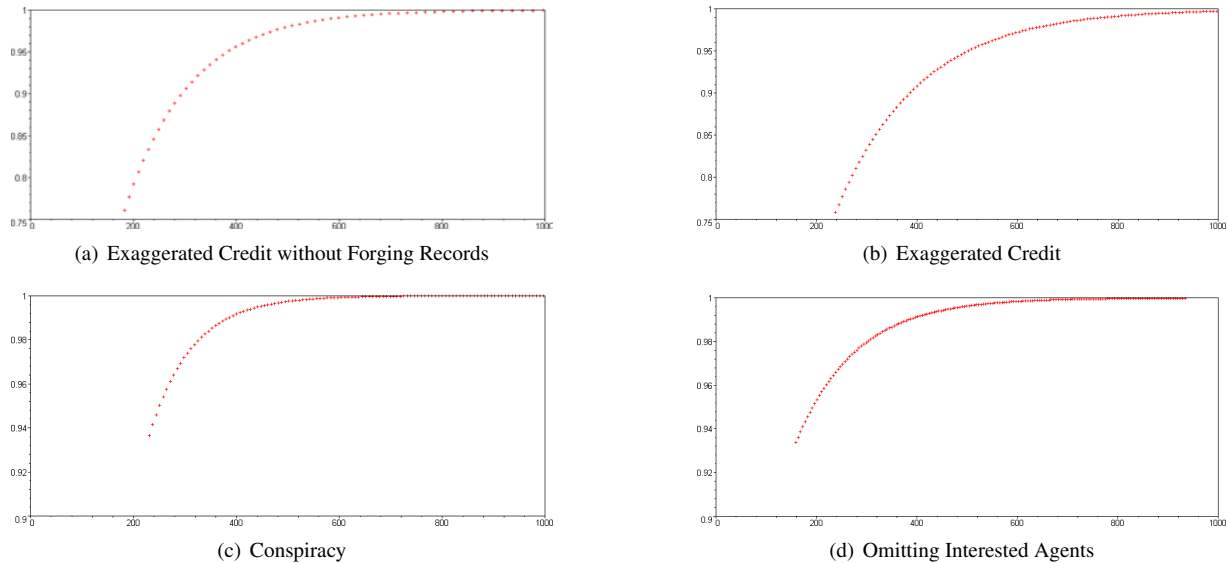


Figure 2. Detection probability and communication overhead. X-axis shows the communication overhead in bytes per agent and per period. Y-axis shows the probability of eventual detection.

the mobility is 0.1, which is the probability that an interested agent is out of communication range during a period after the agent was in the range. The average number of service transactions between two agents in the communication range during a period is assumed to be 0.25.

We assume that cheating agents are smart enough to avoid being locally detected with the checks given in Eq. (5)...(11), which always detect corresponding cheating attempts without communication overhead. We experiment with the cheating described in Section 4.3. The number of cheating agents is assumed to be small; i.e., we assume that in a given communication range, only one agent or a pair of agents (of one team) are cheating.

5.1 Detection Rate

Fig. 5 shows the relation between the detection probability and the communication overhead for the types of cheating that are mentioned in Section 3.6. Fig. 2(a) is for Exaggerated Credit without forging transaction records. This type of cheating is not discussed previously, however, we consider this case in order to see the detection probability of not-so-smart cheating agents and to find a proper p_C value for given conditions.

Fig. 2 plots communication overhead versus detection probability by varying constants of Table 2 (from 0.034 to 0.200) by 0.001. Communication overhead for the audit method used to counter different types of cheating is plotted: Credit Audit for Fig. 2(a), Transaction Record Audit for Fig. 2(b) and 2(c), and Interested Agents List Audit for

Fig. 2(d). Blame Transfer is not considered because it is always detected.

An example setting for the constants (defined in Table 2) that provides 99% or greater detection probability for every type of cheating is $\{p_C = 0.076, p_T = 0.05, p_{TS} = 0.12, p_I = 0.05, p_{IS} = 0.083\}$. This example has the overhead of 1846 bytes and 22.7 transmissions per agent and per period for both directions of communications.

Another example setting, which has 95% or larger detection probability is $\{p_C = 0.061, p_T = 0.05, p_{TS} = 0.074, p_I = 0.05, p_{IS} = 0.041\}$. This example has the overhead of 1076 bytes and 14.4 transmissions per agent and per period for both incoming and outgoing communications.

Because the mobile robot agents use wireless communication, the communication medium will be exclusively occupied by a single transmitting agent unless multi-channel communication is used. Therefore, in the case of 99% detection probability, the communication overhead on a channel is 27690 bytes and 340.5 transmissions per period. For the case of 95% detection probability, it is 16140 bytes and 216.0 transmissions per period. With 802.11b networks, this is ignorable; i.e., the example with 99% detection requires 461.5B/s and 5.7 packets/s.

5.2 Analysis of Overheads

Beside communication for integrity checking, the system requires the communication to propagate the transaction records and interested agents lists. This process requires more bytes of transmission than the integrity check meth-

ods. For the given conditions, the average number of bytes transmitted for transaction records and interested agents list per agent and per period is 700 bytes per interested agent, which incurs 10.3 kB/s of traffic in the wireless channel assuming that an agent uses unicast only. This overhead is significantly larger than the overhead of integrity check. However, this overhead is still affordable for 802.11b networks (about 0.1% of the bandwidth). If an agent uses broadcast for agents in communication range, it is reduced to 343 bytes/s.

The overall communication overhead of the audit mechanism is approximately 10.7kB/sec for a shared medium, which is ignorable for a 802.11b network. The number of communications is $O(n^2)$, where n is the number of interested agents. The number of communications becomes constant when the number of agents in the communication range has a constant bound.

The audit mechanism also incurs computation and memory overhead. We have not analyzed the computation overhead, however, the most significant overhead will be incurred by signing messages and by verifying the signatures of incoming messages. In the given conditions, each robot agent is expected to decrypt or encrypt 1.38 times per second. For a typical mobile computing unit, this should not be a problem. The computation complexity is $O(n^2 + nm)$, where m is the number of periods that are audited and n is the number of interested agents, which becomes a constant when the communication range is constantly bounded, assuming that the density of agents is bounded.

Each robot agent should store every transaction record and list of interested agents for each of interested agent. The stored information should include the whole messages including the signature in order to be used against Blame Transfer and to be used for blacklists. The memory complexity is $O(n^2m)$. Under the scenarios we studied, the size of stored information for auditing is approximately 630kB per agent. Assuming that there are 1000 agents in the system, in order to keep the list of every agent and its public key, about 1MB additionally memory will be required. This memory overhead is affordable for small nodes such as iPAQ h5550 and h4100, which have 128MB or 64MB of RAM and are used to run robot agents in [6].

6 Conclusions and Future Work

We have proposed a credit system with a payoff function and an audit mechanism to address cheating of self-interested agents in a large-scale physical multi-agent system. The credit system and the payoff function calculates the service contribution and consumption of each agent. The audit mechanism verifies the integrity of the credit value of each agent. Our approach suggests that an honor system with limited integrity checking may work in the

presence of a limited percentage of cheating agents. We need to define the payoff function F_p for specific services. The payoff function may be adaptive and use the market system to decide payoff values for each service.

For future research, we would like to further analyze the effect of agent mobility. Because agents tend to form loose groups in problems such as search and rescue, mobility does not result in a random walk; we may need to analyze the grouping characteristics in our simulations. Such analysis may be done by extending our simulator. The effects of the audit mechanisms may be studied by implementing it in multi-robot simulations. We would also like to let the mechanism be fault tolerant.

References

- [1] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.
- [2] R. Aylett and D. Barnes. A multi-robot architecture for planetary rovers. In *Proc. of ASTRA*, 1998.
- [3] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives for sharing in peer-to-peer networks. In *Proc' of EC*, pages 264–267, 2001.
- [4] M. Ham and G. Agha. Ara: A robust audit to prevent free-riding in p2p networks. In *Proc' of P2P*, pages 125–132, 2005.
- [5] M. Ham and G. Agha. Market-based coordination strategies for large-scale multi-agent systems. *CoSIWN*, 2(1):126–131, 2007.
- [6] M. Ham and G. Agha. Market-based coordination strategies for physical multi-agent systems. *SIGBED Rev.*, 5(1), 2008.
- [7] A. Healey. Application of formation control for multi-vehicle robotic minesweeping. In *Proc., of CDC*.
- [8] J. Jennings, G. Whelan, and W. Evans. Cooperative search and rescue with a team of mobile robots. In *Proc. of ICAR*, pages 193–200, 1997.
- [9] S. Kamvar, M. Schlosser, and H. Garcia-Molina. Incentives for combatting freeriding on p2p networks. In *Proc' of EuroPar*, volume 2790, pages 1273–1279, 2003.
- [10] R. K. Karmani, T. Latvala, and G. Agha. On scaling multi-agent task reallocation using market-based approach. In *Proc. of SASO*, pages 173–182, 2007.
- [11] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In *Proc' of P2PECON*, 2003.
- [12] S. Marti and H. Garcia-Molina. Limited reputation sharing in p2p systems. In *Proc' of EC*, pages 91–101, 2004.
- [13] A. Mondal, S. K. Madria, and M. Kitsuregawa. Abide: A bid-based economic incentive model for enticing non-cooperative peers in mobile-p2p networks. In *Proc' of DAS-FAA*, pages 703–714, 2007.
- [14] L. Parker and B. Emmons. Cooperative multi-robot observation of multiple moving targets. In *Proc. of ICRA*, volume 3, pages 2082–2089, 1997.
- [15] N. Szabo. Micropayment and mental transaction costs. *The 2nd Berlin Internet Economics Workshop*, May 1999.