Yale University
Department of Computer Science

A Simple Game for the Study of Trust in Distributed Systems

Zoë Diamadi   Michael J. Fischer

YALEU/DCS/TR-1207
January 4, 2001

A Simple Game for the Study of Trust in Distributed Systems

Zoë Diamadi
Department of Computer Science
Yale University
New Haven, CT 06520–8285
diamadi-zoe@cs.yale.edu

Michael J. Fischer
Department of Computer Science
Yale University
New Haven, CT 06520–8285
fischer-michael@cs.yale.edu

ABSTRACT

Trust is an important aspect of the design and analysis of secure distributed systems. It is often used informally to designate those portions of a system that must function correctly in order to achieve the desired outcome. But it is a notoriously difficult notion to formalize. What are the properties of trust? How is it learned, propagated, and utilized successfully? How can it be modeled? How can a trust model be used to derive protocols that are efficient and reliable when employed in today's expansive networks? Past work has been concerned with only a few of these issues, without concentrating on the need for a comprehensive approach to trust modeling.

In this paper, we take a first step in that direction by studying an artificial community of agents that uses a notion of trust to succeed in a game against nature. The model is simple enough to analyze and simulate, but also rich enough to exhibit phenomena of real-life interactive communities. The model requires agents to make decisions. To do well, the agents are informed by knowledge gained from their own past experience as well as from the experience of other agents. Communication among agents allows knowledge to propagate faster through the network, which in turn can allow for a more successful community. We analyze the model from both a theoretical and an experimental point of view.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; I.2.6 [**Artificial Intelligence**]: Learning—*knowledge acquisition, parameter learning*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*intelligent agents, multiagent systems*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

To appear in the Proceedings of the International Symposium on Software Engineering, March 2001, Wuhan, China

General Terms

Algorithms, Experimentation, Security

Keywords

Agent algorithm, artificial society, trust

1. INTRODUCTION

Uncertainty about another's likely behavior is a significant obstacle in establishing productive cooperative relationships. In everyday life, we reduce risk by relying on knowledge of our own past experiences and on the collective experiences of the community. We *trust* another (for a particular activity) when we have knowledge that reduces our perceived risk to an acceptable level (for that activity). Trust is intimately tied to decision-making, for having trust means that we may engage in an activity with another, whereas lack of trust will lead to avoidance of the activity. Trust can also be viewed as the distillation of a potentially large quantity of information that is sufficient to allow a trust relationship to be established. A successful distillation requires the ability to update the trust data to incorporate new experience.

In recent years, increasingly varied activities are performed on-line, over great distances. Formerly geographically bound communities are rapidly fused into an expansive collection of world-wide virtual ones. Familiar informal methods for establishing trust are overwhelmed by the vastness and complexity of open networks. We need automated methods for trust establishment that use the networks themselves! This thrusts trust into the forefront of secure cooperative distributed systems design.

The following example illustrates the point: Suppose you are registering at a secure web site. Your browser displays a message stating that Trust'em-all Inc. has signed a certificate for the site and asks whether you accept the certificate or not. Your browser provides no helpful information other than prosaic details (name of the certificate signer and that of the holder, expiration date, serial number, etc.) to guide you through the decision process. Depending on your temperament and the desirability of the service provided through the web site, your response could range between proceeding without any further consideration and absolutely refusing to accept the certificate. But what principles should guide you in making this decision? What information would you need in order to decide on the trustworthiness of the web site creator/maintainer? What about the trustworthiness of Trust'em-all Inc.? Should you trust it more or less than Trust'em-not Inc., and why? Would you feel more confident in your decision if you had certificates from both certification authorities? What would you do if the certificates were inconsistent? Reasoning about a decision process seems a formidable task. A similar set of trust concerns underlies not only all certificate-based activities (e.g., using Java applets or Javascript code embedded in a webpage's HTML source, loading an encrypted webpage, sending secure e-mail to a PGP user, etc.), but also a number of services from different areas of secure distributed computing such as key management and network routing.

In order to answer any of these questions effectively, we must understand how information such as beliefs, opinions, and impressions can be transformed into knowledge, and how that knowledge can be used to guide successful decision-making about future actions. In this paper, we present the results of our initial efforts towards this goal.

We model an interactive community of agents who choose among available actions and receive rewards. Some actions are more valuable than others. The success of an agent depends on its ability to select the more valuable action at each stage. The success of the community is measured by the collective success of its agents. We focus attention on agents that base their actions on an explicit trust model.

Section 2 of the paper presents an overview of related work on trust. Section 3 describes our game environment in detail. Section 4 presents the *learn_trust* agent algorithm which is the main focus of this paper, along with other algorithms to provide a basis for comparison. Section 5 analyzes *learn_trust* from a theoretical and an experimental point of view and summarizes our observations. Section 6 outlines possible ways to modify *learn_trust* to improve its asymptotic behavior.

2. RELATED WORK

Most related work has resulted from efforts to use trusted authorities in solving the authentication problem for public-key cryptography. In such a setting, users can communicate securely only if they can retrieve each other's public-key. The traditional solution to establishing the binding between the intended recipient of a message and his/her public key is for the sender to find a sequence of authorities such that each authority in the sequence can authenticate the next, while the last one can authenticate the recipient, and the first one can be authenticated by the sender. Such a construction was first presented in [2] and was later used in [7, 9, 15, 16] etc. Since the success of this approach depends on the correctness of all authorities in the sequence, the use of multiple such sequences was proposed, along with metrics that measure the confidence in the derived binding given the collection of found sequences (e.g., [1, 10, 12, 14, 16][1]).

However, trusted authorities are not necessarily trustworthy, and some metrics consider this possibility ([1, 10, 16]). They operate in the context of a directed graph, whose configuration and semantics vary from metric to metric. For example, in [1] and [10], the graph nodes represent entities that possess and use private/public key pairs, while in [16], they represent public keys. In [1], an edge from the node of entity A to that of B can represent a relationship between the two (e.g., A trusts B to authenticate other entities), while in [10], it can represent a 3-place relationship between the user evaluating the metric, A, and B (e.g., the user holds a certificate for B's public key issued and signed by A). Depending on the metric, the nodes or edges of the graph are labelled with numbers signifying degrees of trust, assigned by either the entities or the user evaluating the metric. Each metric then allows specific calculations to be performed over the trust assignments in order to measure the confidence in the binding between an entity and its suggested public key. Nevertheless, no procedures are supplied for the entities or the user to arrive at those initial trust assignments, and no trust semantics are specified. For example, in [10], the relationship between reality and the trust values used in the model is unspecified. The lack of algorithms for learning trust, along with ambiguities in the interpretation of the trust relationships, makes it impossible to assign clear semantics to the result provided by the metric. These works seem to be concerned primarily with the issue of how trust is propagated. They ignore other aspects of the trust modeling problem such as providing unambiguous semantics and efficient algorithms for learning and using trust.[2]

We should note here what our work is not concerned with. Our efforts are not in the direction of providing a trust logic or a solution to the trust management problem introduced in [4]. Trust logics (e.g., [5, 8, 11, 15]) are tools that draw conclusions from given initial trust relationships, while trust management systems, (e.g., *KeyNote* [3], *PolicyMaker* [4], *REFEREE* [6]), are tools that allow the user to express security policies, credentials and trust relationships, in a flexible and unified fashion. Neither trust logics nor trust management systems are concerned with the questions of where trust comes from, how it is learned, or how it should be used to guide action successfully, all of which are vital issues underlying the design and analysis of a trust policy.

3. THE GAME

Our game is one of imperfect information and chance. A group of n, fully competent, honest agents participate. Their goal is to maximize the expected earnings of the group. Each agent is characterized by two features: a unique identifier $i \in G = \{1, 2, \ldots, n\}$, and a number $r_i \in (0, 1)$, which is the agent's degree of reliability. The reliability values are chosen independently at random according to a fixed probability distribution R when the agents are created. The unique identifier is public information, as is the distribution R. However, the degree of reliability for each individual agent is private.[3] It affects the agent's behavior in a specific fashion described below. Values are assigned to all agents'

[1] [12] and [16] were not proposed as metrics. They were first interpreted as such in [13].

[2] Specifically, the only metric we know how to evaluate efficiently is that proposed in [16]. The metrics proposed in [1] and [10] are exponential in the size of the graph in the worst case. For an extensive comparison of [1], [10], [12] and [16], the reader can consult [13].

[3] The degree of reliability of an agent is not known to itself or to any other agent.

features before the game starts, and those values remain constant throughout the execution of the game.

The game develops in stages of interaction. During each stage, an ordered pair of agents $(i, k) \in G \times G$, $i \neq k$, is chosen and invited to participate in an *encounter*. Agent i is called the *active* agent, and agent k is the *passive* agent. The passive agent always accepts the invitation, while the active agent decides whether or not to do so by use of a decision rule which may depend on the passive agent's public features and on the contents of the active agent's private memory. If the active agent accepts the invitation, then an encounter takes place.

An agent's degree of reliability determines how likely it is for the agent to behave "well" when it participates in an encounter as the passive agent. Specifically, for each accepted invitation, a coin is tossed that is biased according to the passive agent's degree of reliability. Depending on the result of the toss, the *character* χ of the encounter is "good" or "bad".

Only the active agent receives a payoff, which depends on χ. If $\chi = \mathsf{good}$, the profit is P_g. If $\chi = \mathsf{bad}$, the profit is P_b (which will generally be zero or negative). We assume that $P_g > P_b$. All payoff values are fixed before the game starts and remain so throughout its execution. They are also known to all agents. It follows that the expected payoff for an encounter with agent k of reliability r_k is:

$$E[\text{payoff}_k] = r_k P_g + (1 - r_k) P_b. \tag{1}$$

Agent k is called *good* if $E[\text{payoff}_k] > 0$, *bad* if $E[\text{payoff}_k] < 0$, and *neutral* if $E[\text{payoff}_k] = 0$. On average, the society makes profits from encounters with good agents, and it suffers losses from encounters with bad agents.

As the game progresses, each agent tries to gauge the other agents' assigned degrees of reliability so that it can be more successful in its future interactions. The only way an agent receives new information is by participating in encounters as the active agent. If $\chi = \mathsf{good}$, the active agent is given the contents of the passive agent's memory. We call this *indirect* information. In any case, the active agent learns χ. We call this *direct* information.

An agent i who declines an invitation to participate with agent k is said to *distrust* agent k. Such an agent gets no new information about k's reliability from the declined opportunity. Of course, i continues to revise its opinion about k as a result of indirect information from subsequent encounters with third parties, which could later lead it to reverse its decision to distrust k and decide to participate with k again.

For the experimental results presented in this paper, we fix the payoffs and the probability distributions. We take R to be unif$(0, 1)$, the uniform distribution from the real open interval $(0, 1)$. The pairs of agents (i, k) invited to participate in encounters are chosen uniformly at random from $G \times G$ (subject to the constraint that $i \neq k$). We fix $P_g = 2$ and $P_b = -2$.

4. AGENT ALGORITHMS

We now define several kinds of agents for playing our game. These agents differ in the amount of memory they have, in their algorithm for deciding which invitations to accept, and in their method for incorporating new information into their memories. A goal of this study is to find simple algorithms that require relatively small amounts of memory and do almost as well as the best possible algorithm. A candidate such algorithm is *learn_trust*. The other algorithms provide points of comparison in evaluating the quality of *learn_trust*.

4.1 The *learn_trust* agent

A *learn_trust* agent tries to learn the reliability value of each other agent. It maintains a vector of real-valued registers in its memory, one for each other agent, which we call its *opinion* about that agent. All agents' opinions are initialized to the expected value of the random variable R used for the assignment of the reliability values. Its behavior is controlled by a strategy based on its opinions, which it uses as if they were the true reliability values.

A *learn_trust* agent performs two fundamental operations: It revises its opinions of other agents based on its own experiences and on those of other agents, and it uses its opinions to inform its actions. Hence, in order to define such an agent, we need to describe precisely how these operations are to be performed. For the agent to be feasible, these operations must also be efficiently computable.

Let agent pair (i, k) be invited to an encounter, and let $O_{i,k}$ be i's opinion about k's degree of reliability. As mentioned above, agent k always accepts the invitation. Agent i decides whether to accept or decline by use of the following formula:

$$\text{decision}_{i,k} = \begin{cases} \text{accept} & \text{if } \widehat{E}[\text{payoff}_{i,k}] \geq \tau_i \\ \text{decline} & \text{otherwise.} \end{cases} \tag{2}$$

$\widehat{E}[\text{payoff}_{i,k}] = O_{i,k} * P_g + (1 - O_{i,k}) * P_b$ is what agent i believes the expected payoff of an encounter with k is, and τ_i is a constant.

After the encounter takes place, agent i calculates its new opinion $O'_{i,k}$ of k according to a simple exponential-decay rule:

$$O'_{i,k} = (1 - \alpha_{\text{direct}}) * O_{i,k} + \alpha_{\text{direct}} * \nu. \tag{3}$$

α_{direct} is a decay constant that has the same value for all agents, and ν is 1 or 0 depending on whether χ is good or bad, respectively.

If the encounter is good, agent k shares all of its opinions with agent i, except for $O_{k,i}$ and $O_{k,k}$.[4] Agent i uses the information received from k to update its opinion about each other agent j, also according to a simple exponential-decay rule:

$$O'_{i,j} = (1 - \alpha_{\text{indirect}}) * O_{i,j} + \alpha_{\text{indirect}} * O_{k,j}. \tag{4}$$

a_{indirect} is a decay constant that has the same value for all agents (but is not necessarily the same as a_{direct}). If the encounter is bad, agent k shares no information with agent i.

[4]Recall that agents do not maintain opinions about themselves.

For the computer simulations, we set $\tau_i = 0$ for all i. All agents' opinion are initialized to 0.5, since $R \sim \text{unif}(0, 1)$. Notice that because we also assume $P_g = 2$ and $P_b = -2$, i decides to participate in an encounter with k iff $O_{i,k} \geq 1/2$. Finally, we set both decay constants α_{direct} and α_{indirect} to 0.1.

4.2 Memoryless agents

In order to establish a baseline for the *learn_trust* algorithm, we consider two kinds of memoryless agents:

Play_always: This agent always accepts an invitation to an encounter. Because each r_k is a uniformly distributed random variable, its expected value is 1/2. Hence, from equation 1, the expected earnings per encounter are

$$E[\text{earnings}] = \frac{1}{2}(P_g + P_b). \tag{5}$$

In our simulations, $P_g = -P_b = 2$, so $E[\text{earnings}] = 0$.

Play_never: This agent always declines an invitation to an encounter. The expected total earnings for the society are 0.

4.3 The *know_reliability* agent

The *know_reliability* agent is an idealized agent with perfect information about the reliability values of each other agent. Thus, it can play optimally. It accepts an invitation to an encounter if the expected payoff from the encounter is non-negative, and it declines otherwise. The expected payoff for an encounter with agent k with reliability r_k is given by (1). This number is non-negative if

$$r_k \geq \frac{-P_b}{P_g - P_b}. \tag{6}$$

Hence, the *know_reliability* agent sets

$$\tau = \frac{-P_b}{P_g - P_b}, \tag{7}$$

and it accepts an invitation to participate with agent k whenever $r_k \geq \tau$. Thus, from (1), the expected earnings for a *know_reliability* agent offered an encounter with agent k are

$$E[\text{earnings}_k] = \max(0, r_k P_g + (1 - r_k)P_b). \tag{8}$$

For the computer simulations, (7) simplifies to $\tau = 1/2$, and (8) simplifies to

$$E[\text{earnings}_k] = 4 \cdot \max\left(0, r_k - \frac{1}{2}\right). \tag{9}$$

Treating r_k as a random variable $\sim \text{unif}(0, 1)$, the expected value of $\max(0, r_k - 1/2)$ is

$$\int_0^1 \max\left(0, x - \frac{1}{2}\right) dx = \int_{1/2}^1 \left(x - \frac{1}{2}\right) dx = \frac{1}{8}. \tag{10}$$

From (9) and (10), the expected earnings per invitation for a *know_relability* agent is

$$E[\text{earnings}] = 4 \cdot (1/8) = 1/2. \tag{11}$$

However, for a particular society, the r_k's are fixed once and for all. Using (9), the expected earnings per invitation, given the r_k's, are

$$E[\text{earnings} \mid r_1 \dots r_n] = \frac{4}{n} \sum_{k=1}^{n} \max\left(0, r_k - \frac{1}{2}\right). \tag{12}$$

This averages to 1/2 over all possible values for the r_k's, but it could be greater or less than 1/2 for any particular society.

Because *know_reliability* accepts invitations to participate exactly when its expected payoff is non-negative, we have:

THEOREM 1. *No agent without prior knowledge of the outcome of encounters has larger expected earnings per invitation than know_reliability.*

5. ANALYSIS OF *LEARN_TRUST*

In order to determine how good the *learn_trust* algorithm is, we simulated it and compared its performance with that of the idealized *know_reliability* algorithm, which we know by Theorem 1 will outperform any implementable algorithm.

5.1 Simulation method

Our simulations comprise three parts. First, 10 random societies of 100 agents each are generated. Second, for each society, 10 random encounter schedules of 10^6 encounters each are generated. Third, each algorithm of interest is run 100 times, once for each society and each corresponding encounter schedule. The results from the 100 simulation runs are averaged together and the averages plotted on graphs. Although the socities and encounter schedules are randomly generated, the different algorithms are all run on the same data sets, making comparisons among the algorithms more meaningful.

The agents in each society are created with reliability values drawn from the distribution $\text{unif}(0, 1)$. An *encounter schedule* consists of a sequence of triples (i, k, χ). The pair (i, k) is chosen uniformly at random from $\{(i, k) \mid i, k \in G, i \neq k\}$. χ is chosen randomly from $\{\mathsf{good}, \mathsf{bad}\}$, where the probability that $\chi = \mathsf{good}$ is r_k.

Given a society, an encounter schedule, and an algorithm, the simulation proceeds step by step. At step t, the t^{th} encounter from the encounter schedule is used. Call it (i, k, χ). A simulation step proceeds as follows:

1. Agents i and k are invited to participate in an encounter, where i is the active agent and k the passive agent.
2. Agent i's decision rule is evaluated to see whether or not it chooses to accept the invitation. If it declines the invitation, nothing further is done and the simulation step is complete.
3. If agent i accepts the invitation, then the character of the encounter is χ.
4. Agent i's direct updating function is evaluated with χ as an argument.
5. If $\chi = \mathsf{good}$, then agent i's indirect updating function is evaluated with agent k's memory as an argument.

6. Agent i's accumulated earnings are incremented by P_g if $\chi = \mathsf{good}$ and by P_b if $\chi = \mathsf{bad}$.

5.2 Observations

5.2.1 Earnings per step

The first quantity of interest is the society earnings per step, averaged over the first t steps. This is the total society earnings through step t divided by t. Note that this quantity is undefined for $t = 0$.

In the case of *know_reliability*, it follows from (12) that the expected average earnings per step are

$$\frac{4}{10n} \sum_{s=1}^{10} \sum_{k=1}^{n} \max\left(0, r_k^s - \frac{1}{2}\right), \qquad (13)$$

where r_k^s is agent k's reliability value in the s^{th} society of the simulation ($1 \leq s \leq 10$). No algorithm can do better than this. We wish to see how close *learn_trust* comes to achieving this value.

The graphs of Figure 1 show the results of our simulations for the first 10^3, 10^4, 10^5, and 10^6 steps, respectively.[5] We observe from these graphs that *know_reliability* earns about 0.52 per encounter, slightly above its expected value of 0.5. This is presumably due to statistical variation in the samples of reliability values comprising the society. *Learn_trust* starts out behaving like *play_always* since each agent's initial opinion of 1/2 causes it to accept its first encounter. After approximately 3000 steps, *learn_trust*'s average earnings per encounter have risen to about 50% that of *know_reliability*. As time continues up to 10^6 steps, we see that *learn_trust* has average earnings about 0.5, only 4% below that of *know_reliability*. We can safely conclude that most agents at this point in time are making pretty accurate decisions about which encounters to accept and which to reject. In fact, we note that the same is true already after about 200,000 steps. *Learn_trust*'s performance improves very little after that time. Since the performance of *know_reliability* is an upper bound on the performance of any algorithm that operates without prior knowledge of the reliability values, we conclude:

Observation 1. After an initial learning period, the average performance of *learn_trust* agents appears to be close to optimal.

One might hope that the performance of *learn_trust* would approach that of *know_reliability* in the limit. However, the graphs do not seem to support that conclusion. Between 200,000 and 1,000,000 steps, the lines are pretty much flat and parallel. Analysis of the algorithm suggests three reasons why the performance of *learn_trust* might not get arbitrarily close to that of *know_reliability*.

1. Agent i's opinion of agent k does not converge to r_k. Indeed, even if $O_{i,k} = r_k$ just before an invitation to an encounter between i and k, it will not equal r_k

[5]Each graph shows 20 evenly spaced points from the simulation interval.

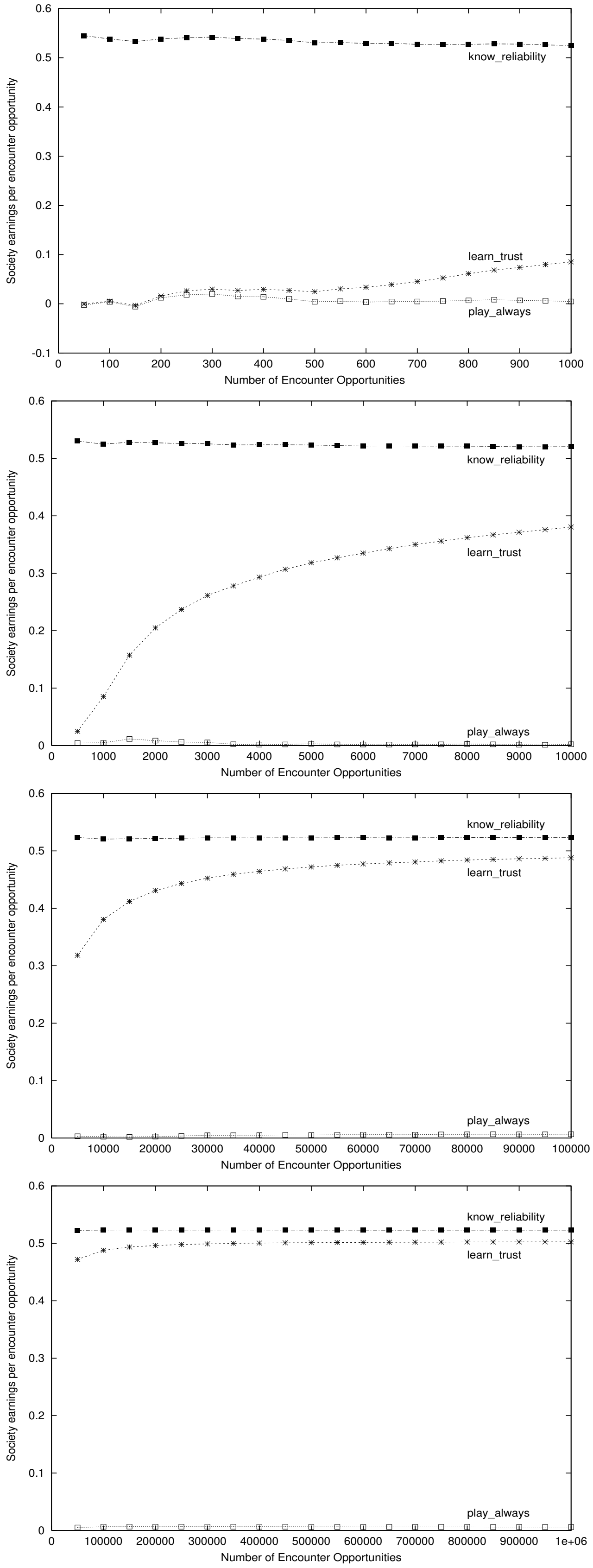

Figure 1: Society earnings per invitation.

afterwards if the invitation is accepted. This is because the new value of $O_{i,k}$ will be either $0.9 * O_{i,k} + 0.1 > O_{i,k}$ or $0.9 * O_{i,k} < O_{i,k}$, depending on whether the outcome is good or bad.

2. It might happen for some agent i that, at some point in time, its opinion of all other agents drops below 0.5. We call such an agent *paranoid* since it distrusts everyone. A paranoid agent declines all invitations to participate in encounters, so it never changes its opinions henceforth. A paranoid agent results in a loss of potential earnings for the society since it forgoes the positive expected payoffs from encounters with good agents that it would otherwise enjoy.

3. It might happen for some agent k that, at some point in time, it is distrusted by all agents $i \neq k$, that is, all have opinions $O_{i,k} < 0.5$ and all will refuse invitations to participate with k. We call such an agent an *outcast* and say that it has been *dropped*. This condition of being an outcast persists forever, for there is no way for the maximum opinion $\max_{i \neq k} O_{i,k}$ of the other agents to increase. Without further encounters with k, there will be no new direct information to cause opinions about k to change. While an individual opinion $O_{i,k}$ can increase as a result of indirect information from some agent j, its new value cannot be larger than the greater of its old value and $O_{j,k}$. Thus, the maximum opinion across the society cannot increase after indirect updates, so it remains below 0.5, and k remains an outcast. When a good agent k is dropped, every declined invitation to participate with k results in a loss of potential earnings for the society.

How important are these three factors in practice? Intuitively, the non-convergence of the opinion to the actual reliability value is not very important. The reason is that good performance only requires making the right decision most of the time, and the further away r_i is from 0.5, the greater the effect the decision has on earnings. But the further away r_i is from 0.5, the more likely it is that the right decision will be made even if the opinion differs considerably from r_i.

For the question of paranoid and outcast agents, we turn again to our simulations. In none of our simulations did we observe a paranoid agent, suggesting that it is highly unlikely for an agent to become paranoid.

Observation 2. Learn_trust rarely produces paranoid agents in practice.

To get some intuition about why this is true, note that *gossip* (the exchange of indirect information) tends to keep the other agents' opinions about an agent k clustered together. As we observed above, gossip cannot raise the maximum opinion about k; similarly, it cannot lower the minimum opinion. Moreover, if the holder of an extreme opinion receives gossip, its new opinion will be less extreme (except in the case that it received gossip from another equally extreme agent).

Now, to become paranoid, agent i's opinion of all other agents must simultaneously drop below 0.5. However, for an agent k with a high reliability value (say $r_k \geq 0.9$), the clustering phenomenon will tend to keep the other agents' opinions about k well above 0.5. While agent i's opinion about k might well drop below 0.5 as a result of a series of bad encounters with k (already a low probability event), gossip stemming from subsequent good encounters with other agents will tend to push its opinion of k upwards. So becoming paranoid requires the constellation of a large number of low probability events.

The situation with dropped agents is different. An agent k becomes an outcast when other agents' opinions drop below 0.5. The lower r_k, the more rapidly we expect this to happen. Indeed, it is by dropping bad agents that *learn_trust* is able to approach the performance of *know_reliability*.

Unfortunately, it is both possible and fairly likely for a good agent to be dropped, particularly near the beginning of the simulation run. If a good agent is the passive agent in a few bad encounters, it will get a bad reputation which then gets propagated throughout the society via gossip. Figure 2 shows the number of dropped good agents, averaged over the 100 runs, as a function of time. In the first 10,000 steps, only about .43 good agents are dropped. But after that the number rises steeply. Almost 3 are dropped after 20,000 steps. By 50,000 steps, the number is up around 6. After 1,000,000 steps, it has risen to above 7.5, and it appears not to be leveling off. From a total population of approximately 50 good agents in all, the 7.5 dropped represents a considerable fraction of the whole.

One would of course expect the rate at which good agents are dropped to decrease over time, if only because fewer and fewer good agents remain. But the good agents most likely to get dropped are those with low reliability values (those only slightly above 0.5). They are also the ones most likely to be dropped early on, so as time goes on, the reliability values of the population of remaining good agents tend to increase, making those agents less likely to get dropped in the future.

Observation 3. Learn_trust drops a considerable number of good agents, but the rate at which they are dropped increases at first and then gradually decreases.

It would be interesting to see what might happen were the simulation to be continued far beyond 10^6 steps. While we were not able to do this, we were able to simulate a much smaller society (only 10 agents) for $2 \cdot 10^7$ steps. Thus, each agent participated in approximately 200 times as many encounters as in the bottom graph of Figure 2. The results are shown in Figure 3. We note that an average of about 3.8 good agents are dropped after $2 \cdot 10^7$ steps. Since the expected number of good agents is only 5, this means that about 75% of the good agents have been dropped by the end of this run. Again the graph has a very noticeable positive slope, suggesting that all good agents will eventually be dropped. The next theorem justifies this claim.

THEOREM 2. *With probability 1, learn_trust eventually drops all agents.*

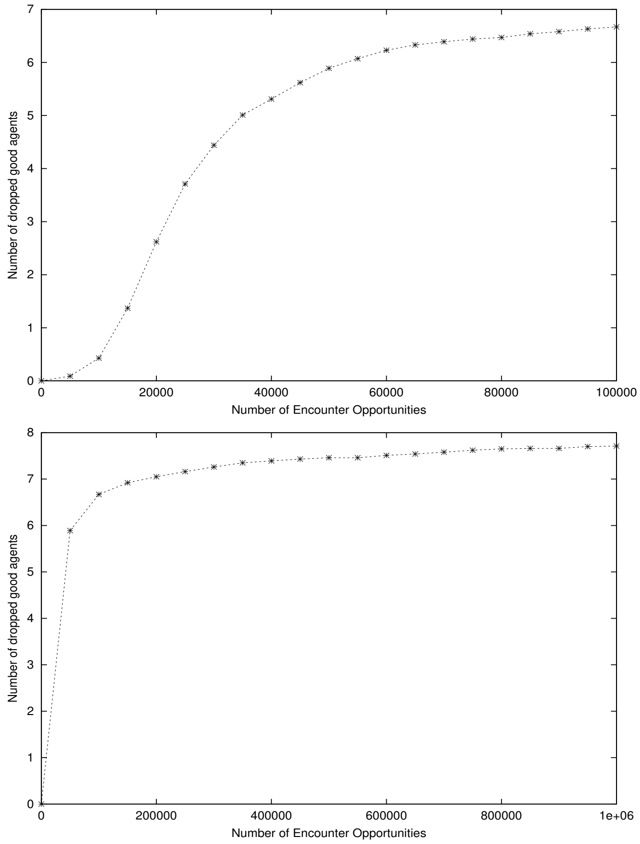

Figure 2: Dropped good agents for *learn_trust*.

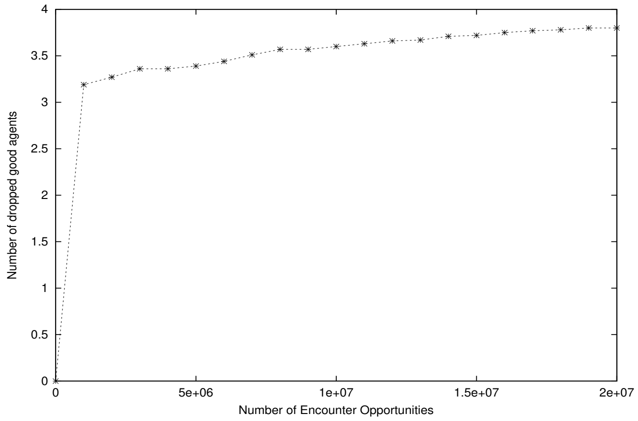

Figure 3: Dropped good agents in a 10-agent society.

PROOF. The proof depends on the fact that there is a fixed encounter sequence σ_k that will cause agent k to be dropped whenever it appears in the simulation. Let

$$t = \left\lceil -\frac{1}{\log_2(1-\alpha_{\text{direct}})} \right\rceil . \tag{14}$$

Let $\sigma_{i,k} = (i, k, \mathsf{bad})^t$, that is, t copies of the triple (i, k, bad). Let $\sigma = \sigma_{1,k} \ldots \sigma_{k-1,k}\sigma_{k+1,k} \ldots \sigma_{n,k}$ Thus, for each agent $i \neq k$, σ_k contains t consecutive bad encounters of i with k.

We must show that agent k is dropped after executing σ_k, no matter what the state of the society when σ_k begins. We must also show that σ_k has a fixed positive probability of occurring during the next $|\sigma_k|$ steps of the simulation.

The fact that agent i distrusts k by the end of σ_k follows from the facts that $O_{i,k} < 1$ at the beginning of σ_k, and $(1-\alpha_{\text{direct}})^t \leq 0.5$. Suppose to the contrary that agent i does not distrust k during σ_k. Then $O_{i,k} \geq 0.5$ throughout σ_k, which means that i accepts all t invitations to encounters with k. But since the character of each such encounter is bad, $O_{i,k}$ is reduced by a factor of $(1-\alpha_{\text{direct}})$ after each. If $O_{i,k}^{(0)}$ denotes agent i's opinion at the start of σ_k and $O_{i,k}^{(t)}$ is its opinion at the end, we have

$$O_{i,k}^{(t)} = (1-\alpha_{\text{direct}})^t O_{i,k}^{(0)} < 0.5, \tag{15}$$

contradicting the assumption that i doesn't distrust k. This proof relies on the fact that agent i never receives any indirect information about k during σ_k, which is true since agent i has no encounters with agents other than k during σ_k.

We now analyze the probability of the next $|\sigma_k|$ encounters being exactly those in σ_k. The probability is $1/(n(n-1)) > 0$ that the next invitation to an encounter will be between agents i and k, with i active and k passive. Given that it is, if the invitation is accepted, the probability is $(1-r_k) > 0$ that the outcome will be bad. Hence, the probability that the next encounter is consistent with σ_k is at least

$$\frac{1-r_k}{n(n-1)} > 0. \tag{16}$$

Since $|\sigma_k| = t(n-1)$, the probability of σ_k is at least

$$\epsilon_k = \left(\frac{1-r_k}{n(n-1)}\right)^{t(n-1)} > 0. \tag{17}$$

We now divide the simulation steps into blocks of length $|\sigma_k|$. With probability ϵ_k, the j^{th} block is equal to σ_k, in which case agent k is dropped. Once dropped, it remains dropped forever after, as has been previously observed. Since σ_k has positive probability, the probability is 1 that σ_k eventually occurs and k is eventually dropped. In fact, the expected number of blocks until k is dropped is $1/\epsilon_k$.

To complete the proof, observe that the above argument applies equally to all agents k. Since each k is eventually dropped with probability 1, all agents are eventually dropped with probability 1. The expected number of blocks before all agents are dropped is thus at most $\sum_k 1/\epsilon_k$. $\square$

Note that when all but one agent have been dropped, the remaining agent is paranoid, and when it is finally dropped, then all agents are paranoid. By Theorem 2, all agents are eventually dropped, establishing:

COROLLARY 3. *With probability 1, all agents eventually become paranoid in learn_trust.*

In contrast to Observation 2, Corollary 3 shows that all agents in *learn_trust* do eventually become paranoid, but only after a very long time.

Theorem 2 and its corollary show that in the long run, *learn_trust* actually does very poorly. Its performance deteriorates until eventually agents cease playing altogether, effectively reverting to the *play_never* strategy. This stands in sharp contrast to Observation 1, which shows that *learn_trust* does well for a very long time. Intuitively, there are two processes at work: Agents learn pretty quickly who the good and bad agents are, after which their play is close to optimal. However, in parallel with this, there is a random process that produces a false negative in regard to the reliability of the agents. With probability 1, a good agent will eventually behave badly enough for long enough to make the society believe it is bad. Unfortunately, a false negative results in permanent dropping of an agent, so once a good agent is dropped, the mistake is never henceforth corrected.[6]

One way to control the effects of the random dropping process is to lengthen the period of bad behavior needed to establish a false negative. A possible way to accomplish this might be by adjusting the values of the decay parameters in order to lengthen the time that agents retain old memories. The further the memory stretches into the past, the greater the number of past good encounters remembered about a good agent, which in turn dictates a longer period of future bad encounters needed for the agent to be perceived as bad. However, if the agents base their decisions to participate in encounters on longer memories, not only will the rate of dropping good agents decrease, but so will the rate of learning, since more encounters will be required for a bad agent to be identified as such. One of our future research directions is to investigate the tradeoff between the rate of learning and the rate of dropping good agents, as well as to calibrate the decay parameters in order to achieve the best possible performance for *learn_trust* for a given number of encounter opportunities.

6. ASYMPTOTICALLY GOOD AGENTS

An obvious question in light of Theorem 2 is whether *learn_trust* can be modified so as to perform better than *play_always* in the long run, or for that matter, whether there is any algorithm at all that has positive expected payoff per invitation in the limit.

We suggest three possibilities for avoiding the consequences of Theorem 2. The first is a birth/death model. Assume from time to time that agents die and are replaced by new agents whose memories are in the initial state for the algorithm. This is of course equivalent to simply resetting an agent's memory from time to time to the initial state.[7] This clearly prevents agents from being permanently dropped, since a newly-born agent is always willing to accept an invitation to participate. On the other hand, every time an experienced agent that has accurate opinions about the other agents dies, it is replaced by a new agent that will do less well for awhile until it learns through experience and gossip whom to trust. So there is a tradeoff here between the advantage of eliminating agents who distrust good agents and the disadvantage of losing valuable knowledge about which agents are bad.

Another possibility for circumventing Theorem 2 is to consider agent algorithms that are allowed to *whitelist* other agents. Once agent i whitelists agent k, it forever after accepts all invitations to participate with k. If all agents are eventually whitelisted or blacklisted, and if the average reliability value of the whitelisted agents is asymptotically positive, the asymptotic earnings per invitation to encounter will be positive rather than 0.

A third possibility for getting around Theorem 2 is to look for algorithms in which the probability of eventually dropping good agents is less than 1. This can occur for agent algorithms that make it harder and harder to drop agents as time progresses, so that an encounter sequence that causes everyone to distrust agent k if it occurs at step t will not necessarily have that effect if it occurs later than t. Depending on the details, the probability of eventually dropping k could be bounded by anything between 0 and 1.

We plan to explore these ideas in future work.

[6]False positives are also inevitable: a bad agent will eventually behave well for long enough to be perceived as good. However, this will only be a temporary belief, in contrast to the permanent dropping of a good agent caused by a false negative.

[7]The reset could be done by the agents themselves, either periodically (if they can remember how many invitations they have received since the last time they were reset), or probabilistically, so the underlying game need not be changed to incorporate the birth/death idea.

7. REFERENCES

[1] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. In *Proc. European Symposium on Research in Computer Security (ESORICS) 1994*, Lecture Notes in Computer Science, pages 3–18. Springer-Verlag, 1995.

[2] A. D. Birell, B. W. Lampson, R. M. Needham, and M. D. Schroeder. A global authentication service without global trust. In *Proc. IEEE Symposium on Security and Privacy*, 1986.

[3] M. Blaze, J. Feigenbaum, and A. Keromytis. Keynote: Trust management for public-key infrastructures. In *Proc. 1998 Cambridge Security Protocols International Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer, 1999.

[4] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. IEEE Symposium on Security and Privacy*, 1996.

[5] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, Feb. 1990.

[6] Y. H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for web applications. *World Wide Web Journal*, 2:127–139, 1997.

[7] M. Gasser, A. Goldstein, C. Kaufman, and B.Lampson. The digital distributed system security architecture. In *Proc. 12th ACM NIST/NCSC National Computer Security Conference*, pages 305–319, Oct. 1989.

[8] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proc. IEEE Symposium on Security and Privacy*, pages 234–248, 1990.

[9] S. Kent. Internet privacy enhanced mail. *Communications of the ACM*, 36(8):48–60, Aug. 1993.

[10] U. Maurer. Modelling a public-key infrastructure. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Proc. European Symposium on Research in Computer Security (ESORICS) 1996*, Lecture Notes in Computer Science, pages 325–350, New York, NY, 1997. Springer-Verlag.

[11] P. V. Rangan. An axiomatic theory of trust in secure communication protocols. In *Computers and Security*, pages 163–172. Elsevier Science Publishers Ltd., Oxford, 1992.

[12] M. K. Reiter and S. G. Stubblebine. Resilient authentication using path independence. *IEEE Trans. Comput.*, 47(12):1351–1362, Dec. 1998.

[13] M. K. Reiter and S. G. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2):138–158, May 1999.

[14] A. Tarah and C. Huitema. Associating metrics to certification paths. In *Proc. European Symposium on Research in Computer Security (ESORICS) 1992*, Lecture Notes in Computer Science, pages 175–189. Springer-Verlag, 1993.

[15] R. Yahalom, B. Klein, and T. Beth. Trust relationships in secure systems—a distributed authentication perspective. In *Proc. IEEE Symposium on Research in Security and Provacy*, pages 150–164, 1993.

[16] P. Zimmerman. *The Official PGP User's Guide*. MIT Press, 1995.