Managing Trust in a Peer-2-Peer Information System

Karl Aberer, Zoran Despotovic

Department of Communication Systems
Swiss Federal Institute of Technology (EPFL)
1015 Lausanne, Switzerland
{karl.aberer, zoran.despotovic}@epfl.ch

Abstract

Managing trust is a problem of particular importance in peer-to-peer environments as one encounters frequently unknown agents. Existing methods for trust management based on reputation do however not scale as they rely on some form of central database or global knowledge to be maintained at each agent. In this paper we illustrate that the problem needs to be addressed at both the data management and the semantic, i.e. trust management, level and we devise a method of how trust assessments can be performed by using at both levels scalable peer-to-peer mechanisms. We expect that such methods are an important factor if fully decentralized peer-to-peer systems should become the platform for more serious applications than simple file exchange.

Keywords. trust management, reputation, peer-to-peer information systems, decentralized databases.

BRIDGE paper.

1 Introduction

Over the last years, mainly due to the arrival of new possibilities for doing business electronically, people started to recognize the importance of trust management in electronic communities. Visitors at 'amazon.com' usually look for customer reviews before deciding to buy new books. Participants at eBay's auctions can rate each other after each transaction. But both examples use completely centralized mechanisms for storing and exploring reputation data. In this paper we want to explore possibilities for trust management in completely decentralized environments, Peer-To-Peer networks in particular, where no central database (or data warehouse) is available.

Peer-To-Peer (P2P) systems are driving a major paradigm shift in the era of *genuinely* distributed computing. Major industrial players believe "P2P reflects society better than

other types of computer architectures [4]. It is similar to when in the 1980's the PC gave us a better reflection of the user" (www.infoworld.com).

In a P2P infrastructure, the traditional distinction between clients and back-end (or middle tier application) servers is simply disappearing. Every node of the system plays the role of a client and a server. The node *pays* its participation in the global exchange community by providing access to its computing resources. Gnutella (www.gnutella.com) is a good example of a P2P success story: a rather simple software enables Internet users to freely exchange files, such as MP3 music files.

The importance of trust management in P2P systems cannot be overemphasized, as illustrated by investigation on Gnutella [3]. The examples justifying this statement range from the simplest possible case where, while downloading (music) files with a Gnutella client, we want to choose only reliable peers to the situation of the entire P2P community playing a role of a marketplace where trusting other peers can influence the whole business.

The basic problem related to trust management in P2P networks is that information about transactions performed between peers (agents) is dispersed throughout the network so that every peer can only build an approximation of 'the global situation in the network'. Of course this is further complicated by the fact that agents participating in storage cannot be considered as unconditionally trustworthy and their eventual malicious behavior must be taken into account, too.

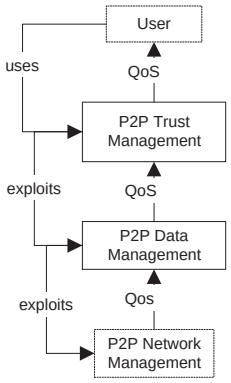

Figure 1: Different system levels of P2P computing

The approach to trust management that we present here can be seen as a simple method of data mining (rather statistical data analysis). We exploit that in a society of agents were cheating is comparably limited, what we assume also in daily life normally, it will become difficult to hide malicious behavior. The method allows to judge trust for agents even when one meets them for the first time by referring to the experiences other agents made. The method relies exclusively on peer-to-peer interactions and requires no centralized services whatsoever, both for trust assessment and data management. It is based on some very recent developments that have been made in the field of data management in decentralized (or peer-2-peer) information systems (P-Grid [2]). In a nutshell we present an architecture

for trust management which relies on *all system layers*, namely network, storage and trust management on peer-2-peer mechanisms. This is illustrated in Figure 1. It is important to observe that in such an architecture a mechanism implemented at a higher level in a peer-2-peer manner has always to take into account the properties, in particular the quality of service, of the mechanisms of the underlying layers. For example, the storage layer has to take into account the unreliability of network connections and thus would replicate data to make it accessible with sufficient reliability. The trust layer similarly has to take into account that not all data is equally well accessible when assessing trust based on statistical evidence derived from behavioral data.

In that sense this paper not only contributes to the question of how to manage trust, but shows also a prototypical case of how a full-fledged peer-2-peer architecture for information systems can be build.

In Section 2 we review existing work on formal trust models and their implementation. In Section 3 we identify the problems that need to be addressed when managing trust in a decentralized information system. On Section 4 we give an overview of the method we propose and that illustrates that the problem is tractable. In Section 5 we present the detailed algorithms and in Section 6 we give the simulation results showing that the method works effectively. We conclude the paper with some final remarks in Section 7.

2 Related Work

One of the first works that tried to give a formal treatment of trust that could be used in computer science was that of Marsh [6]. The model is based on social properties of trust and presents an attempt to integrate all the aspects of trust taken from sociology and psychology. But having such strong sociological foundations the model is rather complex and cannot be easily implemented in today's electronic communities. Moreover the model puts the emphasis on agents' own experiences only so that the agents cannot collectively build a network of trust.

An important practical example of reputation management is eBay (www.ebay.com), most likely the largest online auction site. After each transaction buyers and sellers can rate each other and the overall reputation of a participant is computed as the sum of these ratings over the last six months. Of course a main characteristics with this approach is that everything is completely centralized at the data management level.

Rahman and Hailes [1] proposed a method that can be implemented in P2P networks. This work is based on Marsh's model. Actually it is a kind of adaptation of Marsh's work to today's online environments. Some concepts were simplified (for example, trust can have only four possible values) and some were kept (such as situations or contexts). But the main problem with this approach is that every agent must keep rather complex and very large data structures that represent a kind of global knowledge about the whole network. In real word situations maintaining and updating these data structures can be a labourous and time-consuming task. Also it is not clear how the agents obtain the recommendations and how well the model will scale when the number of agents grows.

Another work is that of Yu and Singh [7]. This model builds a social network among agents that supports participants' reputation both for expertise (providing service) and helpfulness (providing referrals). Every agent keeps a list of its neighbors, that can be changed over time, and computes the trustworthiness of other agents by updating the current values by testimonies obtained from reliable referral chains. After a bad experience with another agent every agent decreases the rating of the 'bad' agent and propagates this bad experience throughout the network so that other agents can update their ratings accordingly. In many ways this approach is similar to the previously mentioned work and the same remarks can be applied.

Also we would like to emphasize that none of these works discusses the data management and retrieval problems that certainly exist in distributed environments. We think that the question of choosing the right model to assess trust and the question of obtaining the data needed to compute trust according to the model cannot be investigated in separation.

3 Managing Trust in a Decentralized System

A common feature of the models of trust investigated in the literature is that they exploit information on the behavior of other agents in order to assess trust. From a global perspective it should be possible to assess trust based on all the behavioral data that is available on an agent.

The notion of context is of great importance when considering trust. The sentence 'I trust my doctor for giving me advice on medical issues but not on financial ones' is only one example that shows how important contexts can be. However, only for the sake of simplicity, trust is in the following considered for one static context, meaning that we do not distinguish the evaluation of trust in different contexts. But the context considerations could be easily integrated into the model.

Let P denote the set of all agents. The behavioral data B are observations $t(q,p)$ an agent $q \in P$ makes when he interacts with an agent $p \in P$. Based on these observations one can assess the behavior of p based on the set

$$B(p) = \{t(p,q) \text{ or } t(q,p) \mid q \in P\} \subseteq B$$

This means we take into account all reports about transactions that are made about p, but as well all reports about transactions that are made by p. In addition the global, aggregate behavior B of the system is used in order to scale results obtained about a specific agent.

Analysis of large amounts of behavioral data is in fact an important application of data mining. Transaction data in business applications is analyzed in order to assess certain characteristics of customers, in particular also their trustworthiness. All these methods have in common that they rely on potentially large amounts of behavioral data in order to assess trust.

When studying methods to assess trust in a decentralized environment we have to face now two questions:

- The semantic question: which is the trust model that allows to assess trust of p based on the data $B(p)$ and B ?
- The data management question: how can the necessary data $B(p)$ and B be obtained to compute trust according to the trust model with reasonable effort ?

Looking at these two questions more closely one sees that they cannot be investigated in separation. A powerful trust model is worthless if it cannot be implemented in a scalable manner, because it requires, for example, some centralized database.

However, with trust another factor comes into play that complicates the situation additionally. The problem of how to efficiently manage the behavioral data in trust management is not just an ordinary distributed data management problem. Each agent providing trust related data about others needs in turn also to be assessed with respect to its own trustworthiness. Thus, we cannot obtain data for determining trust without knowing about the trust we can put into the data sources.

More concretely, if an agent q has to judge on an agent p, the problem is that it has no access to the global data $B(p)$ and B. Rather it has to rely on that part of the information that it has obtained from direct interactions and that it can obtain indirectly through a limited number of referrals $r \in W_q \subseteq P$. Thus q has as information

$$B_q(p) = \{t(q,p) \mid t(q,p) \in B\}$$

and

$$W_q(p) = \{t(r,p) \mid r \in W_q, t(r,p) \in B\}$$

The problem is of course that $t(r,p)$ is not necessarily correct as also a witness r can be malicious. Thus there exists the additional and principal problem that the necessary data to assess trust cannot be obtained in a reliable manner.

Finally, even if the referring agent is honest it may not be reachable reliably over the network. This might distort the quality of the data it receives about the behavior of other agents.

Thus to formulate the problem of managing trust in a decentralized information system we can partition it now, more precisely, into three subproblems that need to be studied:

1. The global model of trust: What is the model that describes whether an agent behaves trustworthy or not ? This model could be based on simple statistical measures, on experiences gained in economic and sociological sciences or on game-theoretic foundations.
2. The local algorithm to determine trust: What is the computational procedure that an agent can apply in order to determine trust under the limitations discussed above,

namely the unreliability of the agnents providing trust data both with respect to their trustworthiness themselves as well as their reachibility over the network. To what extent does this local algorithm provide a good approximation of the global model?

3. The data and communication management: Can the local algorithm be implemented efficiently in a given data management infrastructure? In order to obtain scalable implementations the resources required by each agent must scale gracefully in the agent population size n, ideally as $O(\log n)$.

It is important to observe that these questions are to be answered independently of which specific model of trust is used. They are generic to the problem of managing trust in a distributed amd decentralized environment.

In the following we demontstrate that a solution is feasible. We propose a simple yet effective method, that implements a fairly straightforward trust model but encompasses all the relevant aspects. This model may be considered as an interesting result in itself as well as a starting point for developing further more sophisticated refinements of the method.

4 Overview of the Trust Management Method

4.1 Global Trust Model

The global trust model we consider can be seen as a simplification of the model discussed in [7], with exception of the mechanisms used for witnessing. The model is based on binary trust, i.e. an agent is either trustworthy or not. Agents perform transactions and each transaction $t(p, q)$ can be either performed correctly or not. If an agent p cheats within a transaction it becomes from the global perspective untrustworthy.

In order to disseminate information about transactions agents can forward it to other agents. Since we assume that usually trust exists and malicious behavior is the exception we just consider information on dishonest interactions as relevant. Thus an agent p can in case of malicious behavior of q file a complaint $c(p, q)$. Complaints are the only behavioral data B used in the model.

Let us first look at a simple situation where p and q interact and r later wants to determine the trustworthiness of p and q. We assume that p is cheating and q is honest. After their interaction (assuming p and q are acting rational in a game theoretic sense) q will file a complaint about p, which is perfectly fair. On the other hand also p will file a complaint about q in order to hide its misbehavior. The outside observer r can as a result not distinguish whether p or q is dishonest. This is an important point. A social mechanisms to detect dishonest behavior will not work for private interactions.

The trouble for p starts when it continues to cheat. Assume it cheats in another interaction with s. Then r will observe that p complains about both q and s, whereas both q and s complain about p. It will conclude that it is very probable that p is the cheater.

Generalizing this idea we define a (dis-)trust measure as the product

$$T(p) = |\{c(p,q) \mid q \in P\}| \times |\{c(q,p) \mid q \in P\}|$$

The problem is that we have given a global definition for trust based on the global knowledge on complaints. Whereas it is straightforward for an agent to collect all information about its own interactions with other agents, it is very difficult for it to obtain all the complaints about any other specific agent. From a data management perspective the data is aggregated along the wrong dimension, namely the first argument of $c(p,q)$ rather than along the second. Here is where data management technology can come in, in particular recent developments that have been made for managing data in decentralized (or peer-2-peer) information systems.

4.2 Decentralized Data Management

We will use a method that we have proposed, namely P-Grid [2], in order to store data in a peer-2-peer network in a decentralized and scalable fashion. Without going into the details of how it works we describe here the main properties of the access method that are important for the following discussion. Other, similar methods are currently proposed, like [5], and could be considered as well.

We show in Figure 2 a simple example of the access structure we use. 6 agents support there together a binary search tree of depth 2. Each level corresponds to a level in the search tree and at each level we see the intervals corresponding to the paths of the binary tree, i.e. 0 and 1 at level 1, 00, 01, 10, and 11 at level 2. At level 0 all 6 agents are contained in the root interval, meaning that any search request can be directed to any agent. When a search requests arrives at a level the agent checks whether it is responsible for it at the next level (indicated by its presence in the interval in the Figure) or whether it has to route it to another agent (indicated by a connector from the agent to another agent in the complimentary interval at the next lower level). The resulting routing of two example queries (001 and 100) is shown in the Figure.

At the leaf level the agents store complaints about the agents whose identifier corresponds to the search key, using the encoding $1 = 001, 2 = 010, \ldots, 6 = 110$. One can see that multiple agents can be responsible for the complaints on a specific agent. Thus we have *replicas* of this data. They make the access structure robust against failures in the network.

As the example shows, collisions of interest may occur where agents are responsible for storing complaints about themselves. We do not exclude this, as for large agent populations these cases will be very rare and multiple replicas will be available to doublecheck.

One can see that the access method is organized in a peer-2-peer manner, i.e. there exists no central database. It allows requests of the form:

- $insert(a, k, v)$ where a is an arbitrary agent in the network, k is the key value to be searched for, and v is a data value associated with the key.
- $query(a, k) : v$ which returns the data values v for a corresponding query k.

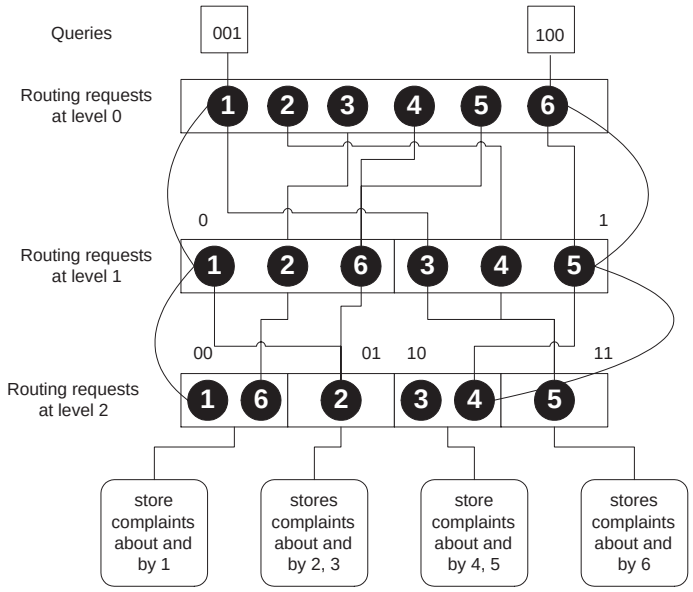

Figure 2: Example P-Grid

In [2] we have shown that this access structure satisfies the following properties.

- There exists an efficient decentralized bootstrap algorithm, based on bilateral communication of agents, which creates the access structure without central control. This algorithm is randomized and requires minimal global knowledge of the agents, namely an agreement on the key space. The mutual communications basically lead to a subsequent partitioning of the search space among agents that interact. Depending on the number of agents and the maximal number of possible keys, multiple agents will store data belonging to the same key, i.e. replicas are generated in the bootstrap.

- The search algorithm consists of forwarding the requests from one agent to the other, according to routing tables that have been constructed in the bootstrap algorithm. The search algorithm is randomized as requests are forwarded to a randomly selected reference, in case multiple references for routing a request exist, as it is normally the case. The access probability to different replicas is not uniformly distributed, which is of importance for the algorithms discussed later.

- All algorithms scale gracefully. In particular search is done in $O(\log n)$ time when n is the number of agents, and the storage space required at each agent scales also as $O(\log n)$.

4.3 Local Computation of Trust

We propose now a mechanism for computing trust using a P-Grid as storage structure for complaints. Every agent p can file a complaint about q at any time and store it by sending a message $insert(a, key(p), c(p, q))$ and $insert(a, key(q), c(p, q))$ to an arbitrary agent a. (We can assume that if p is not willing to send $insert(a, key(q), c(p, q))$ in order to hide the fact that it is complaining some other agent involved in processing $insert(a, key(p), c(p, q))$ will do that habitually). The insertion algorithm forwards the complaints to one or more agents storing complaints about p respectively q. In this way the desired reaggregation of the complaint data is achieved.

When an agent wants to evaluate the trustworthiness of another agent it starts to search for complaints on it. Based on the data obtained it uses the trust function T to decide upon trustworthiness.

When it has found the agents that store the complaints it faces a problem. The agent providing the data could itself be malicious. This can be dealt with by checking in a next step the trustworthiness of the agent that stores the complaints and so on. Eventually this would lead to the exploration of the whole network which is clearly not what we desire. Therefore we proceed as follows.

We assume that the agents are only malicious with a certain probability $\pi \leq \pi_{max} < 1$. Then we configure the storage infrastructure in a way that the number r of replicas is on average such that $\pi_{max}^r < \epsilon$, where ϵ is an acceptable fault-tolerance.

Thus, if we receive the same data about a specific agent from a sufficient number of replicas we need no further checks. If the data is insufficient or contradictory we continue to check. Practically, we will also limit the depth of the exploration of trustworthiness of agents, and might end up in situations where no clear decision can be made. These cases should be rare.

An interesting observation relates to the agents that route the search requests. In fact, if we assume a network with unrestricted connectivity (like the Internet) their trustworthiness does not play a role with respect to obtaining incorrect results. Either they route a request and help to find an agent storing specific complaints, and then we contact that agent directly, or they don't. Only in networks where the complete communication needs to be routed through malicious agents (like in a mobile ad-hoc network) this would be an additional factor to take into account.

5 Algorithms

In the following we describe a specific implementation of the method described before and its evaluation.

5.1 Procedure for Checking Complaints

When an agent p evaluates the trustworthiness of an agent q it retrieves from the decentralized storage complaint data by submitting messages $query(a, key(q))$ to arbitrary agents a. In order to obtain multiple referrals it will do this repeatedly, say s times. As a result it obtains a set

$$W = \{(cr_i(q), cf_i(q), f_i) \mid i = 1, \ldots, w\}$$

where w is the number of different witnesses found, f_i is the frequency with which each of those witnesses is found and $\sum_{i=1}^{w} f_i = s$. $cr_i(q)$ and $cf_i(q)$ are the number of complaints, that q has received respectively filed, and that witness i reports. Different frequencies f_i indicate that not all witnesses are found with same probability due to the non-uniformity of the P-Grid structure. In practice this variations can be rather large. This non-uniformity impacts not only query messages but also storage messages. Thus witnesses found less frequently will probably also not receive as many storage messages when complaints are filed. Thus the number of complaints they report will tend to be too low. Therefore we normalize the values by using the frequencies observed during querying with the following function that compensates for the variable probability of an agent to be found,

$$cr_i^{norm}(q) = cr_i(q)(1 - \frac{s - f_i}{s})^s, \ i = 1, \ldots, w$$

and analoguously for $cf_i^{norm}(q)$. We eliminate the witnesses that are found only once as any conclusion about the probability of their occurrence would be too unreliable.

Once multiple of these values are given a decision criterion is needed to decide when to consider an agent trustworthy. This has to be based on the trust measure T and on the experiences an agent p has made. Thus p keeps a statistics of the average number of complaints received and complaints filed, cr_p^{avg} and cf_p^{avg}, aggregating all observations it makes over its lifetime.

Based on these values we use then the following function to decide whether an agent is trustworthy, where 1 represents trust and 0 mistrust.

$$decide_p(cr_i^{norm}(q), cf_i^{norm}(q)) =$$

$$\textbf{if } cr_i^{norm}(q) cf_i^{norm}(q) \leq (\tfrac{1}{2} + \tfrac{4}{\sqrt{cr_p^{avg} cf_p^{avg}}})^2 cr_p^{avg} cf_p^{avg} \textbf{ then } 1 \textbf{ else } 0$$

This criterion is a heuristics based on the argument that, if an observed value for complaints exceeds the general average of the trust measure too far, the agent must be dishonest. The problem is the determination of the factor by which it may exceed the average value at most. The one we have chosen for this function is motivated by a probabilistic analysis of the underlying Poisson process of filing complaints, which is too complex to be presented within the framework of this paper.

5.2 Procedure for exploring trust

By obtaining the number of complaints in the way described an agent can now start to assess trust. The simplest strategy is to take a majority decision and in case of a tie return "undecided".

```
/* p is the agent assessing trust
   q is the agent to be assessed
   l is the limit on the recursion depth of exploration
   The result of the algorithm can be
   1 = trustworthy, 0 = no assessment possible, -1 = not trustworthy */
ExploreTrustSimple(p, q, l)
IF l>0
  W = GetComplaints(q);
  /* using the procedure described before, returns a number of witnesses and
     the numbers of complaints they report and normalizes them */
  update statistics with W;
  s = SUM(i = 1..|W|, decide(cr_i, cf_i));
  IF s > 0 RETURN 1
  ELSE IF s < 0 RETURN -1 ELSE RETURN 0
```

This does not include the checking of the witnesses themselves. Therefore we consider also a more sophisticated procedure for checking which is the following.

```
ExploreTrustComplex(p, q, l)
IF l>0
  W = GetComplaints(q);
  update statistics with W;
  IF |W|=0 RETURN 0;
  IF |W| = 1
     t = ExploreTrustComplex(p, r, l-1)
     IF t = 1 RETURN decide(cr_1, cf_1)
     ELSE RETURN 0;
  IF |W| > 1
     s = SUM(i = 1..|W|, decide(cr_i, cf_i));
     IF s > 1 RETURN 1
     ELSE IF s < 1 RETURN -1
     ELSE
        FOR i = 1..|W|
          IF ExploreTrustComplex(p, i, l-1) < 1
             W := W \ {entry i of W};
             /* eliminate the non-trusted witnesses */
        s = SUM(i = 1.. |W|, decide(cr_i, cf_i));
        IF s > 0 RETURN 1
        ELSE IF s < 0 RETURN -1 ELSE RETURN 0
```

The assumption underlying this algorithm is that the probability π that an agent is not trustworthy is higher than the tolerance for a wrong assessment, i.e. $\pi \geq \epsilon$, but that two witnesses giving the same assessment are acceptable, i.e. $\pi^2 < \epsilon$. Therefore in case of a single witness it always needs to be checked. Similarly, if the absolute difference between negative and positive assessments is smaller or equal 1, the witnesses need to be checked. If after the check a majority decision can be made, it is accepted.

6 Evaluation

In the simulations we evaluate the reliability of the trust assessments made. We take into account three important factors. The size and nature of the agent population, the amount of resources used for trust assessment and the experience of agents from interacting with other agents. Our basic assumption was that relatively few agents cheat, but we would like to better understand what "few" means. In addition we will compare the simple and complex algorithm for trust assessment.

6.1 Simulation Setting

We have implemented the algorithm in the computer algebra package Mathematica. We chose an agent community of 128 agents. For storage we use binary keys of length 5 and 4 to address agents, i.e. on average thus 4 respectively 2 replicas of the occuring complaints on and by agents are kept. We use for all experiments the same P-Grid in order to exclude effects emerging from variations within the P-Grid structure.

We compare agent populations including $k = 4, 8 \ldots, 32$ cheaters. We consider two kinds of cheater populations. In the first population all cheaters cheat uniformly with probability $\frac{1}{4}$. In the second population, which is more realistic, the cheaters cheat with variable probability of $\frac{1}{i}$, for $i = 1, \ldots k$ for cheater i. This population includes many agents which cheat very rarely and should thus be hard to identify as cheaters.

The experiment proceeds as follows

1. A P-Grid for the choosen key length 4 or 5 is built.

2. We perform 6400 and 12800 random interactions among agents in which they can cheat, i.e. each agent has on average 100 respectively 200 interactions. During that phase complaints are stored in the P-Grid whenever two agents meet of which at least one is dishonest and is cheating in that meeting according to its individual cheating probability.

3. The trust assessment is performed. 4 honest agents evaluate the trust of 100 randomly chosen agents among the remaining 124 agents. The random selection is done in order to exclude any effects resulting from a specific order in which the agents are assessed and statistical information is built from that. For example, checking only honest agents in the beginning would make the assessment more sensitive for subsequent

malicious agents, whereas if only malicious agents occur at the beginning, they could be overtrusted. The agents performing the assessment query the network 15 times in order to obtain data from witnesses. In the trust assessment malicious agents serving as witnesses cheat again according to their individual cheating probability by returning random numbers when being asked as a witness.

The simulations result in the number of correct, undecided and incorrect assessments made, distinguishing among honest and cheating agents.

6.2 Results

First, we evaluate the quality of the trust assessment for each parameter setting by aggregating all 100 assessments of all the 4 assessing agents for all 8 cheater population sizes $(8, \ldots, 32)$. The aggregation function is

$$\frac{(u_{honest} + 2 * w_{cheaters})}{total}$$

where u_{honest} is the number of assessments made where the assessed agent was honest but no decision was made and $w_{cheaters}$ is the number of assessments made where the agent was malicious, but was assessed as being honest.

The rationale for this function is that undecided cases are treated as cheaters. This corresponds to a model of a cautious assessment strategy, assuming that trusting a cheater is potentially more harmful than missing an opportunity for interacting with an honest agent. Thus also the weight for $w_{cheaters}$ is set to 2.

This allows us to make a first rough judgement on the quality of each parameter setting. The results are as follows for the cheater population with constant cheating probability of $\frac{1}{4}$.

# replicas	# interactions	decision algorithm	quality
2	100	simple	0.0537
2	200	simple	0.0587
4	100	simple	0.0515
4	200	simple	0.0478
2	100	complex	0.0578
2	200	complex	0.0587
4	100	complex	**0.0190**
4	200	complex	**0.0228**

The results show that only the combination of using a larger number of replicas and the complex decision criterion substantially increases the quality of the assessments. The differences among the other assessments are not substantial, though their general quality is rather good, i.e. only a very low fraction of the total number of assessments lead to wrong decisions (we can think of the quality measure as a probability of making an error weighted by the expected cost of doing so).

For the cheater population with variable cheating probability the results where obviously not as reliable as the increasing number of cheaters with very low cheating probability becomes difficult to distinguish from the honest agents.

# replicas	# interactions	decision algorithm	quality
2	100	simple	0.1478
2	200	simple	0.0637
4	100	simple	0.1300
4	200	simple	0.0700
2	100	complex	0.1618
2	200	complex	0.0684
4	100	complex	0.2047
4	200	complex	**0.0556**

In contrast to the pevious result, for the cheater population with variable cheating probability the number of interactions, corresponding to the experience of the agents, becomes essential. Among those assessments again the ones made using the combination of the larger number of replicas and the complex decision criterion are the best, though the difference there is not as substantial.

Next we give the detailed results for the optimal parameter settings for the different sizes of the malicious agent population. We denote $c_{cheater}$ as the number of cheaters correctly identified, $u_{cheater}$ as the number of cheaters for which no decision is made, $w_{cheater}$ as the number of cheaters which are wrongly assessed, and analogously c_{honest}, u_{honest} and w_{honest} for the honest agents. For the case of a constant cheating probability we get the following.

# cheaters	$c_{cheater}$	$u_{cheater}$	$w_{cheater}$	c_{honest}	u_{honest}	w_{honest}
4	24	0	0	376	0	0
8	20	0	0	379	1	0
12	39	1	0	357	2	1
16	52	0	0	343	5	0
20	100	0	0	289	6	5
24	125	3	0	252	18	2
28	110	2	0	272	10	6
32	137	3	0	243	9	8

For the case of variable cheating probability we get.

# cheaters	$c_{cheater}$	$u_{cheater}$	$w_{cheater}$	c_{honest}	u_{honest}	w_{honest}
4	4	0	0	396	0	0
8	8	0	0	391	1	0
12	35	1	0	361	3	0
16	60	0	0	338	2	0
20	109	2	5	284	0	0
24	48	2	26	319	3	2
28	93	5	18	283	1	0
32	55	3	30	304	8	0

One can see that the method works almost perfectly in the case of constant cheating probability up to a quite large cheater population. In fact with 32 agents cheating already $\frac{1}{4}$ of the whole population cheats. When comparing with the detailed results using the simple assessment algorithm (which we omit for space reasons) one can see that the main advantage of using the complex assessment algorithm is, that it can resolve many more of the undecided cases. In the case of variable cheating probability the method is quite robust up to 20 agents, but from then more and more agents, that cheat rarely, will be considered as honest.

One might observe that in the experiments the number of experiences made by each agent is of the same order of magnitude as the agent population itself, such that a form of global knowledge is aquired. In fact, we made the simulations with a relatively low number of agents in order to keep the computational cost of the simulation within limits. Running the same experimental setting with 1024 agents where $32, 64, \ldots, 256$ of them are cheating with constant probabilty $\frac{1}{4}$ and where each agent has 200 encounters, shows that the results are of the same quality with a quality factor of 0.02825. This indicates that there exists no (substantial) dependency of the method between the total population size and the amount of experience a single agent collects. This is not unexpected as long as the agent population is built up in a uniform manner.

Summarizing, the experimental evaluations confirm by large what we expected from the method, and they show that a detection of cheating agents is in fact possible in a pure peer-to-peer environment based on a fully decentralized reputation management.

7 Conclusions

We have identified the questions to be addressed when trying to find a solution to the problem of trust assessment based on reputation management in a decentralized environment. We have introduced and analyzed a simple, yet robust method that shows that a solution to this problem is feasible. As this is a first contribution to this problem, a number of issues for future research remain open. First, we see a need for more extensive evaluations of the method over a wider parameter range and using more detailed evaluation criteria. Second, probabilistic analysis are to be made which analyse the underlying probabilistic processes and eventually lead to more precise methods. We made already first steps into that direction which lead in particular to a formal justification of the decision criterion used in the algorithm. Third, we plan to incorporate these mechanisms into actually existing P2P applications, like file sharing systems, and thus to improve their quality for the users and to obtain feedback on practicability.

References

[1] A.Abdul-Rahman and S. Hailes: *Supporting Trust in Virtual Communities* Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000.

[2] K. Aberer: *P-Grid:A self-organizing access structure for P2P information systems* EPFL Technical Report, DSC/2001/016, (to appear at COOPIS 2001, Trento, Italy), 2001.

[3] E. Adar and B. A. Huberman: *Free riding on Gnutella* Technical report, Xerox PARC, 10 Aug. 2000.

[4] D. Clark. *Face-to-Face with Peer-to-Peer Networking.* IEEE Computer, January 2001.

[5] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, Hari Balakrishnan: *Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service* Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), 2001.

[6] S.Marsh: *Formalising Trust as a Computational Concept* Ph.D. Thesis, University of Stirling, 1994.

[7] Bin Yu and Munindar P. Singh: *A Social Mechanism of Reputation Management in Electronic Communities* Proc. of the 4th International Workshop on Cooperative Information Agents, Matthias Klusch, Larry Kerschberg (Eds.), Lecture Notes in Computer Science, Vol. 1860, Springer, 2000.