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 where one frequently encounters unknown agents. Existing methods for trust management, that are based on reputation, focus on the semantic properties of the trust model. They do not scale as they either rely on a central database or require to maintain global knowledge at each agent to provide data on earlier interactions. In this paper we present an approach that addresses the problem of reputation-based trust management at both the data management and the semantic level. We employ at both levels scalable data structures and algorithms that require no central control and allow to assess trust by computing an agents reputation from its former interactions with other agents. Thus the method can be implemented in a peer-to-peer environment and scales well for very large numbers of participants. We expect that scalable methods for trust management 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.

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. A P2P system can be characterized by a number of properties: no central coordination, no central database, no peer has a global view of the system, global behavior emerges from local interactions, peers are autonomous, and peers and connections are unreliable.

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 the investigation of 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 an entire P2P community playing the role of a marketplace, where trusting other peers is a crucial prerequisite for performing business.

The basic problem related to reputation-based 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 storing and processing trust related data cannot be considered as unconditionally trustworthy and their eventual malicious behavior must be taken into account, too.

The approach to trust management that we present in this paper is based on analyzing earlier transactions of agents and deriving from that the reputation of an agent. The reputation is essentially an assessment of the probability that an agent will cheat. Thus, the method can be interpreted as a simple method of data mining using statistical data analysis of former transactions. The data necessary for performing the analysis is provided by a decentralized storage method (P-Grid [2]).

The approach works if we make an assumption that we usually also make in daily life. If the probability of cheating within a society is comparably low, it becomes more difficult

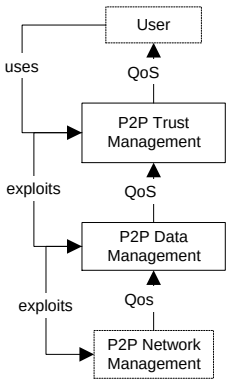

Figure 1: Different system levels of P2P computing

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.

We present an architecture for trust management which relies on *all system layers*, namely network, storage and trust management, on peer-to-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-to-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 will use data replication to increase reliability. Similarly, the trust layer has to take into account, that not all data is equally accessible when assessing trust based on statistical evidence derived from transaction 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-to-peer architecture for information systems can be built.

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. In 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 [5]. 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 such 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 characteristic 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 [8]. 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 apply.

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

Before turning our attention to the technical aspects of trust management we would like to give some terminological clarifications. Mutual trust allows agents to cooperate in a game-theoretic situation that corresponds to the repeated prisoners dilemma and leads in the long term to an increased aggregated utility for the participating agents. Trust management is any mechanism that allows to establish mutual trust. Reputation is a measure that is derived from direct or indirect knowledge on earlier interactions of agents and is used to assess the level of trust an agent puts into another agent. Thus, reputation-based trust management is one specific form of trust management.

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.

We describe now the problem of reputation-based trust management more formally. 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 the simplest case, when data is available globally, the reputation of an agent p can be derived from $B(p)$. In addition the global, aggregate behavior B of the system can be used in order to normalize results obtained about a specific agent.

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

- The semantic question: which is the 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 semantic 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 additionally complicates the situation. 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 data on earlier interactions 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 formally, an agent q, that has to determine trustworthiness of an agent p, has no access to the global data $B(p)$ and B. Rather, it has to rely on that part of the data that it has obtained from direct interactions and that it can obtain indirectly through a limited number of referrals from witnesses $r \in W_q \subseteq P$. Thus q has as information

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

and

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

to determine the reputation of an agent p. 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 received 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. Global trust model: What is the model that describes whether an agent is trustworthy ? This model could be based on simple statistical methods, on experiences gained in economic and sociological sciences or on game-theoretic foundations.

2. 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, which are the unreliability of the agents providing trust data both with respect to their trustworthiness themselves as well as their reachability over the network. To what extent does this local algorithm provide a good approximation of the global model?

3. 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 occur independently of which specific model of trust is used. They are generic to the problem of managing trust in a distributed and decentralized environment.

In the following we demonstrate that a solution is feasible. We propose a simple yet effective method, that implements a fairly straightforward reputation-based 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 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 later on r 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. An outside observer r can therefore not distinguish whether p or q

is the dishonest agent. 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 the reputation of an agent as the product

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

High values of $T(p)$ indicate that p is not trustworthy. The problem is, that we determine the reputation 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 the second. Here we arrive at the point where we have to study the problem from the perspective of data management.

4.2 Decentralized Data Management

In order to store data in a P2P network in a scalable way we use a method that we have proposed earlier, namely P-Grid [2]. Without going into too much detail we describe the main properties of this access method that are important for the subsequent discussion. Other, similar methods are currently proposed, like [7] or [6], and could be considered as well.

We show in Figure 2 a simple example of a P-Grid. 6 agents support there together a virtual binary search tree of depth 2. Each agent is associated with one path of the search tree. It stores data items for which the associated path is a prefix of the data key. For the trust management application this are the complaints indexed by the agent number. Each agent can serve any search request. Either the requested key has as prefix the search path associated with the agent processing the request, or the agent can use its routing table to forward the query to another agent that is responsible for the complementary part of the search tree. We demonstrate in the figure the processing of one sample query $query(6, 100)$ using this search structure. As agent 6 is not associated with keys starting with 0 it looks up in its routing table agent 5, to whom it can forward the query. Agent 5 in turn cannot process a query starting with 10 and therefore looks up in its routing table agent 4, who can finally answer the query, as it stores all data with keys, that start with 10.

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, the same data can be stored at multiple agents and we have *replicas* of this data. Replicas 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.

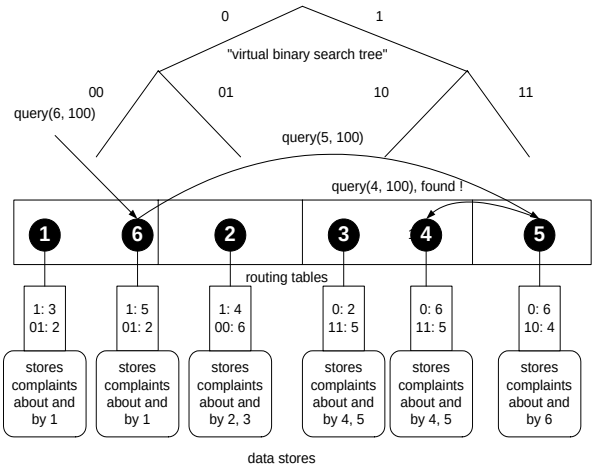

Figure 2: Example P-Grid

One can see that the access method is organized in a peer-to-peer fashion, 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.
- $query(a, k) : v$, where a is an arbitrary agent in the network, which returns the data values v for a corresponding query k.

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

- There exists an efficient decentralized bootstrap algorithm, based on bilateral interactions of agents, which creates the access structure without central control. This algorithm is randomized and requires minimal global knowledge by the agents, namely an agreement on the key space. The mutual communications basically lead to a stepwise 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 a mechanism for computing trust using P-Grid as storage structure for complaints. Every agent p can file a complaint about q at any time. It stores the complaint by sending messages

$$insert(a_1, key(p), c(p,q)) \text{ and } insert(a_2, key(q), c(p,q))$$

to arbitrary agents a_1 and a_2. 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.

After an agent has identified the agents that store the complaints it faces a problem. The agents providing the data could themselves be malicious. This can be dealt with by checking in a next step the trustworthiness of the agents, that store 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, such that the number r of replicas satisfies on average $\pi_{max}^r < \epsilon$, 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. In addition, we will also limit the depth of the exploration of trustworthiness of agents to limit the search space, 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 that the network has unrestricted connectivity (like the Internet), their trustworthiness does not play a role with respect to obtaining correct results. Either agents 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.

5.1 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

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

where w is the number of different witnesses found, a_i is the identifier of the i-th witness, f_i is the frequency with which witness a_i is found and $s = \sum_{i=1}^{w} f_i$. $cr_i(q)$ and $cf_i(q)$ are the number of complaints, that according to witness a_i agent q has received respectively filed. Different frequencies f_i indicate that not all witnesses are found with the 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. The following function 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 $cf_i^{norm}(q)$. The factor $1 - (\frac{s-f_i}{s})^s$ corresponds to the probability of not finding witness i in s attempts. If f_i is high this probability is low and vice versa, which leads to the desired compensation effect. We ignore 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. It is 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 -1 mistrust.

$$decide_p(cr_i^{norm}(q), cf_i^{norm}(q)) =$$
$$\textbf{if}$$
$$cr_i^{norm}(q) cf_i^{norm}(q) \leq (\frac{1}{2} + \frac{4}{\sqrt{cr_p^{avg} cf_p^{avg}}})^2 cr_p^{avg} cf_p^{avg}$$
$$\textbf{then } 1 \textbf{ else } -1$$

This criterion is a heuristics. It is based on the argument that, if an observed value for complaints exceeds the general average of the trust measure too much, the agent must be dishonest. The problem is the determination of the factor, by which it may exceed the average value at most.

To determine this factor we performed a probabilistic analysis, which we can describe here only informally because of space limitations. We model agent interactions as Poisson processes. In each interaction cheating occurs with a certain probability, depending on the cheating probability of the participating agents. From that we derive upper and lower Chernoff bounds for the deviation of the total number of complaints on an agent from the expected number of complaints. Comparing the lower bound for a malicious agent with the upper bound for an honest agent we can determine the optimal value for distinguishing the two types of agents considering their total number of complaints. This results in an equation which cannot be solved in closed form. Therefore we determine an approximate solution function, which has been used in the decision function $decide_p$.

5.2 Exploring Trust

By obtaining the number of complaints in the way de-

scribed 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 of witnesses

   The result of the algorithm can be
    1 = trustworthy
    0 = no assessment possible
   -1 = not trustworthy */

ExploreTrustSimple(p, q)

  W = GetComplaints(q);
  /* using the P-Grid */
  /* returns a number of witnesses and the
  normalized number of complaints they report */

  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 strategy does not include the checking of the witnesses. Therefore we consider also a more sophisticated procedure in the following.

```
ExploreTrustComplex(p, q, l)

IF l<=0 RETURN 0 ELSE

  W = GetComplaints(q);
  update statistics with W;

  IF |W| = 0 RETURN 0;

  IF |W| = 1
     t = ExploreTrustComplex(p, a_1, 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, a_i, l-1) < 1
             W := W \ {entry i of W};
        /* eliminate 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 underlying assumption for 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 value of 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 understand better what "few" means. In addition, we compare the simple and complex algorithm for trust assessment.

6.1 Simulation Setting

We have implemented the algorithm using 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 generated. 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.

The experiment proceeds as follows.

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

2. We perform 6400 and 12800 random interactions, i.e., each agent has on average 100 respectively 200 interactions with another agent. During that phase complaints are stored in the P-Grid whenever two agents meet and at least one of them 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 trustworthiness 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 assessment. For example, checking only honest agents in the beginning would make the assessment more sensitive for subsequent malicious agents, whereas if only malicious agents are assessed 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. In that case

they return random numbers, when being asked as a witness.

The simulations return 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 considered as cheaters. This corresponds to a cautious 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 of the quality of the algorithm for each parameter setting. For the cheater population with constant cheating probability of $\frac{1}{4}$ the results are as follows.

replicas	interactions	decision alg.	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 many cheaters with very low cheating probability become difficult to distinguish from the honest agents.

replicas	interactions	decision alg.	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 in case of optimal parameter settings for the different malicious agent populations. We denote c_{cheat} as the number of cheaters correctly identified, u_{cheat} as the number of cheaters for which no decision is made, w_{cheat} as the number of cheaters which are wrongly assessed, and analogously c_{hon}, u_{hon} and w_{hon} for the honest agents. For malicious agents with constant cheating probability we get.

cheaters	c_{cheat}	u_{cheat}	w_{cheat}	c_{hon}	u_{hon}	w_{hon}
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 malicious agents with variable cheating probability we get.

cheaters	c_{cheat}	u_{cheat}	w_{cheat}	c_{hon}	u_{hon}	w_{hon}
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 cheaters 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 the ability to resolve many more of the undecided cases. For the case of variable cheating probability the method is quite robust up to 20 agents. For larger numbers of agents an increasing fraction of agents that cheat rarely will be considered as honest.

One might remark 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 acquired. In fact, we made the simulations with a relatively low number of agents in order to keep the computational cost of the simulation low. We ran 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 also only 200 encounters. The results are of the same quality with a quality factor of 0.02825. This indicates that there exists no (substantial) dependency in 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. They show that a detection of cheating agents is in fact possible in a pure peer-to-peer environment based on a fully decentralized method.

7. CONCLUSIONS

To summarize, 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.

Having introduced our approach, allows us to provide a more detailed comparison to related approaches. If we compare our approach with that from [8], the approach closest to ours, we can identify the following major qualitative differences:

First, in our approach data about the observed behavior about all other agents and from all other agents is made directly available for evaluating trustworthiness using the local trust computation. This allows to compute directly the expected outcome respectively risk involved in an interaction with an agent, and makes the level of trust directly dependent on the expected utility of the interaction. This also easily allows to adapt the computation of the trust values to changing requirements or for exploiting additional context information. In contrast, in [8] an abstract measure for trust is made available. Only trust values, but no observation data, is made available to other agents by propagation along referral chains. The trust values are qualitative in nature and satisfy due to their construction certain desireable properties, like symmetry, transitivity or self-reinforcment. They allow to locate cheaters in the limiting case, but no detailed, quantitative risk assessments.

Second, more importantly, the approach in [8] has difficulties to scale for large numbers of agents, similarly as other, related approaches. Every agent has to be able to provide referrals on the trustworthiness of any other agent in order to make the approach working. Although the authors remain unclear with respect to the implementation, either this requires the storage of trust information for all other agents at each agent, or the propagation of requests for referral through the whole network (or some combination of the two). In the first case the approach cannot scale in storage space for large agent populations, in the second case the communication cost is high and the approach would experience similar problems by flooding the network with messages, as Gnutella does.

The ultimate litmus test is of course to directly compare the approaches in a simulation setting and to evaluate, which trust management system leads to the best performance under the same resource consumption. This is part of future work.

As our approach is a first contribution to this problem, a number of other 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.

8. 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* Proc. of the Ninth International Conference on Cooperative Information Systems (CoopIS 2001), 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] S.Marsh: *Formalising Trust as a Computational Concept* Ph.D. Thesis, University of Stirling, 1994.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker: *A Scalable Content-Addressable Network* Proc. of the ACM SIGCOMM, 2001.

[7] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan: *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications* Proc. of the ACM SIGCOMM, 2001.

[8] 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.