

# Decentralized Trust Management for Ad-Hoc Peer-to-Peer Networks

Thomas Repantis     Vana Kalogeraki  
Department of Computer Science & Engineering  
University of California, Riverside  
Riverside, CA 92521  
{trep,vana}@cs.ucr.edu

## ABSTRACT

Modern mobile devices can form ad-hoc networks to autonomously share data and services. While such self-organizing, peer-to-peer communities offer exciting collaboration opportunities, deciding whether to trust another peer can be challenging. In this work we propose a decentralized trust management middleware for ad-hoc, peer-to-peer networks, based on reputation. Our middleware's protocols take advantage of the unstructured nature of the network to render malicious behavior, such as lying and colluding, risky. The reputation information of each peer is stored in its neighbors and piggy-backed on its replies. By simulating the behavior of networks both with and without a rating scheme we were able to show that just a few dishonest peers can flood the network with false results, whereas this phenomenon is virtually eliminated when using our middleware.

## Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems

## General Terms

Algorithms, Design, Security

## Keywords

Trust, reputation, peer-to-peer networks, ad-hoc, unstructured, decentralized.

## 1. INTRODUCTION

The vision of pervasive or ubiquitous computing has been brought closer by advances in the networking, processing and storage capabilities of personal mobile devices such as laptops, cellphones, and PDAs [14]. Such devices can form ad-hoc networks to autonomously share data and services [13]. Self-organizing, peer-to-peer networks, in which nodes act as

both clients and servers, and without a central coordinator, offer exciting opportunities for dynamic and cost-effective collaboration. Users can form localized communities to participate in work-related projects, multi-player games, social networks, or auctions. However, in an unstructured and decentralized topology several security issues arise. One of the most challenging problems is to enable a peer to decide whether to trust another peer, in the absence of a central trust managing authority [2]. Trust is important when sharing data or processing power, and crucial for e-commerce applications such as auctioning.

By saying that peer A puts a level of trust into peer B, we mean that A estimates the probability of B acting in a way that will allow A to achieve a desired level of satisfaction. One way a peer A can estimate the level of trust to put into another peer B, is based on the reputation of peer B. The reputation of peer B is measured from previous interactions of peer A with peer B, and also from previous interactions of other peers with peer B.

One of the main difficulties in managing reputation-based trust in ad-hoc, peer-to-peer networks is that information about peer interactions is spread across the network, and no single peer has a complete global view of the peers' reputations. Furthermore, malicious peers might tamper with reputation information while it is stored locally or transmitted, or even try to defame other peers. A middleware solution to these challenges can facilitate secure peer interoperability without user intervention.

We have identified the following major requirements for a reputation-based trust management middleware: i) To enable peers to identify trustworthy and untrustworthy peers for the particular resource and level of trust they require. ii) To be light-weight, so that the protocol overhead for identifying the peers' trust level is not hindering their actual interaction. iii) To be resistant to collusions; preventing malicious peers from forming cliques to boost their reputation or to defame other peers. iv) To be resistant to malicious attempts to tamper with the reputation information of peers, identifying such attacks.

Distributed trust management based on reputations has been the focus of several research efforts. One of the main challenges, which is also the focus of this work, is how to decide where to store the reputation information. Storing the reputation information in a distributed manner and conducting polling to gather it, as proposed in [3], generates a large amount of network traffic and delay. Storing the reputation information in the peer this information refers to re-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MPAC '06, November 27-December 1, 2006 Melbourne, Australia  
Copyright 2006 ACM 1-59593-421-9/06/11 ...\$5.00

quires complicated operations to ensure that this peer will not tamper with its reputation [12]. On the other hand, storing the reputation information in just one peer is also risky, since that peer is controlling another peer’s fate and may try blackmailing or colluding with it. Anonymous storing of reputation information [15] is complicated and requires broadcasting, which has unacceptable overhead for an unstructured, peer-to-peer system. Storing the same reputation information in a group of peers, like [8] proposes for structured, peer-to-peer networks, seems a reasonable approach, since it will allow the comparison and verification of the reputation information received by all or some of the peers of that group.

In this paper we propose a decentralized trust management middleware for unstructured, ad-hoc, peer-to-peer networks, that is based on reputation. To achieve the aforementioned goals and overcome the limitations of the existing schemes, we propose storing the reputation information in a group of peers that is not easily identifiable, so that collusions and blackmailing are hindered. Thus, in our middleware the reputation information of each peer is stored in its neighbors and piggy-backed on its replies to requests for data or services. The novelty of our middleware lies in the fact that it relies on the lack of network structure to manage reputation information in a secure way. The lack of structure and the dynamic nature of the network are usually regarded as obstacles in managing trust information in self-organizing networks. Our approach utilizes these characteristics to build a self-organizing, non-intrusive trust management infrastructure resistant to tampering and collusions. Our experiments show that a few dishonest peers can seriously threaten the operation of an ad-hoc, peer-to-peer network, and that our trust management middleware can effectively prevent this from happening.

The rest of the paper progresses as follows: We discuss our system model in section 2, and our system’s operation in section 3. Section 4 describes how representative attacks are prevented, while section 5 elaborates on system algorithms. Section 6 presents our experimental evaluation. Related work is discussed in section 7, and section 8 concludes the paper and summarizes our contributions.

## 2. SYSTEM MODEL

We assume a logical network of peers that provide data or services to each other. We will use the term object to collectively refer to both data and services. Each peer  $p_i$  is identified by a public/private key pair, and maintains connections to other peers. The network is unstructured, decentralized and self-organizing, meaning that peers make their own decisions as to which peers to connect to or to query for objects. Each peer that offers an object  $o_j$  to another peer receives a rating  $r_j$ . A peer’s reputation  $R_i$  is defined as the sum of ratings it has received so far,  $R_i = \sum_j r_j$ . When a peer acts as a consumer of an object, its goal is to identify the peer with the highest reputation, that is offering the particular object. Knowing the peer’s reputation the consumer can decide whether to trust the provider, depending on the minimum trust level  $L_j$  it requires for this particular type of object. For example, a peer might have different required trust levels for different types of transactions, depending on their cost. While the minimum trust levels are set once by the user, discovering the providers with the highest reputation  $R_i$  and comparing  $R_i$  to  $L_j$  to decide whether a peer can

be trusted (if  $R_i \geq L_j$ ) are the responsibilities of the trust management middleware. When a peer acts as a provider of an object, its goal is to have as many consumers as possible. This can result in monetary gain, privileges as a consumer, or any other benefit defined by an incentives policy, such as [17]. Since the way for a provider to attract more consumers is to have a reputation higher than their trust level, every provider’s goal is to have as high a reputation as possible. Honest peers try to boost their reputation by offering objects as promised, to receive good ratings. Malicious peers try to either increase their reputation by tampering with it, without actually having received the corresponding ratings, or decrease the reputation of other peers, to increase their own chances of being selected as providers.

## 3. SYSTEM OPERATION

A peer  $p_s$  searches for an object by sending query messages to its immediate neighbors. These queries are evaluated locally in each peer and in case a peer  $p_r$  offers a matching object, the positive reply (query-hit) is returned to  $p_s$ . The queries are propagated further, until their number of hops to travel (Time To Live –TTL) expires. Similarly to Gnutella [4], every query is identified by a globally unique identifier (GUID), which we call the *transaction GUID – TID*. The TID is the same for the query message, and the possible query-hit, and rating messages that are produced as a result of this query. It is defined as a random number, generated by the peer  $p_s$  that produced the query, together with its public key. The query-hits follow the same path as the queries to reach  $p_s$ . This is easily achieved by the peers caching the TID from a query they have routed and using the reverse route when routing the corresponding query-hit, which has the same TID.

Every immediate neighbor of  $p_r$ , through which a query-hit of this peer travels, is responsible for adding the reputation of  $p_r$  to the query-hit message. Depending on the topology of the network,  $p_r$  has several immediate neighbors and all of them are responsible for piggy-backing its reputation to its query-hit messages. The peer  $p_s$  that generated the query compares all query-hits originating from the same peer  $p_r$ , to ensure that all report the same reputation for it. The reputation  $R$  of each peer is associated at  $p_s$  with a confidence value  $C$ , which is equal to the number of peers that have reported  $R$ . Honest peers are encouraged through  $C$  to maintain multiple neighbors, which makes attacks riskier, due to the entropy introduced by the unstructured topology as we explain in the following sections. The necessity of having more than one neighbor reporting the same reputation is further explained in section 4.1, while the importance of the confidence value is explained in section 4.4.

Figure 1 shows an example of a query and query-hit exchange. Let us assume that peer A ( $p_s$ ) creates a query with  $TTL = 3$  that is propagated and eventually reaches F ( $p_r$ ), who –having the object– creates a query-hit. That reply follows the same path as the query to reach A. The neighbors of F, namely C and D add its reputation to the query-hit, before propagating it further. In this topology two query-hits will be generated, so that peer B will be able to verify that F’s reputation on both of them is the same. This redundancy is newly introduced, since normally F would have just replied once. Since F’s neighbors do not know if they will be the only ones propagating the current query-hit, they cannot risk tampering with the reputation.

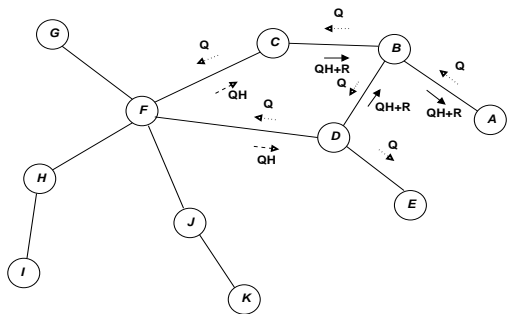


Figure 1: Query and query-hit example.

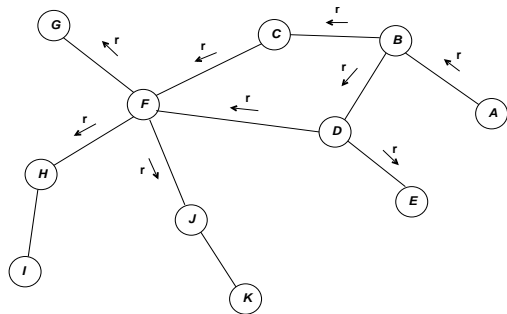


Figure 2: Rating example.

After an interaction,  $p_s$  rates the object it was provided. The rating message is signed by  $p_s$  and propagated using the same flooding-based mechanism as the query message. However, the TTL of the rating message is larger than the TTL of the query message by 1, so that the rating can reach all the neighbors of the peer  $p_r$  that is being rated.

Figure 2 shows an example of rating. After A uses the object provided by F, it creates a rating message, with  $TTL = 3 + 1 = 4$ , that is propagated and reaches all of F’s immediate neighbors (C, D, G, H, J), who update the reputation they store for F. The rating message also reaches peers like B and E, that will not store the rating of F, since they are not its neighbors and do not maintain its reputation.

Each of  $p_r$ ’s neighbors stores locally the query-hit’s TID when piggy-backing  $p_r$ ’s reputation on its query-hit. The TID contains the public key of  $p_s$ , that was contained in the query. The public key is used to verify the signed rating produced by  $p_s$  if the transaction takes place. Storing the query-hits’ TIDs enables each neighbor to keep track of the query-hits  $p_r$  has produced. Once a rating is received, it also contains the TID of the original query and query-hit. This enables the neighbors to associate the rating with the original query-hit. While a query-hit may not always result to a transaction and a corresponding rating, a rating must always contain a TID seen by the neighbor in the past, as long as the rating’s TTL has not expired. This enables the neighbors to detect collusions and is explained in section 4.4. The TIDs are periodically purged from the neighbors, but are stored for long enough time, to ensure that it will be possible for the rating to be associated with the query-hit, if the transaction actually takes place.

## 4. ATTACKS

We now present several attacks of malicious peers and show how our middleware prevents them, by storing each peer’s reputation in all of its immediate neighbors.

### 4.1 Tampering

**Alter Reputation.** A peer does not store its own reputation, thus it cannot tamper with it. A malicious peer however can change the reputation it stores for one of its neighbors. Yet, such an attempt is detected by the recipient of the query-hit (the generator of the query,  $p_s$ ), since multiple neighbors of  $p_r$ , depending on the topology, will report  $p_r$ ’s reputation and all of them should report the same value. In other words, a neighbor reporting bogus reputation might be revealed, since it may not be the only one answering, due to the unstructured topology. For example,

in figure 1, A will make sure that the reputation of F reported by both C and D is the same. The redundancy in reputation reports and the unknown topology also deter random peers from altering reputations they propagate, since the alterations might be revealed.

**Alter Ratings.** Similarly, tampering with a peer’s rating could be detected by that peer. For example, in figure 2, F could detect a change in its rating by D, by comparing it to the rating it received from C for the same query-hit. However, comparisons like this are not needed, because ratings are signed by their creator  $p_s$ . This way, the recipients of the rating, namely the neighbors of the peer  $p_r$  that offered the object, can verify that no peer on the way has altered the rating in any way. They already have  $p_s$ ’s public key, cached in the TID of the corresponding query-hit. Signing the rating messages is needed not only to prevent alterations by random peers on the way, but also to prevent alterations by the peer the rating refers to. For example, in figure 2, F is asked to propagate its own rating. Even though if a rating altered by F was stored in G, H, and J, peers C and D would still have the correct value and the discrepancy would be noted in future reputation reports, having digitally signed ratings minimizes the risk of a successful alteration.

### 4.2 Blackmailing

Peers store their neighbors’ reputation and their neighbors store theirs’. This balance of power makes blackmailing infeasible. Furthermore, if just one neighbor decides to report bogus reputation, it is running the risk of being identified as was described in section 4.1.

### 4.3 Multiple Ratings

A malicious peer can try to submit multiple positive or negative ratings for others. Such an attack would be mounted by impersonating multiple ratings coming from the same or different peers (public keys). In either case, since no query-hits with the same TIDs as the ratings are stored in the neighbors, they can detect the discrepancy. In other words, maximally one rating per TID is stored in every neighbor.

### 4.4 Collusions

**Symmetric Boost.** A collusion can take place in which two neighbors agree to boost each other’s reputation. This however would be revealed by the replies of the rest of the neighbors, for both peers. Thus, to mutually boost each other’s reputation, all neighbors of each peer would have to cooperate and consequently all of their neighbors, until the whole network was part of the collusion. For example, in figure 1, if F and C decided to boost each other’s reputation,

D, G, H, and J (for F), as well as B (for C) would have to cooperate too, and consequently also I (for H), K (for J), E (for D), and A (for B).

**Incomplete Asymmetric Boost.** An attack that would seem more feasible would be for a malicious peer to bribe some of its neighbors to boost its reputation, without however the attacker boosting their reputation in return. Obviously, the attacker would not propagate query-hits through the neighbors that do not store the boosted reputation, otherwise the discrepancy in the reporting values would be noted in the recipient of the query-hits. This attack however is detected by the neighbors that are not part of the collusion, due to the fact that they are comparing the TIDs of the ratings they receive, with the query-hit TIDs they keep stored. In more detail, they would receive ratings with TIDs they have not propagated, and those ratings would have a TTL that has not expired. For example, in figure 2 if D had been excluded from propagating a query-hit of F, it would detect the discrepancy when receiving the corresponding rating. On the contrary, G, H, and J will not be alarmed since the rating reaches them but its TTL expires. This means that they are the TTL+1 hop of the rating. Therefore the query (that traveled through TTL hops) did not reach them and they were not supposed to propagate a corresponding query-hit. B and E are not alarmed either, since they received a rating for a peer that is not one of their neighbors and they can safely ignore it.

**Complete Asymmetric Boost.** An even more elaborate collusion involves a malicious peer bribing *all* of its neighbors to boost its reputation, without the attacker boosting their reputation in return. Since all peers participating in the protocol are now malicious the attack cannot be detected. This attack however reveals the use of the confidence value  $C$ . The higher the number of peers that report a reputation value  $R$ , the higher the number of peers the attacker would have to bribe. Thus, an attacker maintaining just a small number of bribed neighbors will only gain a reputation with a small confidence.

The lack of structure is usually regarded as a major hindrance in managing trust information in unstructured, peer-to-peer systems. Our approach is novel, in that it utilizes exactly this characteristic to create an environment that makes tampering with reputation information cumbersome and risky. Higher reliability –at a higher message overhead– can be achieved by storing the reputation of each peer in neighbors more than one hop away.

## 5. SYSTEM ALGORITHMS

Even though the focus of this work is on how to decide where to store the reputation information, the algorithms for selecting and for rating the provider of an object are also of interest, as are peer dynamics. The algorithm for selecting the provider with the best reputation might weigh the ratings, according to the personal opinion of the peer for the raters, or according to the raters’ reputation as peers or even as raters. In addition to the reputation of a provider  $R_i$  and the minimum trust level for a particular object acceptable by the consumer,  $L_j$ , we have introduced one more factor in the peer selection, namely the confidence  $C_i$  in the reported  $R_i$  for each provider. Currently we let the consumer set minimum confidence levels  $K_j$  per object, once. Consequently we let the middleware select the provider with maximum  $R_i$ , as long as  $R_i \geq L_j$  and  $C_i \geq K_j$ . For providers with

equal  $R_i$ ’s other criteria, such as the network distance, can be utilized to break the tie.

The algorithm for rating a provider might use a scale that allows comparison with the current rating average for a provider. In that way, ratings far away from the average might be noted, and the responsibility of the rater might also be rated. Moreover, both the object provider and consumer may rate each other. Currently we do not associate the reputation of a peer as a provider with any reputation it may have as a rater. Therefore, to rate providers we use a simple binary rating scheme, to denote dissatisfaction (-1) or satisfaction (+1) with an object. This scheme allows the ratings given by different peers to be as comparable as possible, as it leaves little room for subjective interpretation. Furthermore, it enables ratings to be assigned automatically, since success or failure in the consumption of an object are easier to be determined than user satisfaction.

Issues related to peer dynamics are also interesting. We briefly discuss peer reconnection here and leave as future work a thorough investigation of highly dynamic networks, such as those formed by mobile peers. When a peer enters the system it receives the reputation of the peers it connects to from their neighbors. Its new neighbors receive any existing reputation it has by its old neighbors which still store it together with its public key<sup>1</sup>. Not all of the old neighbors need to be online at the same time for the reputation to be retrieved. These two types of updates are achieved following protocols similar to the exchange of query and query-hit messages described earlier. If a peer is new in the network, its reputation is zero and it is built as the peer engages in transactions. By not giving any initial reputation to newcomers, we discourage peers with bad reputation to leave the system and reenter under a new identity, since building a reputation is a tedious process. Furthermore, this particular kind of attack, also known as the sybil attack, in which a malicious peer assumes multiple identities, has been the study of recent research [19]. Our middleware, which provides the infrastructure to store reputation information pertaining to an identity, can be combined with a solution such as [19] to prevent peers from joining under a new identity.

## 6. EXPERIMENTAL EVALUATION

We conducted an evaluation of our middleware in networks of thousands of peers, implementing it on top of the Gnutella [4] unstructured, peer-to-peer network, using the NeuroGrid simulator [7]. We used 3000 types of objects, distributed uniformly (30 objects per peer) across randomly connected peers. In each experiment we ran 100 random searches and averaged our results from 5 measurements.

Apart from the honest peers, that provide the objects they claim they have, we included a number of dishonest peers in the network. These malicious peers claim that they have every object they are asked for, in other words reply with a bogus query-hit to every query they receive, without of course being able to provide the real requested object. We observed the effect of that behavior on the operation of the network, with and without using a rating scheme. When the rating scheme is used, we assume that the malicious peers can only cheat once, since then they are discovered

<sup>1</sup>They will delete it once the peer reconnects to another place, since its new neighbors will be responsible for storing that information from now on.

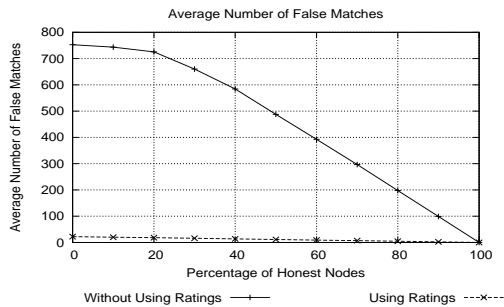


Figure 3: False matches to a search, for varying percentage of honest peers.

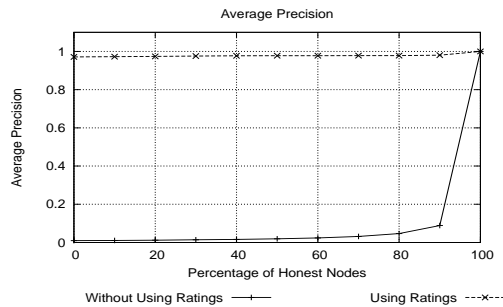


Figure 4: Proportion of genuine matches, for varying percentage of honest peers.

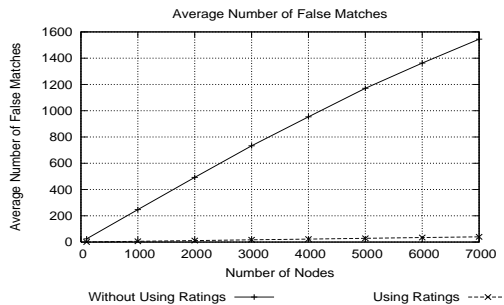


Figure 5: False matches to a search, for varying total number of peers.

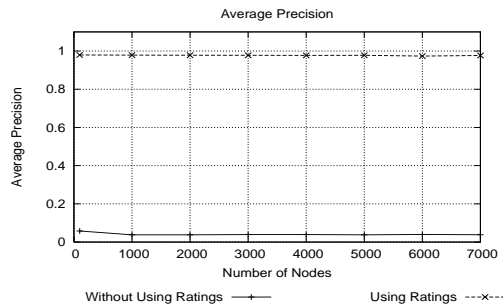


Figure 6: Proportion of genuine matches, for varying total number of peers.

and receive a bad rating that discourages other peers to interact with them.

**Variable percentage of dishonest peers.** For the first experiment, we kept the total number of peers to 1000, and we varied the number of honest peers in the network. Our goal was to determine to what extent the percentage of dishonest peers affects the operation of the network.

Figure 3 shows the average number of false matches, i.e., bogus query-hits. Without utilizing the rating scheme, this is quite high, even for relatively small percentages of dishonest peers. By using the rating scheme the number of false matches is virtually eliminated, even for networks with many malicious peers.

Figure 4 shows the average precision, i.e., the average proportion of query-hits that were genuine (not bogus). When using the rating scheme, that proportion remains very high. Surprising are the results when not using any rating scheme. The precision remains practically close to zero, even when 80% of the peers are honest. If 1 out of 10 peers is dishonest, 9 out of 10 query-hits are bogus. This means that a few dishonest peers have the ability to flood the network with false matches, representing a real threat to its operation.

**Variable number of peers.** It was interesting to see if dishonest behavior is equally threatening for larger-scale networks. Therefore in the second experiment we kept the percentage of honest peers to 75%, and we varied the total number of peers in the network.

Figure 5 shows the average number of bogus query-hits. Without using the rating scheme, the number of false matches grows very fast for large networks. Again, the rating scheme prevents this behavior.

Figure 6 shows the average proportion of genuine query-

hits. Again by using the rating scheme this proportion remains high, even for large networks. However, without a rating scheme, the dishonest peers present a threat even to large-scale networks. Even though 3 out of 4 peers are honest, the percentage of genuine query-hits remains close to zero. We observe that the dishonest peers are able to flood even large networks.

## 7. RELATED WORK

Several peer-to-peer reputation systems have already been proposed, taking different approaches as to where to store the reputation information. In RCertPX [12] a reputation certificate is stored in the peer that it refers to and is updated after each transaction. To avoid tampering, the last rater always digitally signs the whole certificate. Thus, the last rater needs to be online for another peer to be able to verify the certificate's correctness. Another complication arises from the fact that a rater and a ratee could collude to change all the ratings of the ratee. In P2PRep [3] a polling-based protocol is proposed and implemented. Any peer that wants to query the trust value of another peer, broadcasts a query to the network, collects the replies, and individually contacts the voters for confirmation. Apart from the network traffic generated and the delay of the process, this approach counts only the reviews of present peers that can be reached. A similar approach of voting, but on the reputation of objects instead of that of peers, is implemented in Credence [18], while [16] focuses on identifying feedback that does not correspond to actual transactions.

TrustMe [15] identifies anonymity as an important feature of trust-managing systems. The trust rating of each peer is placed at another random peer, which replies to all queries

for the trust values it holds. One drawback of the protocol is that it relies on broadcasting, making it unacceptable for large-scale, unstructured networks. EigenTrust [8], a global variable regarding a peer's reputation is stored in a peer's mother peers. The global variable is generated by aggregating local variables in all peers, in an iterative process. The algorithm does not prevent mother peers from blackmailing a peer, nor from colluding against a peer.

In NICE [9] cooperating peers form a graph, and a peer providing a service is responsible to prove its reliability to a peer that would like to use it, by finding a path in the graph to that peer. However during this discovery process flooding is used and many irrelevant peers may be contacted. Moreover, since the peer providing a service is gathering its reputation information, it may omit bad ratings. In [1] a trust managing system on top of the P-Grid peer-to-peer system is described. Complaints about peers are stored in a virtual binary search tree. However no measures are taken against peers storing complaints about themselves, or against malicious peers, which might tamper with ratings while they transmit them. Similar to EigenTrust, P-Grid requires a network structure to be maintained, for the reputation information to be stored and retrieved.

Challenges related to the mobility of nodes have also been the focus of recent research efforts, identifying the mobility patterns of nodes [11], their location [5], time, context [10], and current environmental conditions [6] as other important factors related to trust.

## 8. CONCLUSIONS

We have proposed a decentralized trust management middleware based on reputation, for ad-hoc, peer-to-peer networks. We have shown how random topologies that may be created make malicious behavior like lying and colluding risky. Moreover, all peers are equally powerful, controlling the fates of their neighbors, while their fates are controlled by their neighbors. The unstructured nature of ad-hoc, peer-to-peer networks is usually regarded as an obstacle in ensuring trust. Our middleware relies on exactly this characteristic to achieve this goal and to avoid relying on a central authority. We have tried to keep our middleware's protocol simple and easy to build on top of infrastructures already available for the exchange of messages, to minimize its overhead. The communication overhead of polling-based protocols is avoided and the only extra messages introduced are those carrying a new rating. Furthermore, the ratings of peers that have left the system are still present. Our future work includes investigating the effects of mobility and elaborating on the peer selection and rating algorithms.

## 9. ACKNOWLEDGMENTS

We wish to thank China V. Ravishankar and Dimitrios Gunopulos for their helpful advice during the initial phase of this work.

This research has been supported by NSF Awards 0330481 and 0627191.

## 10. REFERENCES

- [1] K. Aberer and Z. Despotovic. Managing trust in a peer-to-peer information system. In *International Conference on Information and Knowledge Management, CIKM*, 2001.
- [2] V. Cahill et al. Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 2:52–61, 2003.
- [3] F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servers in a P2P network. In *International World Wide Web Conference, WWW*, 2002.
- [4] Gnutella Protocol Development. <http://rfc-gnutella.sourceforge.net/>, 2003.
- [5] T. Horozov, N. Narasimhan, and V. Vasudevan. Using location for personalized POI recommendations in mobile environments. In *International Symposium on Applications on Internet, SAINT*, 2006.
- [6] M. Huebscher and J. McCann. A learning model for trustworthiness of context-awareness services. In *2nd International Workshop on Pervasive Computing and Communication Security, PerSec*, 2005.
- [7] S. Joseph. An extendible open source P2P simulator. *P2P Journal*, pages 1–15, November 2003.
- [8] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *International World Wide Web Conference, WWW*, 2003.
- [9] S. Lee, R. Sherwood, and B. Bhattacharjee. Cooperative peer groups in NICE. In *IEEE INFOCOM*, 2003.
- [10] J. Liu and V. Issarny. Enhanced reputation mechanism for mobile ad hoc networks. In *International Conference on Trust Management, iTrust*, 2004.
- [11] L. McNamara, C. Mascolo, and L. Capra. Trust and mobility aware service provision for pervasive computing. In *1st International Workshop on Requirements and Solutions for Pervasive Software Infrastructures, RSPSI*, 2006.
- [12] B. Ooi, C. Liau, and K. Tau. Managing trust in peer-to-peer systems using reputation-based techniques. In *International Conference on Web Age Information Management, WAIM*, 2003.
- [13] T. Repantis and V. Kalogeraki. Data dissemination in mobile peer-to-peer networks. In *International Conference on Mobile Data Management, MDM*, 2005.
- [14] M. Roman et al. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1:74–83, 2002.
- [15] A. Singh and L. Liu. TrustMe: Anonymous management of trust relationships in decentralized P2P systems. In *International Conference on Peer-to-Peer Computing, P2P*, 2003.
- [16] M. Srivatsa, L. Xiong, and L. Liu. TrustGuard: Countering vulnerabilities in reputation management for decentralized overlay networks. In *International World Wide Web Conference, WWW*, 2005.
- [17] Q. Sun and H. Garcia-Molina. SLIC: A selfish link-based incentive mechanism for unstructured peer-to-peer networks. In *International Conference on Distributed Computing Systems, ICDCS*, 2004.
- [18] K. Walsh and E. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *Networked Systems Design and Implementation, NSDI*, 2006.
- [19] H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman. SybilGuard: Defending against sybil attacks via social networks. In *ACM SIGCOMM*, 2006.