TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems

Aameek Singh
College of Computing, Georgia Tech
aameek@cc.gatech.edu

Ling Liu
College of Computing, Georgia Tech
lingliu@cc.gatech.edu

Abstract

Decentralized Peer to Peer (P2P) networks offer both opportunities and threats. Its open and decentralized nature makes it extremely susceptible to malicious users spreading harmful content like viruses, trojans or, even just wasting valuable resources of the network. In order to minimize such threats, the use of community-based reputations as trust measurements is fast becoming a de-facto standard. The idea is to dynamically assign each peer a trust rating based on its performance in the network and store it at a suitable place. Any peer wishing to interact with another peer can make an informed decision based on such a rating.

An important challenge in managing such trust relationships is to design a protocol to secure the placement and access of these trust ratings. Surprisingly, all the related work in this area either support very limited anonymity or assume anonymity to be an undesired feature and neglect it.

In this paper, we motivate the importance of anonymity, **especially** *in such trust based systems. We then present TrustMe – a secure and anonymous underlying protocol for trust management. The protocol provides mutual anonymity for both the trust host and the trust querying peer. Through a series of simulation-based experiments, we show that the TrustMe protocol is extremely secure in the face of a variety of possible attacks and present a thorough analysis of the protocol.*

1. Introduction

Peer-to-Peer overlay networks are increasingly gaining acceptance on the Internet as they provide an infrastructure in which the desired information and products can be located and traded. While P2P systems based on central indices have run into legal problems, the decentalized systems have continued to flourish. Gnutella, Kazaa, Freenet [3] are extremely popular amongst the WWW community with millions of users worldwide. One of the most attractive features of a typical P2P resource-sharing application is the anonymity that it provides to the requester and the provider of a resource.

However, the open nature of the P2P networks also makes the system vulnerable to malicious users trying to infect the network. Genuine looking files may actually contain viruses, which can potentially destroy data and infect programs on a peer's hard drive. Though the use of anti-virus software can detect such incidents, there is hardly any mechanism to prevent this threat or to punish such malicious users. Also, there is no accountability of a peer for providing a particular resource. Such drawbacks are some of the primary hurdles in the promotion of P2P systems to a more useful setting.

Assuming a pure P2P environment, strategies for tackling such issues have to be decentralized in nature, i.e. we need to develop mechanisms which avoid presence of central trusted authorities. Many existing P2P communities have used community-based reputations to estimate the trustworthiness and predict the future behavior of peers. Concretely, the current systems associate with each peer, a *trustworthiness* metric and allow other peers to have access to this information and decide by themselves whether to interact with that peer or not. Such a metric is primarily based on *peer-reviews* i.e. each peer on interacting with another peer, rates the performance of the peer on a common scale and vice versa. This trustworthiness metric is usually called reputation-based **trust value**. A high trust value indicates that the peer has gained good reputation in terms of its past performance and thus is more trustworthy, whereas a low trust value means the peer has relatively poor quality of service (QoS) in the past and are rated with low reputations by other peers in the community.

A typical transaction in trust-enabled P2P resource sharing networks will be as follows - A peer will query for a particular resource using the normal P2P protocol queries. It then will receive a number of offers from various peers within the network, who are willing to provide that resource (provider peers). Then the requestor peer will query for the trust values of the provider peers and select the best peer to provide the service. After interacting with the chosen provider peer, it will then rate the provider peer based on its performance and vice versa. Two important issues are involved in this process: (1) what trust metrics are effective for computing the reputation-based trust? And (2) how to distribute, store, and access the trust values of peers securely? The first issue is related to trust models used for building trust among peers. The second issue is related to secure storage and secure access of trust values against possible misuses and abuses by malicious peers. A fair amount of work has been done in the area of computing reputation-based trust ratings [2, 8, 15]. However, the area of developing secure underlying protocols to distribute and access the trust ratings in the overlay network has been, relatively unexplored.

In this paper we present TrustMe – an anonymous and

secure protocol for maintaining and accessing trust rating information. TrustMe uses Public Key cryptography schemes to provide security and is resistant to various attacks. We argue that the distribution and access of trust ratings should be distinguished from the routine functions such as file inquiry and downloading performed in a P2P system. In this paper, we attempt to answer questions like:

- *Where should the trust value of a peer be stored?* Clearly, no peer should hold its own trust value (in which case, every peer would be the most trustworthy!). Also, because of the decentralized nature, there is no central place where it can be stored.
- *How to securely access other peers' trust values?* P2P protocols follow message forwarding mechanisms, where a message reaches the desired peer after going through a number of other peers in the network. Hence we cannot send un-encrypted messages, since anybody can fake critical information.

A unique characteristic of the TrustMe protocol is its support for mutual anonymity in managing peers' trust relationships. Not only the peers who access trust ratings of other peers remain anonymous but also peers who store trust ratings of other peers are protected from targeted attacks by keeping their identity hidden. In addition, the TrustMe design ensures the following properties:

1. **Security:** Due to decentralized management of trust relationships, the trust rating of a peer is stored at other peers in the network and it is extremely important to protect these trust hosting peers from targetted attacks.
2. **Reliability:** It is important to ensure that anybody querying for a trust value gets the *true* trust value inspite of presence of various malicious users.
3. **Accountability:** In a peer-review based trust systems, it is important that peers are accountable for the feedback they provide about other peers. Any malicious peer trying to manipulate trust ratings should be identifiable.

In the remaining sections, we first briefly overview the related work. Then we describe the proposed trust management protocol, TrustMe. We report initial results of our simulation-based experiments, showing the effectiveness and the cost of the TrustMe protocol.

2. Related Work

Most of the security threats presented in Peer-to-Peer information sharing environments are due to two main features of the peer-to-peer design: *anonymous peer-to-peer communication* (for example, Gnutella servants (peers) are anonymous and are only identified by a self-identified servant id) and *variety of the shared information* (e.g., the files authorized to be shared in Gnutella can include all media types, including executable and binary files). The former feature involves a weakness due to the combination of low accountability and low trust of the individual peers. Under the shield of anonymity, malicious peers can answer to virtually any query providing tampered-with information. The two features combined make the P2P environments more vulnerable to certain security attacks compared to the centralized systems.

While some adhoc mechanisms have been applied in the current systems (blacklisting hosts, checking file signatures, virus scans, etc.), there is a greater need for more dependable security framework and protocols. Amongst the e-commerce community, the notion of reputation-based trust metrics has received a great deal of attention. While a formal treatment appeared in [10], a number of other approaches like [1, 16], have been found to be relatively more useful. The most successful usage of trust in an industrial setting is the reputation system of eBay, where each user rates other users and shares the information with other participants. However, none of the aforementioned are well suited to a distributed and decentralized environment existing in pure P2P systems. Whereas [10, 1, 16] suffer from the inherent complexity involved in maintaining the trust models, the eBay model relies on a centralized server to store and present the knowledge.

Within the P2P community, recently several efforts have been engaged in mechanisms for building reputation-based trust models [2, 15, 8]. However, there is scarce amount of research in developing secure means of accessing and maintaining such models in decentralized P2P systems. In the absence of such mechanisms, there are huge threats possible, which can render the trust models effectively useless by preventing users gaining access to the correct trust ratings. Meanwhile, the security research in the P2P environment has focussed on the issues of identity [7], and Distributed Hash Tables (DHT) [12].

To the best of our knowledge, only [4] has attempted to present a secure protocol at message level in addition to a trust model. Their protocol is based on a polling based mechanism and use public key cryptography to provide various security features. In contrast, the TrustMe design argues that, to provide secure, reliable, and accountable distribution and access of trust ratings of peers, it is not only important to authenticate the P2P messages but also critical to ensure requestor anonymity and provider anonymity for distributed management of trust relationships in dcentralized P2P systems.

3. TrustMe: A Protocol for Anonymous Trust Management

3.1. Anonymity: Why is it essential ?

One of the major factors in the rise of the P2P systems is the anonymity they provide. It has been felt to be extremely essential to design mechanisms that provide privacy and escape censorship. There have been various attempts to develop such systems even outside the P2P realm with Crowds [5] and commercial available anonymizers as examples. Also within the scope of the P2P, the anonymous services provided by Freenet [3] and FreeHaven [6] have received attention. Another line of work [14] uses trusted third parties, established at super-peer nodes to provide mutual anonymity to

both initiatior and responder for hybrid P2P systems.

Unfortunately from a security point of view, anonymity has been regarded as a rogue element. Most of the trust based P2P systems [4, 8] have sacrificed anonymity in order to provide secure underlying protocols. This is a huge threat, with the potential to disrupt the whole functioning of the system. If a malicious peer can identify the peers who are reporting its poor trust values, it can launch targetted attacks preventing them from doing so. Such attacks can range from spam, threatening emails at best, to a complete DoS attack at worst. And yet these are only the possible *electronic* confrontation means. Such possibilities demotivate peers from reviewing other peers and letting their reviews be publicly available. This lack of anonymity has been expressed as the possible cause of eBay users having very little negative ratings.

On the other end of the spectrum, a peer would like to maintain anonymity while querying for another peer's trust value. It could very well be an employee looking for ratings of available career counsellors or a corporation seeking new suppliers without letting their current supplier know about it.

Clearly an anonymous protocol for such trust based systems will go a long way in uplifting the P2P computing to more mission-critical applications.

3.2. Protocol Design Considerations

There are two popular approaches to derive trust ratings of peers based on their reputations: *Transaction-based* rating and *user-based* rating. In the transaction-based approach, peers provide feedbacks upon completion of each transaction, and the trust rating of a peer is computed based on all transactions it has performed with other peers. In the user-based approach a peer gives a reputation rating to another peer based on all its experiences with that peer, instead of a per-transaction basis. The trust rating of the peer is an aggregation of all ratings obtained from the rest of the peers.

Most of the current work, including TrustMe, use the user-based approach because of its greater robustness against malicious groups acting together. A representative work is presented by Cornelli et al [4], in which any peer, say Peer A, who wants to query for the trust value of another peer, say Peer B, broadcasts a query to the network. Then the peers who have interacted with Peer B and would like to express their opinions reply back with their ⟨IP, Port⟩ tuple, encrypted with public key of Peer A. After receiving the replies, Peer A will individually contact the voters and ask them to confirm their votes to filter out incorrect fake messages. There are a number of drawbacks of this approach:

- **No persistence**: The trust metrics are not persistent. All the peers who have interacted with Peer B, but are not present in network cannot have their reviews counted. This can be potentially exploited by a collective of malicious peers who are always present in the network sending the same high/low value for Peer B, thus masking the opinions of various other peers who would have genuine ratings. As our experiments indicate (ref Section-4), even a small number of malicious peers can totally dominate the ratings of a targetted peer.
- **No anonymity**: The peers expressing opinions lose their anonymity. A peer can potentially query for its own trust value (if protected against this, ask a friend to query), and identify the peers who are giving poor trust values for it. Now these peers can be selectively targetted with other attacks like DoS. This is equivalent to voters not having a right to secret ballot.
- **Tedious decision-making**: The decision making process becomes extremely lengthy and tedious. Peer A has to contact all the voters and confirm their votes, thus increasing the time taken to make a decision on whether it wants to trust Peer B or not. Also Peer A has to combine all the valid votes before arriving at a decision.

Ideally, the system should be such that voters have *secret ballot*, votes stay in the system even when the voters have logged out and the decision making process is fast. TrustMe is designed with all these qualities in mind. We place the trust rating of each peer at a random Peer X which replies to all queries for the trust values it holds. A peer can anonymously issue a query and get the *true* value without needing to know the identity of Peer X. Also a single reply message from Peer X is enough to make a decision.

Eigenrep [8] presented an alternative approach. They propose a simplistic underlying protocol based on a DHT based mechanism like CAN [11] or Chord [13]. Each peer has a set of mother peers, which hold the trust values of that peer. The mother peers are decided based on hashing the ID of the peer (Different hashes are used to obtain a number of mother peers). If Peer A wants to query for the trust value of Peer B, it just hashes its ID to obtain the various mother peers and then queries them for the trust values. Then, it decides by taking the majority of those values. This kind of approach also suffers from a number of shortcomings:

- **Insecure communication**: After hashing a peer's ID to obtain various mother peers, the communication between the mother peer and the querying peer is not secured. This is vulnerable to a host of threats like Man-in-the-middle/Bucket-brigade [9] attacks.
- **DHT Threats**: By relying on a DHT based design, the system automatically becomes prone to numerous other possible attacks, like those discussed in [12]. Malicious routing-information tampering, malicious lookup replies and a host of other possible scenarios come into the picture. Even a small percentage of malicious nodes can have devastating consequences. The security solutions to some of these attacks are extremely complex and are not compatible with the current available systems.
- **No anonymity**: Also in such a scheme the identity of mother peers is exposed. And as already mentioned, a malicious node can attack those mother peers to prevent them from sending the *true* trust values.
- **Group threats**: Another problem is that since user-chosen IDs are used to hash, after significant monitoring, a malicious group of nodes (or a single node simulating a group of nodes) can select good combinations of

IDs to have a favorable scenario, for example, in which trust values of a particular node in the group are most likely to be hosted by other group members.

To handle such problems, TrustMe uses a random assignment of Trust-Holding Agent peers (henceforth called THA peers) and uses smart Public Key mechanisms to prevent any loss of anonymity. Also it ensures all communication to be secure.

The following notations are used:
THA-peer stands for a peer which holds the trust value for a particular peer. A *private key* is denoted by the alphabet P and a *public key* by the alphabet B. A *special private key* is denoted by SP and a *special public key* by SB (ref. Section-3.3.1). The *trust value* is denoted by the abbreviation TV. The *time stamp* (time at a particular instant) is denoted by TS. The symbol '|' stands for concatenation i.e. $X|Y$ denotes X followed by Y. Encryption of a message M by a particular key K is given by $K(M)$. The Bootstrap server (entry point for peers in a network) is denoted by BS. ID stands for the identifier of a peer in the network. It can be user-chosen or an IP address. BID stands for a special peer identifier assigned by the BS (ref. Section-3.3.1). A peer offering some resources is called an *offering peer* and the peer querying for trust values of offering peers is called the *querying peer*. A query for the trust value of a particular peer is called the *trust query*. Also the words *node* and *peer* will be used interchangebly.

3.3. Protocol

TrustMe broadly functions in the following manner. Each peer is equipped with a couple of public-private key pairs. The trust values of a peer (say Peer B) are randomly assigned to another peer (THA peer) in the network. This assignment is done by the bootstrap server in a way that the trust holding responsibilities are equally distributed amongst the participating peers. This assignment is unknown to all peers including Peer B. All the communication with the THA peer is carried out using a special key which indicates its knowledge of the trust value of Peer B. Any peer (say Peer A) interested in querying for the trust value of Peer B can broadcast a trust query for Peer B. The THA peer replies with the trust value along with some other information. Depending upon the trust value, Peer A can decide to interact with Peer B or not. Also after an interaction, Peer A can securely file a report (after giving adequate proof of the interataction) for Peer B, indicating Peer A's new trust value for Peer B. Then, the THA peer can modify the trust rating of Peer B. To provide security, reliability and accountability, TrustMe uses smart public key cryptography mechanisms.

3.3.1 Details

Firstly, the bootstrap server has a pair of keys $\langle P_{BS},\ B_{BS}\rangle$. The B_{BS} key is known to all peers in the network. This can be easily achieved by distributing it at the time of a peer join. In case of multiple bootstrap servers, either they can share a single pair or all the keys can be distributed during the join. This pair of keys is primary used as a certification mechanism certifying the validity of a node (i.e. it joined the network in a correct manner) and thus helps in pseudo-identification. Also TrustMe requires each Peer i to possess a couple of private-public pair of keys before it joins the network ($\langle P_i,\ B_i\rangle$ and $\langle P_i',\ B_i'\rangle$). The former is used for providing/receiving services whereas the latter is used while serving as the THA peer for other peers in the network.

When a Peer i joins the network, the bootstrap server generates another private-public pair, called the *Special-Private* and *Special-Public* – $\langle SP_i,\ SB_i\rangle$. They are special in the sense that even Peer i does not have the knowlege of SP_i. The bootstrap server assigns a set of peers to be the THA peers for Peer i and only those peers know SP_i. It is used as an *authentication* mechanism (receiving a message encrypted with SP_i indicates that it is coming from a THA peer) and as *secure* transmission mechanism (encrypting a message with SB_i means that only the THA peer can read it). Overall, the following information about Peer i is securely transmitted to the THA peers – $\langle ID_i,\ B_i,\ SP_i,\ SB_i\rangle$. The exact procedure in which this is achieved is explained later in Section-3.3.3. For the moment, assume that a set of peers are made the THA peers for Peer i (i.e. they possess the abovementioned information about Peer i). Also the bootstrap server assigns an identifier to the peer i denoted by BID_i. This identifier is assigned when Peer i joins the network and is given by:

$$BID_i = P_{BS}(\text{“}Valid\ Node\text{”}|B_i')$$

Notice that any node reading BID_i can be assured that it is a valid node (since nobody can fake P_{BS}). However, it does not compromise the identity of Peer i. B_i' is used to prevent the loss of anonymity in a particular attack explained later. There are four phases in the entire protocol:

Query: Any peer, say Peer j, intending to query for the trust value of Peer i can just broadcast the trust query message containing ID_i. Typically, a peer will simultaneously query for a number of peers (who offered their services for a particular resource). In such a case the trust query would be the concatenated IDs of all such peers.

$$Q(j, \{i_1, i_2, ..., i_n\}) = ID_{i_1}|ID_{i_2}|...|ID_{i_n}$$

Note that because of the message forwarding mechanism of the P2P system, the querying peer cannot be identified by looking at the query message. This provides complete privacy to the querying peer.

Reply: On receiving a trust query for Peer i, **its** THA peer, say Peer x can generate a reply and forward it back to the network. The goal of this message is to ensure that the querying peer can identify it to be generated by a THA peer and that it has not been tampered with, enroute. The reply message looks like:

$$R(x,i) = ID_i|B_i|SB_i|SP_i(TV|TS|BID_x|P_x'(TS))$$

There are a number of points to note:

- The ID_i field indicates that the reply message contains the trust value for Peer i. The SB_i key is provided to decrypt the encrypted part and the B_i key is used later to communicate with Peer i.

- Any peer in the network can read the trust value for Peer i. This is not undesirable, since anyway, any peer can query for that trust value specifically and know the value.
- The use of encryption with SP_i makes sure that the reply is coming from a THA peer. This prevents any peer from just randomly sending a value.
- The use of BID_x, apart from ensuring that a valid node is replying, ensures accountability. Though no peer can identify the THA peer, the malicious THA peers can be blacklisted by their BID_x values.
- The use of TS ensures that a message is not replayed at a later time. Also it provides a caching opportunity, in which peers can cache the value and can use the same trust value within a particular time window.
- The use of $P'_x(TS)$ and the presence of B'_x in BID_x ensures that no peer can use another peer's BID_x (which can be extracted from its earlier replies) and send replies trying to pose as that peer.
- Note that, there is still a possibility of any peer to generate a random combination of $\langle SP_i, SB_i \rangle$ and use that as a fake reply message. We will explain how this can be prevented later in Section-3.3.2.

Collecting Proof-of-Interaction: Whenever two peers (Peer i and j) interact, they exchange $P_i(TS|B_j|ID_j)$'s with each other, i.e. Peer i gets $P_j(TS|B_i|ID_i)$ from Peer j and Peer j gets $P_i(TS|B_j|ID_j)$. This value is used as a proof of an interaction. Note that no peer can generate such a value in a fake manner, since it does not know the other peer's private key. TS is used to prevent replaying of such a message. The use of ID_j and B_j is for added protection against somebody using a message from Peer i's interaction with some other peer. Another important use of the interaction message is that if a group of co-operating peers are attempting to boost each other's rating, they will need to exchange such messages every time (unless they compromise on each others' private keys as well), thus making them pay for every malicious attempt.

Report: After having interacted with a Peer i, Peer j can file a report indicating Peer j's new trust value, V, for Peer i, by broadcasting the *Report* message. For this, we need to ensure that only the THA peer can read the message and that only a peer which has *actually* interacted with Peer i can send the report. The Report message is of the form:

$$ID_i|SB_i(\text{“}Report\text{”}|V|B_j|P_j(P_i(TS|B_j|ID_j)))$$

The important features of the Report message are:

- Only the THA peers can read the message (since they are the only ones possessing SP_i). This serves as a *secret ballot* mechanism.
- Note that the THA peer will need the ID of the reporting peer to update the global trust value since the global value is based on the individual trust values of all peers. This ID can be obtained by decrypting with B_j (contained in the message) and B_i (already known; it is the THA peer for Peer i).
- The interaction-exchange message is further encrypted with P_j and B_j is sent to decrypt it. This is done to prevent any peer from sending a fake report, in an unlikely scenario of it getting hold of the $P_i(TS|B_j|ID_j)$ value from another interaction. The tallying of B_j outside and the B_j in the interaction-exchange message ensures that the right peer is sending the report.

3.3.2 Discussion

Now let us look at various possible attack scenarios and how TrustMe prevents it:

- **Manipulating Reply Messages:** This can be attempted by either a malicious THA peer or a non-THA peer.
  THA Peer: A THA peer can send a wrong trust value in the reply. To prevent this, the bootstrap server assigns a number of THA peers for a single peer. Then the querying peer can take a majority vote amongst them and select that value. This ensures reliability. Also TrustMe presents a possibility to punish such a malicious peer. If we broadcase all the reply messages, the THA peers will be able to see what other THA peers are replying with. And later on, they can include in their reply messages – the BID_x's of THA peers which are sending wrong values. A querying peer can easily identify such peers by looking at what the majority of peers are advising against. Also, since the assignment of THA peers is random, there is extremely small possibility of majority of peers being malicious and co-operative, thus making the above mechanism secure.
  There is another possibility of a malicious THA peer to get a BID_x of another THA peer (it can see the broadcasted reply) and send a wrong trust value using that BID_x. However, as mentioned earlier, this is easily prevented by the use of B' in BID_x and the $P'(TS)$ in the reply message.
  Non-THA Peer: It can either attempt to replay a genuine reply message at a later time or try and use a fake pair of $\langle SP_i, SB_i \rangle$ keys. The former scenario is prevented by including TS in the reply message. Any message older than a user-decided window is disregarded. The latter scenario can be also easily prevented by the following mechanism. The offering peers (the peers which offered resources and are being queried for their trust values) send their B_i and SB_i keys as a part of the offer, which preceeds the Query phase. That can be tallied with the value in the reply. Though this can be used to prevent anybody from tampering *another* peer's trust value (no peer can generate the correct SP_i for another peer), it is still possible for an offering peer itself to generate the fake reply message (it can send an incorrect SB_i with the offer and send fake reply accordingly). This also can be prevented, since other THA peers would send the same value of correct SB_i. So on receiving any conflicting reply messages, a peer just rejects the offer from that peer and potentially blacklists it.
- **Manipulating Proof-of-Interaction Messages:** A proof-of-interaction message can be manipulated either

by attempting to replay an old message or by using a fake pair of $\langle P_j, B_j \rangle$ keys. The replay is avoided by the use of TS. Any report message, containing the proof-of-interaction message outside a reasonable time frame, is discarded. The other possibility is that a peer uses a fake pair of $\langle P_j, B_j \rangle$ keys. Note that it is not possible for an offering peer to fake, since its THA peer would have included its true B_j value in the reply message. To prevent the querying peer from faking, the offering peer can also get its actual public key from its THA peer.

- **Manipulating Report Messages:** There is very little that can be done to manipulate a report message. As mentioned earlier, no peer can fake a report message and also that it is secure in the sense that only the THA peer can read the message. Only possibility is that a peer can rate another peer in a wrong manner, e.g., giving a poor rating inspite of good performance. This is tackled on the trust model level. Any good trust model will not effect a peer's trust rating because of a single peer.
- **Attempting to Identify THA peers:** In case only one private-public key pair is used, that is, $\langle P, B \rangle$ is used wherever $\langle P', B' \rangle$ is used, there exists a possibility of identifying the THA peers. Some peer can monitor the network for a long time and on reading various reply messages can construct a mapping of ID_i's with their corresponding B_i's. Then on reading BID_i's of THA peers, it can identify it. The use of a second pair prevents this from happening. The $\langle P', B' \rangle$ pair is only used while acting as a THA peer.

Till now we have demostrated how the protocol achieves **complete anonymity**, **security**, **reliability** and **accountability**. In the following, we summarize a few other important aspects of the TrustMe protocol:

- **Persistence:** Any peer just has to file a report to make sure its experiences are accounted for in the trust value of the interacting peer. After that, even if that peer logs out of the system, its review is counted. This way, we provide persistence to votes and provide a stronger trust mechanism. As our experiments indicate (Sec-4), non-persistent systems can provide highly misleading trust values in the presence of even a small number of malicious peers.
- **No Central Trusted Authority:** It is important to notice that the bootstrap server does not act as a CTA. It is rather a form of a certification authority. The use of P_{BS} is only as a pseudo-identification mechanism, which peers can use to mark other peers. It does not signify, in any manner, a peer being trusted or not. All the trust mechanisms are within the network and the bootstrap server does not participate in it.
- **Small decision time:** Notice that only a reply messages is enough for a peer to make a decision on whether to interact with a particular peer or not. This is extremely convenient and fast.
- **Ease of contribution:** It is extremely easy for a peer to contribute its trust value for another peer – by sending a *single* report message. This is a highly convenient as opposed to notifying each THA peer as in Eigenrep [8].

3.3.3 Peer Join

In this section, we will explain how a peer join is handled. Whenever Peer i contacts a bootstrap server for joining the network, the bootstrap server will collect its B_i and B_i' values. It also generates a new pair of private and public keys - $\langle SP_i, SB_i \rangle$. It gives SB_i to Peer i. Then, from a cache of available peers in the network, the bootstrap server will select a set of peers that will serve as the THA peer for Peer i. For the THA peer, say Peer x, it then prepares a Trust-Host-message:

$$TH_{BS}(i) = BID_x | P_{BS}(BID_x | B_x'(ID_i | B_i | SP_i | SB_i))$$

This message can be given to Peer i to be broadcasted to the network and the THA peer on receiving such a message updates its local database. Note that no peer can generate such a message (because it doesnt know P_{BS}) and also only Peer x can read the entire information, because only it knows P_x'. The use of BID_x both outside and inside is for efficiency purposes. The one on the outside prevents every peer to do a decryption by B_{BS} to check if the message is for itself and the one on the inside ensures that a fake message is not used (by prefixing a "friendly" BID_x to the rest of the message).

Note that one attack is still possible in this scenario. The Peer i does not broadcast the messages and generates a fake $\langle SP_i, SB_i \rangle$ pair and sends it to friendly peers, which will later reply with excellent trust ratings for Peer i. This can be prevented in two ways:

1. The broadcast can be done by the bootstrap server.
2. Any peer interacting with Peer i, before interacting, can ask for this message (or a shortened version like $P_{BS}(BID_x | ID_i)$) and tally the BID_x with what it will be getting from other THA peers.

3.3.4 Peer Leave

There are essentially two main things to be taken care of whenever Peer i leaves the system:

1. *Data about peers for which Peer i is the THA peer:* Before exiting, Peer i contacts the bootstrap server and gets a new peer which will take its place. It broadcasts a relevant TH_{BS} message, thus assigning the responsibility to a new peer. Note that a more convenient buffer mechanism can also be implemented by assigning M THA peers for a peer, whereas using information from only K at every step ($M > K$). Then such a step will need to take place only after the number of THA peers goes below K.
2. *Data at THA peers for Peer i:* To invalidate such information, every THA peer maintains a time stamp whenever a peer's records are accessed and if the information is not accessed for a long enough time, it just deletes it from its database.

4. Analysis

First, we look at the importance of persistence in a typical trust based decentralized P2P system. As mentioned earlier [4] presented a non-persistent system in which peers make decisions based on locally available trust values. Locally available trust values are obtained by combining individual ratings from peers (which have interacted with the desired peer) currently present in the network. Hence, it does not take into account the ratings from peers which interacted with the desired peer and later left the system. Global trust values are those which take into account the trust ratings of all the peers which ever interacted with the desired peer (as in TrustMe based systems), thus providing a true account. We simulated that scenario with a randomly generated network of 500 peers and periodically removed 5% of the peers. Figure-1 presents the mean squared errors of local trust values for three kinds of peer behaviour - (a) Inconsistent - in which peers rated every other peer randomly between 0 and 1; (b) Consistent - in which "good" peers always get rated between $[0.5, 1)$ and "poor" peers always between $[0, 0.5)$; (c) Malicious - in which there were 10 malicious peers always rating the good peers as poor and poor peers as good. Each transaction selects a random querying peer and a random queried peer and comprises of the complete four phases (from querying to reporting) of the protocol.

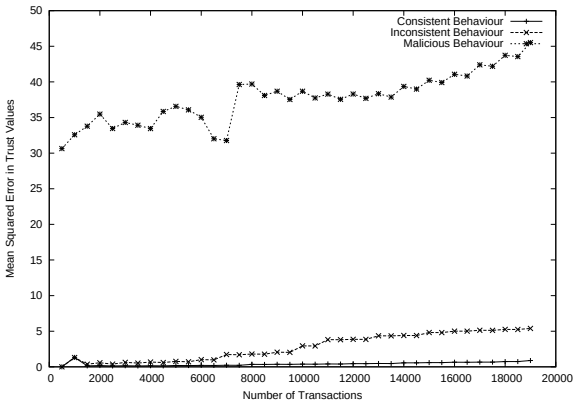

Figure 1: Mean Squared Errors for Non-Persistent Systems

The Consistent behaviour scenario has the least MSE, since in this case the local values will always be a close approximation to the global values. The Inconsistent behaviour scenario is a little worse since a peer might lose the peers who are rating it as good (they might have left the system) and all the current peers might be rating it as bad or vice versa. The Malicious behaviour scenario is the worst since a few malicious peers deliberately try to reverse the ratings of the peers and tend to mislead the querying peer the most.

Figure-2 shows the effects of a targetted attack by 10 malicious peers on a particular peer. They target a high trust value peer by rating it as bad everytime somebody queries for its trust value. As a result the systems based on locally available values rates the peer between $0.3 - 0.4$, whereas its actual trust value lies between $0.7 - 0.8$. This proves that having as little as 10 malicious peers, the trust value in a non-persistent system can be highly misleading.

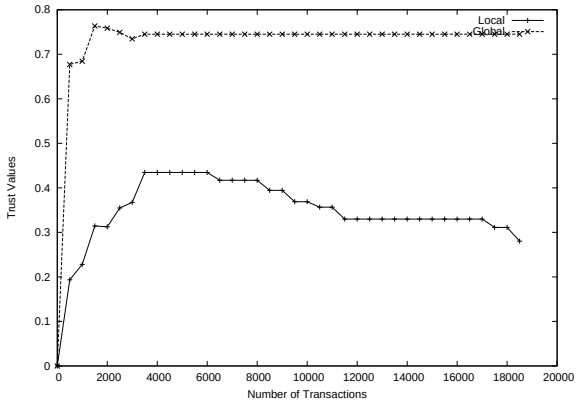

Figure 2: Targetted attack on a single peer

Next we look at the costs of various protocols. There are two components to the costs. The first is the cost of using cryptography primitives. This is of the same order for all the currently available systems since all are based on public key cryptography. The second component is the messaging costs because of the increased number of messages due to different protocols. Figure-3 presents the messaging costs of polling and TrustMe based systems. The TrustMe based systems are twice as expensive as the polling based systems. The primary component of the cost in both the mechanisms is the cost of broadcasting. In the polling mechanism, only the query is broadcasted, whereas in TrustMe the query and the report are both broadcasted. The graph shows that the number of messages increase linearly with the number of trasactions for both non-persistent and persistent trust based systems. The messaging costs vary little with the number of THA peers. This is because the cost of broadcasting does not change significantly with the addition of new THA peers.

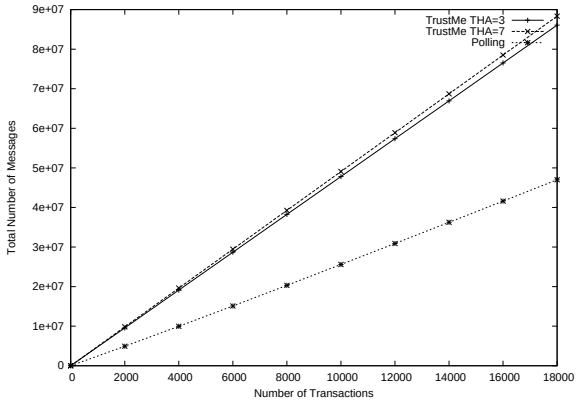

Figure 3: Messaging Costs

Next, we compare the response times of these systems. We assume a response to be complete when the querying peer receives the first message that provides it with enough information to take a decision. Note that, it might still be expecting other messages in order to strictly verify the authenticity of the first message. We make this choice, since we can always choose a peer to accept services from, at the receipt of such a message and abort the process in case this message is proved to be in-authentic. The cumulative response times is the total response time for all the transactions and is

measured in time units where each unit is equal to the time taken to exchange one message from a peer to its neighbour (assuming it to be the same for all). We used 3 THA peers.

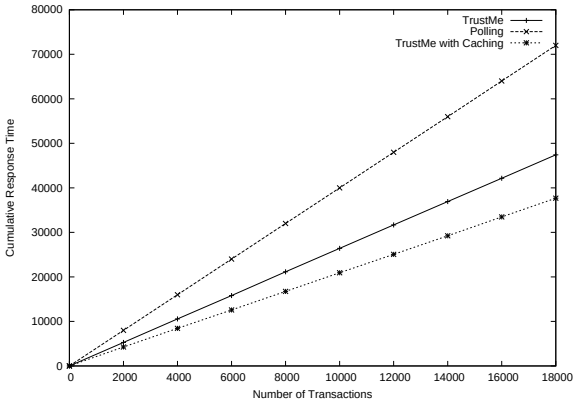

Figure 4: Cumulative Response Times

As Figure-4 shows, TrustMe based systems have the least cumulative response time. The reason for this is the fact that receiving a single reply from a THA peer is enough for the local peer to make a decision. On the other hand, the polling based mechanism is more expensive, since it has to collect the replies from all peers and then combine them. Caching trust values also significantly reduces the response times of the querying peers. Every peer while forwarding a query reply, caches that reply locally and responds to the queries with that cached value. Hence, the trust values will be available closer than normal. Here the term "close" is an overlay network property and not the geographical proximity. Figure-5 shows the variation of the cumulative response times with the number of THA peers. As the number of THA peers increases, there is a greater chance of a THA peer being close to the querying peer and hence reducing the response times.

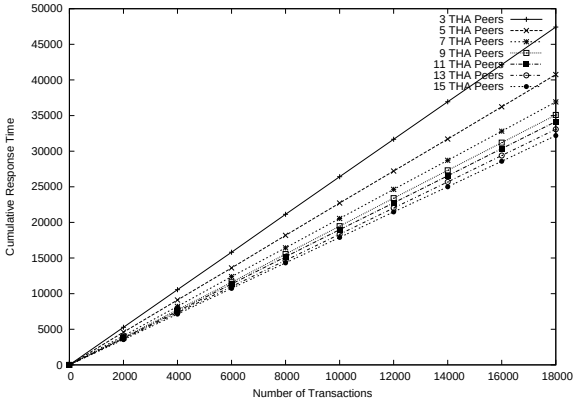

Figure 5: Cumulative Response Times

5. Conclusions and Future Work

We have described the design of TrustMe – a secure and anonymous underlying protocol for trust management. The protocol provides mutual anonymity for both the trust host and the trust querying peer, aiming at providing secure, reliable, and accountable distribution and access of ratings of peers. We have also presented a thorough security analysis of the protocol and reported some initial experimental results, showing that the TrustMe protocol has desirable features of anonymity, reliability, accountability and is secure in the presence of a variety of possible attacks.

One of our ongoing work on TrustMe is to explore various other cryptographic primitives which can be more efficient as compared to public key cryptography. The use of broadcast authentication mechanisms like TESLA which are based on symmetric key cryptography could be worth exploring. Also, we intend to study mechanisms when some sort of a central authority is available.

References

[1] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *HICSS*, 2000.

[2] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *CIKM*, 2001.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2001.

[4] F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a p2p network. In *Eleventh International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[5] Crowds. http://www.research.att.com/projects/crowds/.

[6] R. Dingledine, M. J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[7] J. Douceur. The sybil attack. In *IPTPS02 Workshop, Cambridge, MA (USA)*, March 2002.

[8] S. Kamvar, M. Schlosser, and H. Garcia-Molina. Eigenrep: Reputation management in p2p networks. In *Twelvth International World Wide Web Conference*, 2003.

[9] C. Kaufman, R. Perlman, and M. Speciner. *Network security: private communication in a public world*. Prentice-Hall, Inc., 1995.

[10] S. Marsh. Formalising trust as a computational concept. In *Ph.D. Thesis, University of Stirling*, 1994.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[12] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *IPTPS02 Workshop, Cambridge, MA (USA)*, March 2002.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001.

[14] L. Xiao, Z. Xu, and X. Zhang. Mutual anonymity protocols for hybrid p2p systems. In *23rd International Conference on Distributed Computing Systems*, Providence, Rhode Island, May 2003.

[15] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer ecommerce communities. In *IEEE International Conference on Electronic Commerce*, 2003.

[16] B. Yu and M. P. Singh. A social mechanism of reputation management in electronic communities. In *Cooperative Information Agents*, 2000.