Managing Trust in Peer-to-Peer Systems Using Reputation-Based Techniques

Beng Chin Ooi Chu Yee Liau Kian-Lee Tan

Department of Computer Science
National University of Singapore
3 Scinece Drive 2, Singapore 117543

Abstract. In this paper, we examine the issue of managing trust in peer-to-peer systems. In particular, we focus on reputation-based schemes. We look at some design considerations in implementing distributed reputation-based systems, namely storage, integrity, metrics and changing of identity. We provide a survey of related work on the storage and integrity issues, and present our solution to address these issues.

1 Introduction

The social impact of reputation on an individual or group is long known. Research has shown that reputation plays a vital role in the decision of initiating an interaction and the pricing of services. For example, [1] has shown that the rating in eBay's [2] feedback system does encourage transactions and in some occasions making the item sold by a highly rated seller to be higher in price.

In electronic marketplaces, the reputation that a user has is the result of aggregating all the impressions of the other users that interacted with the user in the past. A reputation system is an effective way to facilitate the trust in a P2P system. It collects and aggregates the feedback of participants' past behaviors, which is known as reputation, and publishes the reputations so that everyone can view it freely. The reputation informs the participant about other's ability and disposition, and helps the participant to decide who to trust. Furthermore, reputation system also encourages participant to be more trustworthy and discourages those who are not from participating.

Existing reputation systems are those implemented in online store and auction site, such as eBay [2] and Amazon [3]. In eBay, after buying a collectible in an auction, the buyer can go back to the site and rate the seller for prompt shipping and whether the physical item actually matched the description in the auction. The rating given by the buyer is recorded into the seller's reputation by the website. When the subsequent buyer wishes to make a purchase from the seller, he can refer to seller's reputation before he makes any decision. If the reputation shows that the previous buyers were mostly well treated, then the seller is honest and worth dealing with. With reputation schemes in place sellers are highly motivated to offer the best possible service to every single buyer.

Existing work on peer-to-peer applications focuses on routing and discovery [4, 5], data exchange [6] and caching [7]. Trust has gained lesser attention despite

its importance. In this paper, we will examine the issue of managing trust in peer-to-peer systems that are based on reputation. We will look at some design considerations in implementing distributed reputation-based systems, namely storage, integrity, metrics and changing of identity. We provide a survey of related work on the storage and integrity issues, and present our initial effort to address these issues.

The rest of this paper is organized as follows. In the next section, we discuss the design considerations for distributed reputation-based systems. Section 3 surveys existing reputation-based systems in terms of their storage and integrity issues. In Section 4, we present our solution to these two issues. Finally, we conclude in Section 5.

2 Design Considerations

Although peer-to-peer systems have been extensively studied in the past few years, the research on peer-to-peer reputation has been relatively small in number. Here, we provides an overview on the study of various peer-to-peer reputation systems. To begin with, we start with discussion of the design considerations of reputation system for peer-to-peer.

1. Storage of the reputation information. The reputation has to be stored in a distributed manner, but with high availability, especially in P2P systems where peers can appear offline from time to time. Additionally, the reputation should be retrieved efficiently since it is used frequently.
2. Integrity of the reputation information. The integrity of the reputation information will dictate the usefulness of a reputation system. While in a centralized design, the integrity issue can be easily addressed, it is much more challenging in a decentralized environment.
3. Reputation metrics. Reputation metrics provide the representation of a user's reputation. The complexity of the calculation of reputation metric will undermine the performance of the whole system.
4. Changing of identity. In a peer-to-peer system, due to its decentralized nature, changing of identity is extremely easy and usually zero-cost. This is slightly different as compared to real-world where shifting of identity is usually more complicated, which often involve government or authority. A good reputation system should prevent any incentive of changing identity.

3 Survey of Existing Peer-to-Peer Repuation-based Systems

This section briefly reviews some of the existing P2P reputation systems, focusing particularly on the storage and integrity issues. We start by giving an overview of the reputation systems.

3.1 Overview

Kevin A. Burton designed the OpenPrivacy Distributed Reputation System [8] on P2P, which is derived from the distributed trust model. It proposed the concept of reputation network, which is composed by identities (representing nodes) and evaluation certificates (representing edges). Therefore, the trustworthiness of the identities can be estimated from a visible sub-graph of the reputation network.

P2PREP [9] is a reputation sharing protocol proposed for *Gnutella*, where each peer keeps track and shares with others the reputation of their peers. Reputation sharing is based on a distributed polling protocol. Service requesters can assess the reliability by polling peers.

Karl Aberer and Zoran Despotovic [10] proposed a trust managing system on the P2P system P-Grid [11] (Managing trust). It integrates the trust management and data management schemes to build a full-fledged P2P architecture for information systems. The reputations in this system are expressed as complaints; the more complaints a peer gets, the less trustworthy it could be. This system assumes peers in the network to be honest normally. After each transaction, and only if there is dissatisfaction, a peer will file a complaint about the unhappy experience. To evaluate the reputation of a peer involves searching for complaints about the peer.

Dietrich Fahrenholtz and Winfried Lamersdof [12] introduced a distributed reputation management system (RMS). In RMS, reputation information is kept by its owner, and public key cryptography is used to solve the integrity and non-repudiation issues. During each transaction, a portal acts as a trusted third party to resolve the possible disputation during the reputation update.

Kamvar et. al [13] proposed a reputation management system, EigenRep, for P2P file sharing systems such as Gnutella to combat the spread of inauthentic file. In their system, each peer is given a global reputation that reflects the experiences of other peers with it.

3.2 Storage of Reputation Information

OpenPrivacy In OpenPrivacy, the reputation information is stored in a certificate. The system is similar in concept to *web of trust* [14]. A peer certifies another peer through the use of certificate. Every certificate stores the value of the target's reputation and the confidence of the certificate creator. To prevent tampering, each certificate is digitally signed with the private key of the certificate creator. These certificates are stored at the certificate creator as well as the certification target.

P2PRep In P2PRep, every peer in the system stores their interaction experience with other peers (based on pseudonym). This reputation records are being updated every time an interaction takes place. These reputation records can be used by other peers to make decision when initializing an interaction. In this case, before a peer consumes a service, the peer polls other peers about their

knowledge of the service provider. At the end of the interaction, the service consumer updates the reputation of the provider and at the same time updates the credibility of the peers that addressed opinion on the provider.

Managing Trust Managing Trust stores the complaints about a peers in the P-Grid [11]. The underlying idea of the P-Grid approach is to create a virtual binary search structure with replication that is distributed over the peers and supports efficient search. The construction and the search/update operations can be performed without any central control or global knowledge.

RMS RMS also stores the reputation information in a certificate. However, RMS is different from OpenPrivacy in the implementation of the reputation certificate. In RMS, there exists a trusted third party to record the transaction history for the subscribers. The transaction history that the trusted party stored is used by others to check the correctness of the certificate presented by a peer.

EigenRep In EigenRep, two types of value, local and global, are being stored in the systems. The local value is stored in every peer and the global value, which is derived from multiple local values, are being handled by random peers in a distributed hash table (DHT) such as CAN [15] or Chord [4].

Discussion All of the aforementioned reputation systems use decentralized storage for storing the reputation information. This is very important as centralized storage for reputation information will limit the scalability of the P2P reputation system in the long term and affect the performance for retrieving the reputation information.

Efficient retrieval of reputation information minimizes the communication overhead. For instance, to retrieve reputation for a peer in *RMS* or *OpenPrivacy*, we need to issue only a query message to the peer since the certificate stores all the reputation information of the peer. In *EigenRep* and *Managing Trust*, the cost of retrieving reputation information is proportional to retrieving information from DHT system and P-Grid respectively. However, the cost for *P2PRep* to retrieve reputation information is proportional to the O(N) for the network with N peers.

3.3 Integrity of Reputation Information

OpenPrivacy Integrity of the reputation information stored in *OpenPrivacy* is preserved through the use of cryptography means. Every certificate is digitally signed by the private key of the certificate creator. A peer needs the public key of the certificate creator in order to verify the validity of the certificate and the information stored within. If the content of the certificate is tampered, the verification of the certificate will fail.

P2PRep The integrity of the reputation information is also being protected with cryptography means. Unlike *OpenPrivacy*, the reputation is only being encrypted and signed for the purpose of transmission. Since the reputation information is being stored at the rating peer and not the target peer, there is very minimal risk that the target peer is able to change the reputation information. However, the risk do exist when the reputation information traveled from the sender to the requestor. Therefore the protocol defined in P2PRep provides integrity (and confidentiality when needed). Before the reputation information is transferred, it is being signed with the private key of the sender so that the information will be intact while being transmitted.

Managing Trust In *Managing Trust*, the integrity of the complaints depend on the behavior of peers in the network. In order to overcome this problem, the system assumes the probability of the peers in the P-Grid storage system that are malicious is π. This value cannot be greater than a certain maximum, π_{max}. Its storage infrastructure is configured in such a way that r replicas must satisfy the condition $\pi^r_{max} < \varepsilon$, where π^r_{max} is the average probability of r replicas and ε the acceptable tolerance.

RMS In *RMS*, the integrity is preserved through the signature of the trusted third party. The trusted third party could be implemented as a centralized server or multiple servers. If it is implemented across multiple servers, there must be trust between the servers.

EigenRep The integrity of the reputation information in *EigenRep* also depends on the trustworthiness of the peers that calculate and store the global reputation value. However, the system reduces the possibility of malicious acts through random selection of peers that calculate the global values and redundancy in global value.

Discussion It seems that one of the most challenging issues of decentralized reputation management system is the integrity of the reputation information. On one hand, cryptography techniques that preserved the integrity of the reputation information seems effective, it suffers from the overhead of verification. The number of public keys needed to verify the reputation depend on the number of certificates to be verified and for a large number of certificates, the cost of retrieving the public keys can be very high. On the other hand, the integrity of information on systems such as *Managing Trust* and *EigenRep* depends on the storage infrastructure.

4 Our Solution

We propose a P2P reputation scheme that aims at providing efficient retrieval of reputation information and providing integrity of the information. In our scheme,

the reputation is maintained by the owner. This greatly simplifies the problem of storing reputation information. In addition, the retrieval of the reputation information can be done efficiently without any additional communication cost. By having the owner to store the reputation information, there is the risk of information integrity. To protect the integrity of the reputation, we have introduced the notion of reputation certificate we termed *RCert*. At the same time we have proposed protocols to facilitate the update of the reputation information.

4.1 Components

Public Key Infrastructure (PKI) PKI [16] is employed to provide security properties which include confidentiality, integrity, authentication and non-repudiation. All these are achieved through the use of symmetric and asymmetric cryptography as well as digital signatures. We have omitted the confidentiality requirement in our proposed scheme as our goal is not to provide communication secrecy among peers.

Entities There are two entities in the system. A peer that provides services (service provider) and a peer that consumes services (service consumer). In P2P system, a peer can act as a service provider as well as service consumer. This is because in P2P there is no true distinction between server and client. Entities in the network has a pair of public and private keys that represent its identity. At the same time, the pair of keys is used in the digital signature process. We assume there exists a mechanism that allows a peer to be located and contacted given its identity. This can be achieved through the use of P2P systems such as [4], that provide efficient lookup mechanisms.

Roles There are two different roles a peer plays. After a peer has finished consuming a service provided by a peer, it takes up the role of a *rater*. The peer that provides the services will be termed *ratee*. The *rater* is responsible for evaluating the *ratee* based on the experience of the interaction with *ratee*. We shall defer the protocol used in the rating process to section 4.2.

Reputation Certificates (*RCert*) *RCert* consists of two components: header and *RCertUnit*. The information is updated by the service consumer each time after a transaction has taken place. Every update is appended to the end of *RCert* and is digitally signed by the *ratee* to prevent the owner from changing the information. Figure 1 depicts the format of *RCert*.

RCert header gives information about its owner, such as owner's identity and owner's public key. This information binds the *RCert* to its owner. Besides, the header also includes information about *RCert* such as *RCert*'s current ID and previous ID if this certificate is not the first created by the owner. With the ID information, this allows the owner to create a new *RCert* but still provides a pointer to previous *RCert* owned by the owner. When an *RCert* grows too

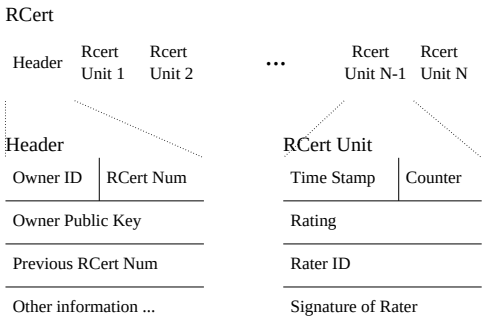

Fig. 1. Format of the *RCert*

big, the owner can create a new *RCert* and provides the reference to the old *RCert* in the header. The old *RCert* can be stored locally in the system and will only be sent to service requester which requested it. *RCertUnit* contains the following entries:

- *TimeStamp* - issued by the owner right before a transaction is started. It is digitally signed by the issuer and is used as a proof of transaction.
- *Rating* - this is the comment given by a peer that had the transaction with the owner. It records the transaction experience of the *rater* with the owner.
- *RaterID* - this is the identity of the peer that created this rating (*RCertUnit*).
- *Signature* - the signature is created by the *rater*, using its private key, on the entire *RCert* including the header for the integrity of the *RCert*.

4.2 The Protocol - *RCertPX*

The *RCertPX* protocol involves ten steps and is shown in figure 2. Assuming a peer needs certain service from other peers. It first uses resource discovery mechanism such as those mentioned in [17, 18] to locate service provider (step 1). All the peers that have the resources needed by the requesting peer send their replies together with their Reputation certificates (*RCert*) (step 2). Upon receiving the *RCert*, the requester needs to verify the validity of the *RCert* (step 3). This is done by checking the last *RCert Unit* in the *RCert* by contacting the *rater*. If the *rater* returns a *Last-TimeStamp* that has not been revoked, the *RCert* is valid (step 4). A *Last-TimeStamp* consists of three elements:

- *TimeStamp* issued by service provider
- Status of the *TimeStamp* (valid/revoked)
- *RevokedPeer* - identity of the party authorized the revoked

The *Last-TimeStamp* provides the validity of the *RCert* currently used by an *RCert* owner. In the event where the last *rater* is not available (eg. offline),

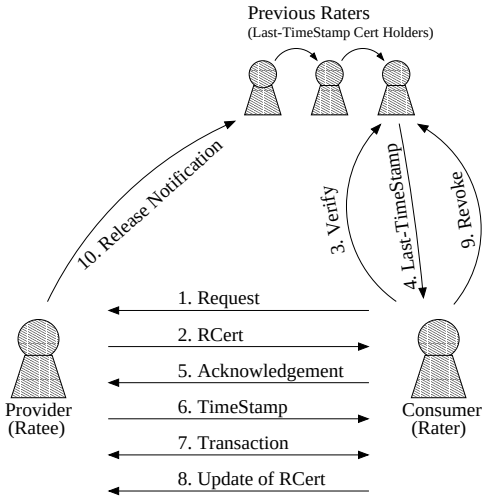

Fig. 2. *RCertP* Protocol

the requester can try to contact the preceding *raters* until there is one that is available. In this case, the verification is done by checking on the *Last-TimeStamp* in the following way. The *TimeStamp* information in the *Last-TimeStamp* should match those on the *RCertUnit* created by the *rater* and since the *Last-TimeStamp* has been revoked, the *RevokedPeer* in the *Last-TimeStamp* must match the next *rater* specified in the *RCert*. This verification mechanism provides more information about the transaction history of the *RCert*'s owner and refrains a peer from using any of its old *RCert*.

After evaluating all the *RCert*, the requesting peer makes decision on which peer to choose as service provider and sends an acknowledgement to the provider (step 5). The acknowledgment is digitally signed with the requester's private key and it shall be used as a proof of transaction request.

This is followed by the sending of TimeStamp from the provider to the requester (step 6). The TimeStamp is signed by the provider and in this protocol it contains the time value on the provider machine and the transaction counter. The requester will then verify the time and signature on the timestamp by using the public key of provider. We do not assume there exists a synchronized time between requester and provider. However, there should be a way for the requester to check the correctness of the time (e.g., the time should not be too different from the time in requester system). The counter incorporated reflects the latest information about the transaction sequence. For instance, if there have been 20 transaction so far, the counter information in the *TimeStamp* should reflect 21 as its value.

Peers then start the transaction (step 7). Upon completion of the transaction, the service requester starts to rate its service provider. The *rater* (service requester) updates the *RCert* sent to it in step 2 by adding the timestamp from

step 6, followed by the rating based on the transaction experience. The *rater* also added its ID. The *rater* completes the updates by hashing the content of the certificate and digitally signs the hash with its private key. In addition, the *rater* will perform two extra steps.

1. The *rater* needs to create and store the $Last - TimeStamp$ and make it available to others when needed.
2. If the *rater* is not the first one to rate the service provider, it needs to contact the previous *rater* to 'revoke' the piece of $Last - TimeStamp$ store.

Next, the *rater* sends the updated certificate to the *ratee* (step 8). The *rater* then issues a request to the preceding *rater* to revoke the timestamp stored there by sending the latest timestamp sent to it by the *ratee*. To verify the request, the preceding *rater* checks the timestamp.

- The time in timestamp must be more current than the one currently stored.
- The counter in the timestamp must be the next number to the one currently stored.
- The timestamp must indeed sign by the *ratee*.

Once the preceding *rater* is convinced that the timestamp sends to it is correct, it revokes the timestamp information stored locally by creating a status 'revoke' and place a digital signature on the revoked timestamp (step 9). Upon receiving the acknowledgement that the preceding *rater* has revoked the timestamp on its side, the current *rater* sends the updated reputation certificate to the *ratee*. The *ratee* should use the updated certificate for its next transaction. Finally, the provider notifies the previous *rater* that it can remove its Last-TimeStamp Certificate.

4.3 Analysis and Discussion

RCertPX provides the assurance that if an *RCert* is presented and the signature is verified to be valid, it means that the content in the *RCert* has not been changed by the owner. This is achieved through the use of digital signature on the entire *RCert*. In addition, *Last-TimeStamp* used in the protocol provides information about the validity of *RCert*. With the *Last-TimeStamp*, a requester can verify the validity of the *RCert* by contacting previous *rater*. If the *Last-TimeStamp* has not been revoked, it indicates that the *RCert* is up to date; otherwise, the *RCert* is an old one, and might not be valid. This prevents the provider from discarding the unsatisfied rating by reusing its old *RCert*.

Three parties are evolved in this protocol. They are the *ratee*, the current *rater* and one of the the previous *raters*. In the following discussion, we show that if anyone of them is malicious, the correctness of the *RCert* will not get tampered.

In the case where *ratee* turns malicious, it will be able to send a blank *RCert* to the user. Therefore, a blank *RCert* should be regarded as having very low correctness. A malicious *ratee* will not be able to reuse its old *RCert*. This is

because the *Last-TimeStamp* introduced provides the mechanism to prevent this from happening. When a *ratee* is using back the old *RCert*, during verification of the *RCert*, its act will be exposed.

On the other hand, if the current *rater* acts maliciously, it can either refuse to give a rating or give an invalid signature on the *RCert*. However, this will not cause any problem at all. When the *rater* refuse to give any rating, the *ratee* can present the acknowledgement sent by the *rater* during transaction confirmed (step 5 of *RCertPX*) that the *rater* has indeed requested for the transaction. In the event where *rater* purposefully gives an invalid signature on the *RCert*, the *ratee* can present the acknowledgement to the previous *rater* to request arbitration. Then the previous *rater* can require the current *rater* to present his update again. If the current *rater* refuse to give the update, or present an invalid one, the previous *rater* can cancel the revocation on its *Last-TimeStamp*. If the current *rater* present a valid update, the previous rater will send it to the *ratee*.

When the previous *rater* acts maliciously, it can:

1. refuse to present the *Last-TimeStamp*
2. give a revoked *Last-TimeStamp* even if it has not been revoked

For case 1, if the current *rater* cannot get the *Last-TimeStamp*, it cannot verify the validity of *RCert*. The same thing happens when the previous *rater* is off-line for the moment. This is very common in the P2P networks. Our amendment to this problem is to use a group of previous *raters* rather a single previous *rater*. Each previous *rater* keeps a count number on the *Last-TimeStamp*, whose initial value is the total number of previous *rater*s. In each revocation of *Last-TimeStamp*, the count number is reduced by 1. When the count number reaches 0, the *Last-TimeStamp* is revoked completely, and the *rater* leaves the previous *rater* group automatically. Therefore, if the number of previous *rater*s is N, the last N raters are all capable of verifying the validity of *RCert*. When the last previous *rater* refuses to present the *Last-TimeStamp*, the current *rater* can refer to the second last previous *rater*. If there are enough previous raters, there is always a previous rater that can do the verification.

For case 2, to prevent the previous *rater* from giving a forged revoked *Last-TimeStamp*, we require it to present a certificate by the revoker as well. If it cannot show any evidence of the revocation, the current rater can regard the *Last-TimeStamp* as a fresh one.

5 Conclusion

In this paper, we have look at how trust can be managed using reputation-based systems. Besides looking at existing solution, we have also presented our solutions to address the storage and integrity issues. In particular, we have proposed the notion of *RCert* and the RCertPX protocol. Although *RCertPX* can prevent tampering of the *RCert*, it cannot prevent malicious participants collude to distort the reputation information. For example, if the *ratee* and current *rater* collude, they might succeed to discard the latest ratings of the *RCert*. However,

with our mechanism, it is harder for the *rater* to achieve this as it will need to collude with N previous *ratees* at the same time. We are currently looking at how to address this collusion issue.

References

1. P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara. Reputation systems. In *Communications of the ACM*, 2000.
2. eBay. ebay home page. http://www.ebay.com.
3. Amazon. Amazon home page. http://www.amazon.com.
4. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
5. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 23–30, VIenna, Austria, July 2002.
6. W. S. Ng, B. C. Ooi, K. L. Tan, and A. Zhou. PeerDB: A p2p-based system for distributed data sharing. In *Proceedings of the 19th International Conference on Data Engineering*, Bangalore, India, March 2003.
7. P. Kalnis, W.S. Ng, B.C. Ooi, D. Papadias, and K.L. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *ACM SIGMOD 2002*, 2002.
8. K. A. Burton. Design of the openprivacy distributed reputation system. http://www.peerfear.org/papers/openprivacy-reputation.pdf, May 2002.
9. F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a p2p network. In *Proceedings of the eleventh international conference on World Wide Web*, 2002.
10. K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the tenth international conference on Information and knowledge management*, 2002.
11. K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *Proc. of COOPIS*, 2001.
12. D. Fahrenholtz and W. Lamersdorf. Transactional security for a distributed reputation management system. 2002.
13. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. Eigenrep: Repuation management in p2p networks. In *Proceedings of the twelfth international conference on World Wide Web*, May 2003.
14. P. Zimmermann. Pretty good privacy user's guide, volume i and ii. Distributed with the PGP software, 1993.
15. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.
16. PKI. Public-key infrastructure. http://www.ietf.org/html.charters/pkix-charter.html.
17. Gnutella. The gnutella protocol specification v0.4, june 2001. http://www.clip2.com/GnutellaProtocol04.pdf.
18. I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.