R-Chain: A Self-Maintained Reputation Management System in P2P Networks

Lintao Liu, Shu Zhang, Kyung Dong Ryu, Partha Dasgupta
Dept. of Computer Science & Engineering
Arizona State University
Tempe, AZ 85287
{lintao.liu, shu.zhang, kdryu, partha.dasgupta}@asu.edu

Abstract

Trust management is a critical component in P2P file sharing systems due to the free riding and security problem. In this paper, we propose a lightweight reputation management system, ***R-Chain****, where each peer maintains its own transaction history as the reputation. In R-Chain, each transaction is supervised by a few randomly chosen witnesses and each peer keeps a record for every transaction it has participated in. To ensure the integrity, transaction records are signed by the witnesses and organized as a chronicle chain. With very limited data stored on the network, no peer can modify or discard these transaction records without being detected. Compared with other reputation systems, our design is more light-weighted in terms of reputation maintenance and retrieval. It is scalable since each transaction only involves a constant number of peers. By recording the raw transaction history, R-Chain can also be integrated with different trust models to prevent various attacks.*

1 INTRODUCTION

With the increase in network bandwidth, storage space and computing power of modern personal computers, information sharing systems using Peer-to-Peer (P2P) techniques have gained tremendous attention in both academy and industry. P2P systems like Gnutella and KaZaA are used by a large number of users everyday, albeit for not-so-legal reasons. P2P networks however hold much promise for sharing and collaborations between ordinary users without the need for controlling servers and corporate interests.

While P2P networks provide an open and anonymous environment for information sharing, they are also vulnerable to various attacks, such as distributing tampered files or even spreading virus like VBS.Gnutella [1]. Due to the opaque-identification that each peer can create, change, discard its own identity, malicious peers cannot be identified and published for their misbehavior. Therefore, choosing a trustworthy peer for each transaction is very important. In addition, trust management can also be used to address the prevalent "free-riding" problem in P2P networks [2]. One key problem in a trust management system is the management of reputation data. Since it is stored and maintained on highly dynamic environment where unreliable peers keep joining and leaving frequently [2], it is critical to ensure that such data cannot be changed or lost.

Two different methods have been proposed to determine whether a peer is trustworthy. With the first method [3-5], peers only maintain their direct interaction experience with other peers. Before transaction, a peer polls or searches the network for the reputation of the target peer. While this method is simple and robust with low maintenance cost, it is expensive for reputation retrieval and can be biased due to the incomplete information collected. The second method is built on top of DHT-based P2P networks like [6-8]. It adapts a global storage scheme for reputation data, i.e. each peer is assigned with a "reputation manager" deterministically, which stores all feedbacks about this peer during previous transactions, and answer reputation lookup requests. While this method can consider the complete transaction history and provide more efficient reputation retrieval, it requires high maintenance cost. In addition, it keeps the reputation available all the time. Considering that a peer's uptime is usually very short, this is not only unnecessary but also too expensive. Moreover, it requires the participation of the whole system.

In this paper, we will propose a reputation management system called *R-Chain*, where each peer maintains its own transaction history as the reputation. The main design objective of R-Chain is to keep it lightweight for both reputation maintenance and retrieval. In addition, our system achieves:

1. Data Integrity: Even though it is stored locally, no peer can add, modify or delete any transaction history data.
2. Pseudo-anonymity: Each peer manages its own identity, which is not associated with any external identification information.
3. Adaptability: Each node can individually decide whether to participate in the reputation management for other peers.

Note that R-Chain presents a mechanism to maintain a peer's complete transaction history. It does not specify the metric to determine the trustworthiness of a peer. Instead, it can be integrated with any trust model that makes use of the transaction history. In this paper, we will use PeerTrust in [9] as the trust model.

2 RELATED WORK

According to the way reputation data is stored and retrieved, existing reputation systems in P2P networks can be classified into two groups.

In the first group, each peer only stores its own direct interaction experience with other peers, gained from previous transactions. For example in P2Prep [3], before a peer downloads a file from an unknown peer, it polls the network to ask all nearby nodes whether the target peer is trustworthy, and simply counts the replies. Similarly, [5] uses resource query for resource discovery and trust query to select a reliable one, but the feedbacks are weighted based on the reputation of the reviewers. Unlike [3,5], peers in [4] searches the network on the runtime and builds a trust graph which contains reference paths from the initiator peer to the target peer. Each edge in the graph represents how much the source trusts the sink. A reputation value is calculated based on this trust reference graph.

In the second group [10] and [11], a peer's reputation must be maintained and managed by its "reputation managers". And, reputation is retrieved by looking up, instead of searching or polling. The peers in [10] are organized in a virtual binary tree [6]. All complaints about a specific peer are stored on multiple agents deterministically. The reputation retrieval request about that peer is also forwarded to and answered by these agent nodes. However, storing only complaints cannot even distinguish a long existing good peer from a newcomer. EigenTrust [11] is based on DHT-based P2P systems like Chord [8]. In EigenTrust, a peer's reputation is stored and calculated by its "score manager". By requiring the whole network to participate, this method is resistant to cooperative cheating. However, the distributed reputation algorithm generates considerable network traffic by exchanging intermediate results iteratively.

RXCert [12] is very similar to our R-Chain, in the sense that the reputation data is stored locally and organized as a chain. But the reputation verification in RXCert requires the other party of the last transaction to be present. This significantly limits its usability. In addition, RXCert cannot guarantee the integrity of the reputation data and can be easily compromised by the collectives of malicious collaborating peers. Another similar work is proposed in [14] which also store reputation data on the owner node. However, it uses a very different identity management strategy.

Compared with above systems, R-Chain minimizes the maintenance and retrieval cost by maintaining the transaction history on the owner node. With other techniques, we also ensure that no peer can modify, discard or forge any reputation data; even it's stored locally on its owner.

3 R-CHAIN

In this section, we will present *R-Chain*, a transaction-based reputation management system. We assume that each transaction in R-Chain involves two equal parties and use file downloading as the example. Each transaction will result in a transaction record (*TR*) as the proof of its existence. Every peer saves the TRs as its reputation data for all the transactions it has participated in.

With all TRs saved on its owner, one obvious challenge is to guarantee the integrity. Every node, especially the malicious nodes, would like to keep only those favorable TRs and drop adverse ones. To address this problem, a few techniques are used: First, we introduce a neutral third party, called ***witness***, to supervise each transaction and sign the TR such that a peer cannot modify or forge a TR. Second, all TRs are linked together as a chain. Discarding any one in the middle will result in a broken chain, which can be easily detected. Third, to prevent the owner from deleting any TR at the end of this chain, we keep very limited information on the network for verification purpose. In the following subsections, we will discuss these techniques in more detail

3.1 Peer Identifier

As in many reputation management systems [3-5], R-Chain requires each peer to have a (public key, private key) pair, such that it can produce a unique and verifiable signature. Each node in DHT-based P2P networks also needs a node ID that should be evenly distributed in the ID space. For security reason, nodes should not be able to choose their own IDs. As a result, Pastry [7] assigns a random ID to each joining node while Chord [8] hashes the IP address as the node ID.

Initially, a node's ID has no relation to its (public key, private key) pair. But in R-Chain, given the public key of a peer, we should be able to locate the node efficiently. Therefore, we propose to use the one-way hashing of a peer's public key as its node ID, i.e.

Node ID = hash (public key)

This method not only satisfies all requirements, but also is compatible with original DHT design. First, since the node ID is the result of a one-way hashing function, a node cannot choose its own ID and all generated IDs will be evenly distributed over the ID space. Second, given the public key of a peer, we can calculate its node ID easily and thus locate it efficiently using the underlying P2P service. Moreover, we can verify a node ID by requesting its public key, which in turn can be challenged and verified. Note that this method maintains the pseudo-anonymity of peers. Each node can generate more than one (public key, private key) pair and there is no central authority involved for assigning IDs.

To prevent Sybil attack [13], where one single peer can create thousands of identities to affect the functionality of the whole system, R-Chain can enforce some POW (proof of work) for initial join, as proposed in [11], such that a peer can only have limited number of identities.

With a (public key, private key) pair, the witness node can sign the TR of the supervised transaction to ensure the integrity of a single transaction record. This also prevents the owner from forging TRs for nonexistent transactions. This will be explained in more detail in section 4 when we discuss the selection of witnesses. For clarity of exposition, for now we just assume that a set of honest witnesses is already chosen and will be always online. This assumption will be weakened in Section 4.

3.2 R-Chain: A chain of transaction records

While the signature of the witness can prevent the owner from modifying the content of TRs, the owner can delete the whole TRs. So, we organize all the TRs into a single linked chain. Because such a chain represents the reputation of a peer, we call it a peer's *R-Chain*. Figure 1 shows an example of Alice's current R-Chain. We assume Alice did her last transaction with Bob.

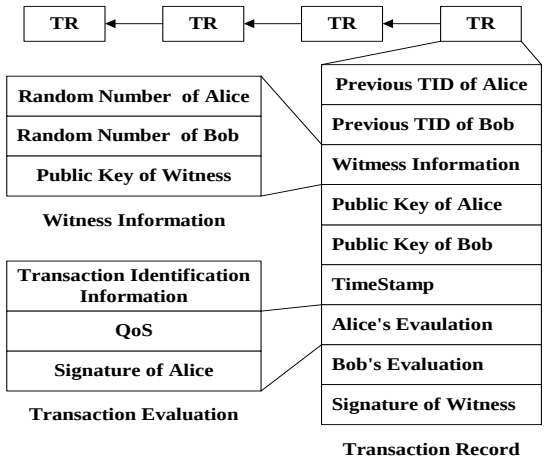

Figure 1: Data Structure of R-Chain

As we can see, each TR structure includes the previous transaction ID of Alice, which works as a pointer linking it to previous TR in the chain. By recording Bob's previous TID, the same TR is shared by both Alice and Bob. Note that there is not an explicit TID field in transaction record. We use the timestamp as the implicit TID. For the first TR in the chain, previous TID fields can be filled with some pre-defined values, e.g. 0.

Besides the previous TID fields, a TR also includes a few other fields used to identify the transaction globally, such as witness information, public keys of both parties and current timestamp. These fields are also included in the transaction evaluation. A transaction evaluation (TE) is used to indicate one party's opinion about the other party's service quality in the transaction. Besides the transaction identification information, it includes metric specific information used to evaluate the reputation. In Figure 1, we just use QoS as an example. Depending on the trust metric, more information can be included, such as the evaluator's own reputation.

With all TRs organized in a linked chain, the owner node cannot remove or change any elements in the middle. But it can still discard TRs from the end. To prevent this attack, we need to verify whether the last TR in the R-Chain really represents the last transaction the peer has participated in. In next subsection, we will discuss how this verification is performed during a transaction.

3.3 Transactions in R-Chain

To explain how discarding TRs from the end of R-Chain is prevented, let's look at the following transaction procedure: Assuming Alice wants to download a file from Bob, and they both want to check each other's reputation first. W is the chosen witness for this transaction. We assume that *W-A* is the witness for Alice's previous transaction; *W-B* is the witness of Bob's previous transaction. Figure 2 shows the detailed procedure.

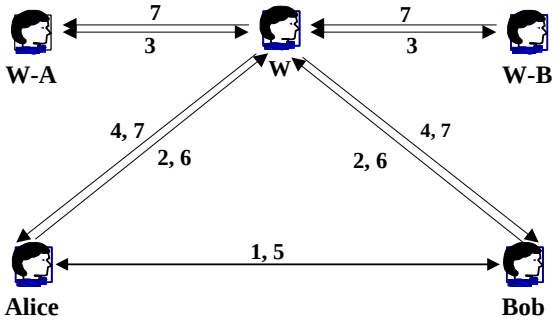

Figure 2: Transaction procedure in R-Chain

1. A (Alice) and B (Bob) exchange R-Chains;
2. A and B send each other's last TR and a pre-TE to W;
3. W contacts W-A and W-B for verification;
4. W asks A and B to proceed;
5. A downloads the file from B;
6. A and B send final TE to W;
7. W builds a TR and sends it to A and B. W keeps a TR locally and notifies W-A and W-B to remove their TRs.

There are a few important places we need to notice in this transaction procedure. First of all, to verify the authenticity of an R-Chain, W only needs the last transaction records in Alice and Bob's R-Chains; because W only needs to verify that the last two witnesses really supervised the last transactions of Alice and Bob. Second, since generating a TR needs the transaction evaluation (TE) from both parties (Figure 1), a malicious node can perform "*hit and run*", i.e. do some bad transaction and run away without sending a TE. To prevent this simple attack, both parties are required to submit a pre-transaction evaluation (pre-TE) before the real transaction happens (step 2). A pre-TE is exactly the same with a normal TE, except that the QoS field is filled with some reserved value indicating this is a pre-transaction evaluation. With this method, if a malicious node does not submit a final TE, the witness can still build a TR, where the special value in QoS field indicates this peer did not submit a valid final TE. Pre-

transaction evaluations can also better protect the peer privacy. Only with a pre-transaction evaluation, W-A and W-B will answer the verification request. With previous TID field and the digital signature in a pre-TE, only Alice and Bob can authorize a witness to check the status of their R-Chains (step 4).

In step 7, the new TR will be also present to W-A and W-B as a proof that Alice and Bob have finished a new transaction. So the old TRs saved on W-A and W-B should be removed. As a result, Alice and Bob have to give the complete R-Chains, including the new TR, for later transactions, since W-A and W-B has forgotten about the old transaction and cannot provide verification any more.

To this point, we explained why a peer cannot discard any TR from its R-Chain. Only the witness of last transaction needs to store one TR for verification. This is a distributed and localized transaction protocol. No central server is required and only constant number of nodes is involved for each transaction, in spite of the network size.

4 WITNESS

To describe the protocol, we had made two unrealistic assumptions about witnesses: First, the witness is neutral and honest. Second, the witness will always be online to provide verification. In this section, we will present how the system still works without these rigid assumptions.

4.1 Selection of Witness

Due to the important responsibility of the witness in R-Chain system, no peer should be able to choose its friend as the witness. So, we propose to use one-way hash function to decide the ID of the witness. One simple way would be:

$$\overline{ID} = h(R_{Alice}, R_{Bob}) \quad (1)$$

where R_{Alice} and R_{Bob} are two random numbers chosen by Alice and Bob, and they are exchanged during the transaction setup, along with R-Chains. With this method, even though the witness is chosen by Alice and Bob, they cannot choose any desired peer, just as they cannot select their Node IDs. So, the chosen witness should be neutral and have no preference for either Alice or Bob.

There are two problems with this simple method: First, it is very likely there is no such node whose ID is equal to $\overline{ID}$. To address this problem, we take $\overline{ID}$ as a key to find out the node that is responsible for it, and use this node as the witness. For example, in Pastry, the witness would be the node whose ID is closest to $\overline{ID}$ numerically. In Chord, the witness would be the successor of $\overline{ID}$. With the underlying P2P lookup service, this node can be efficiently located. Second, this method is vulnerable to reuse attack. A node can reside in the network and wait until it is chosen as the witness. Then, it can reuse these two numbers and choose itself as the witness. Considering that, we can use more input for the hashing function to prevent reusing as an enhancement:

$$\overline{ID} = h(K_{(Alice,\ public)}, K_{(Bob,\ public)}, R_{Alice}, R_{Bob,}\ Timestamp) \quad (2)$$

Including the public keys of both parties and the current time can effectively prevent reuse attack. The two random numbers give us the chance to select more than one witness, for example, in case that the peer with the chosen ID is not willing to act as a witness.

Now we can revisit Figure 1 and explain the witness information included in TR and TE: The two random numbers are used to generate $\overline{ID}$. The public key of the chosen witness is used to generate $ID_{witness}$. $ID_{witness}$ and $\overline{ID}$ should be close enough to make people believe that the witness was responsible for key $\overline{ID}$. With such a structure, transaction records are by itself verifiable. Combining the way node IDs are generated, it is extremely difficult for a peer to forge a transaction record, because it needs to generate a (public key, private key) pair, such that the corresponding node ID (i.e. hashing of the public key) matches the desired witness id: $\overline{ID}$, which is the hashing of $(K_{(Alice,\ public)}, K_{(Bob,\ public)}, R_{Alice}, R_{Bob,}\ Timestamp)$.

However, a dishonest peer can be selected as witness as well. And a dishonest witness will can the R-Chains of both parties, by refusing to generated a valid TR or discard it afterwards. To address this problem, we can select multiple witnesses for each transaction. With the example in Figure 2, Alice and Bob can generate k (e.g. k=3) pairs of random numbers to select k witnesses. Considering the way that witnesses are chosen, with a large enough k, it is very unlikely all k witnesses are dishonest. To make the system even stronger, two transaction parties can first select more than k nodes. Then, they choose the k nodes with better reputation among them as the final witnesses. This method can further reduce the probability that a peer loses its reputation chain due to all witnesses are malicious without increasing too much overhead.

It is possible that a dishonest witness keeps an obsolete TR that should be removed. But considering that the owner and the witness are strangers and the witness can't benefit, we believe this will not be a common attack in our system.

4.2 Absence of Witness

While only online nodes can be chosen as witnesses to supervise transactions, a witness could be offline when the verification is required. In this sub-section, we will discuss how to handle this problem.

We observed that R-Chain verification doesn't require the previous witness to be present because we neither challenge its private key nor use its private key to sign any data. A R-Chain is authenticated if the last TR does exist in the network. The TR itself, as we explained earlier, cannot be forged. So, instead of asking the witness to keep a copy of the TR, we insert a ($\overline{ID}$, TR) pair into the P2P

network, after a witness signs the TR. Recall the way the witness is selected. This ($\overline{ID}$, TR) pair will be saved on the witness itself. When the witness leaves the network, this pair will be handed over to a new node, as any other data managed by P2P system. Such handover can happen again if this new holder leaves or a better holder joins. By this way, we can keep this ($\overline{ID}$, TR) pair always available until the verification is required.

($\overline{ID}$, TR) pair could be lost due to node crash or handover to malicious nodes. This is a similar problem as dishonest witnesses. However, multiple witnesses can create multiple pairs to provide higher availability, as for malicious witnesses during transactions.

There is one possible attack that a node re-inserts one of its old TRs into the network, hoping to use it for verification again. This can be easily prevented by requiring that a TR can only be handed over by the current holder to a new holder. Since the node ID of the owner is far from $\overline{ID}$, it can never be the holder of ($\overline{ID}$, TR) and thus insert it to other peer (Otherwise, it would have selected itself as the witness).

5 TRUST MODEL

While R-Chain presents a mechanism to record the transaction history, we don't specify how to determine a peer's trustworthiness. Instead, R-Chain can be adapted with any trust model that makes use of the transaction history. As an example, we will use PeerTrust from [9] where the trust of peer *w* on peer *u* is calculated as:

$$T(w,u) = 1 - \frac{\sum_{u \neq v} C(u,v) * T(w,v)}{I(u)} \qquad (3)$$

In this model, *I(u)* is the total number of transaction that peer u has participated in. *C(u, v)* is the number of complaints (or negative feedbacks) peer *v* issued to peer *u*. This model only count on the number of complaints a peer received, while it weights the feedbacks based on the reputation of the reviewers.

6 EVALUATION & COMPARISON

In this section, we will simulate a P2P file-sharing network to demonstrate that R-Chain is a robust and lightweight reputation management system.

In our simulation, we create a network with 1000 nodes and 3000 files. There are totally 400000 user queries. Both the content popularity and the user queries follow a *Zipf* distribution. The percentage of malicious peers varies from 0% to 30%. We assume that a malicious peer will return inauthentic files with a probability of 0.50 while a good peer will return inauthentic files with a probability of 0.05. When a user query returns multiple results, we randomly select one while their probabilities of being chosen is based on their reputation.

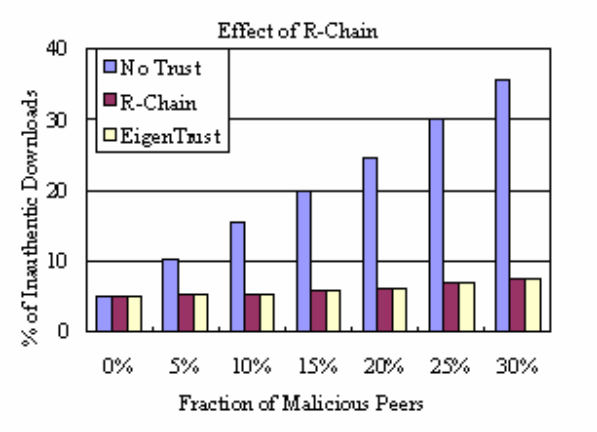

Figure 3: Effective of R-Chain

The first experiment is to show the effectiveness of R-Chain. We compare R-Chain with the cases that no trust management and the distributed EigenTrust system. Figure 3 shows the experiment results. Compared with the case where no trust management system is applied, both R-Chain and EigenTrust can greatly reduce the inauthentic file download fraction, by effectively distinguish malicious peers from normal ones.

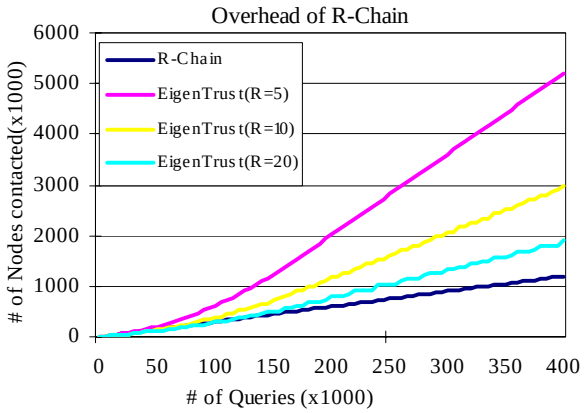

Figure 4: Overhead of R-Chain

The second experiment is to compare the overhead of R-Chain and EigenTrust. While EigenTrust always computes the latest global trust values for all peers, R-Chain uses "on-demand" reputation calculation. Considering P2P systems where only a small fraction of peers are active and a peer's reputation global trust value could change many times before it is really required, always calculating the latest reputation for all peers is expensive yet unnecessary. In contrast, R-Chain removes all off-transaction overheads, in the expense of transferring R-Chains during a transaction.

Assuming one witness for each transaction in R-Chain and one score manager for each peer in EigenTrust, we count how many nodes are contacted for 400000 transactions, since most messages are very small. In R-Chain, each transaction involves 3 extra nodes: current

witness and two previous witnesses for both parties. On the other hand, the overhead of EigenTrust algorithm depends on how frequently the system updates the reputation. We define a parameter R (R = 5, 10 and 20 in our experiments) as the average number of transactions each peer has done between two iterations. R cannot be too large since the system must catch up with the reputation change. In addition, EigenTrust needs to contact more peers as more transactions are performed. Our simulation contacts maximum 50 peers, by discarding old transactions. Figure 4 demonstrates the experiment results.

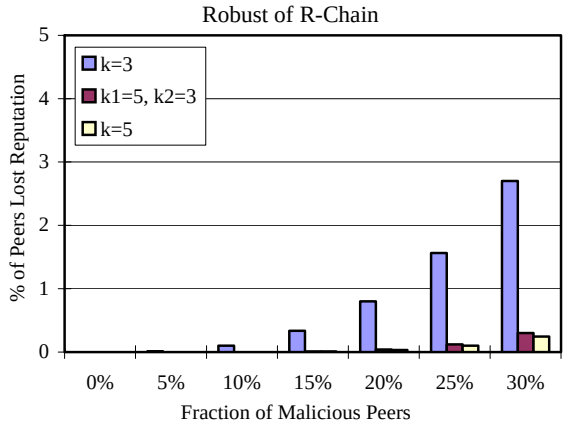

Figure 5: Robust of R-Chain

The third experiment is to illuminate the robustness of R-Chain system. With k witnesses for each transaction and m be the fraction of malicious peers, the expected probability that a peer loses its reputation is:

$$P = m^k$$

With only 3 witnesses per transaction and 20% peers are malicious, only less than 1% of the peers would lose their reputation. Although more witnesses can provide higher security to the reputation, it also produces more overheads. So, we also experiment another method: first choose k_1 nodes randomly and then select k_2 nodes with better reputation among them to be witnesses. Figure 5 shows the experiment results. Furthermore, selecting 3 witnesses out of 5 randomly chosen nodes can perform much better than simply choosing 3 witnesses randomly and almost the same as the case where 5 random witnesses, especially when the fraction of malicious peers is small.

It is worthy to notice that the number of witnesses has almost no effect on inauthentic downloads. The reason is: even some nodes could lose their reputation; the percentage is too small to affect the overall number of inauthentic downloads.

7 CONCLUSION

In this paper, we present R-Chain, a lightweight reputation management mechanism for DHT-based P2P networks. In our design, each peer maintains its own reputation information locally, which greatly reduces the reputation maintenance and retrieval cost. A small number of witnesses are randomly chosen to supervise each transaction and sign the transaction record. With very limited data stored on the network, we ensure that these transaction records cannot be changed, discarded or forged. In spite of the network size, each transaction only involves a constant number of nodes, thus making the system scalable for large P2P networks.

8 REFERENCES

[1] VBS.Gnutella Worm: http://securityresponse.symantec.com/avcenter/venc/data/vbs.gnutella.html

[2] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking (MMCN)*, 2002.

[3]F. Cornelli, E. Damiani, S. D. C. d. Vimercati, and S. P. Samarati, "Choosing Reputable Servents in a P2P Network," *Proc. WWW 2002.*

[4] S. Lee, R. Sherwood, and B. Bhattacharjee, "Cooperative Peer Groups in NICE," *Proc. IEEE Infocom*, 2003.

[5] A. A. Selcuk, E. Uzun, and M. R. Pariente, "A Reputation-Based Trust Management System for P2P Networks," *Proc. 4th Int'l workshop on GP2PC*, 2004.

[6] K. Aberer, "P-Grid: A self-organizing access structure for P2P information systems," *Proc. Sixth Int'l Conference on Cooperative Information Systems*, 2001.

[7] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *Proc. IFIP/ACM Int'l Conference on Distributed Systems Platforms*, 2001.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and Hari Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM*, 2001.

[9] L. Xiong and L. Liu, "A Reputation-Based Trust Model for Peer-to-Peer eCommerce Communities," *Proc. IEEE Int'l Conference on E-Commerce*, 2003.

[10] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," *Proc. the tenth int'l Conference on Information and Knowledge Management*, 2002.

[11] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks," *Proc. WWW 2003*.

[12] B. C. Ooi, C. Y. Liau, and K.-L. Tan, "Managing Trust in Peer-to-Peer Systems Using Reputation-Based Techniques," *Proc. 4th Int'l Conference on Web Age Information Management (WAIM)*, 2003.

[13] J. Douceur, "the Sybil Attack," *Proc. IPTPS'02*, 2002.

[14] P. Dewan "Peer-to-Peer Reputations" WPDRTS 2004