

The Application of Collaborative Filtering for Trust Management in P2P Communities

Min Zuo*, Kai Wang, and Jianhua Li

Department of Electronic Engineering, Shanghai Jiaotong University, Shanghai, China
{zuomin, smile_shining94, lijh888}@sjtu.edu.cn

Abstract. The open and anonymous nature of P2P services opens the door to malicious peers who cause the loss of trust by providing corrupted data or harmful services. The introduction of a trust management system is one of the possible ways to combat this problem. This paper presents some new ideas for the design of a P2P trust management system. Its main contributions include: a recommendation-aggregating model based on collaborative filtering (CF), a polling protocol for trust queries and responses, and the use of identity-based cryptosystem to secure recommendations. Simulations show that our CF-based trust model performs pretty well even when malicious peers make the majority.

Keywords: P2P, Trust management, Reputation system, Collaborative filtering.

1 Introduction

Since the appearance of Napster in 1999, P2P (Peer-to-Peer) applications, especially file-sharing applications such as Gnutella, KaZaa, eDonkey and BitTorrent, have become the largest traffic source on the Internet. According to a report by CacheLogic [1], in the first half of year 2004, P2P traffic volumes in Europe are at least double that of HTTP during peak evening periods and as much as tenfold at other times, and at the same time, P2P traffic is continuing to grow despite the disturbances from recording industry (RIAA and others) every now and then.

P2P is a great appeal mainly because it is more efficient than any other ways to distribute large volumes of data through Internet connections. Another reason for P2P's popularity is that, P2P is less dependent on central servers, thus makes it easy for everyone to share information or to work together with others. The increasing availability of broadband Internet connections and low-cost PCs, together with improved multi-media compression technologies, also stimulates the adoption of P2P file sharing and exchanging technologies.

However, P2P services nowadays are still far from being satisfying, and there are yet many people and companies who are hesitating to use them.

Problems with P2P mainly concern security, reliability, privacy or copyright. P2P communities are often established dynamically with peers who are unknown to each

* Supported by the National High-Tech Research and Development Plan of China (863) under Grant No.2003AA142070.

other. These peers communicate directly with each other to exchange information, distribute tasks, or do businesses. Usually there is no authoritative supervision over the trading process and the involving two parties. For a peer who wants to transact with others, there is no guarantee that other peers will act properly as they have claimed. Therefore, P2P systems are extremely vulnerable to malicious users trying to poison the system with corrupted data or harmful services for personal or commercial gains, or just for monkeyshines. And it is up to the peers to protect themselves.

When facing these problems, users of P2P applications must be wary of the quality and validity of the resources or services they wish to order. They need a mechanism to guide them in making decisions such as whether or where to download a file, and whether or with whom to do a business. In other words, they need a mechanism to evaluate the trustworthiness of an item (a peer, a file, a service, etc.) *before* they actually get to it. That is the main purpose of so called P2P trust management systems.

The above-mentioned problem is quite similar to that of collaborative filtering (CF) in recommender systems. CF-based recommender systems try to predict which items a given user might like without using any information about the actual content of the items. But rather, they use a database of users' preferences (implicit ratings or explicit ratings). Similarly, in a P2P community, a peer can gather from other peers the feedback ratings about trustworthiness of the items it is interested in *before* it actually gets to them, then it can operate over these feedback ratings using a CF-like technology, and pick out items that might be trustworthy from its point of view.

Starting from this point, in this paper, we will borrow the ideas from the research field of collaborative filtering and put forward our P2P trust model. The rest of this paper is organized as follows: firstly we discuss some related work in section 2; then we present our CF-based model in section 3, followed by the distributed communication protocol in section 4; we discuss the security issues (identities and keys) in section 5, where we propose the use of an identity-based public key system; finally, we give some experimental results in section 6 and conclude this paper in section 7.

2 Related Work

Along with the rapid development of P2P technologies, a large body of literature on P2P trust management has sprung up in the past three to four years. Similar to [2], we classify most of the existing P2P trust management systems into three categories: police-based systems, social network-based systems, and reputation-based systems.

Some typical examples of policy-based trust management system are PolicyMaker [3] and KeyNote [4]. To our best knowledge, the notion of trust management was first introduced by M. Blaze et.al. in 1996[3]. In that paper, trust management problem is defined to be a collective of security policies, security credentials and trust relationships. The authors also designed an architectural framework named PolicyMaker for distributed trust management. Later in [4], the same authors presented the successor of PolicyMaker which is called KeyNote. KeyNote has been accepted as a RFC standard. However, these frameworks focus more on access control than on resource/peer selection, so they are not suitable for the scenario we are discussing.

Social network-based systems make use of social relationships to evaluate trust. They try to reconstruct a social network which represents the relationships within the

community, and then draw conclusions about peers' trustworthiness based on different aspects of this social network. Examples of such trust management systems include group-identifying systems [5] and expert-identifying systems [6], etc.

There are two mainstreams of reputation-based trust management systems. One is based on so called "web of trust". The other is based on reputation ratings.

"Web of trust" is a phenomenon reflecting the fact that trust is conditionally transitive. For example, if A trusts B and B trusts C, then A will probably trust C to some degree. A typical example of "web of trust" is the PGP community [7]. Some distributed trust management schemes, such as [8] and [9], are based on this model. The main point of these systems is to find out at least one path from the truster to the trustee, and evaluate the degree of trust using some aggregating methods along the paths.

The validity of "web of trust" is limited by the length of the paths because degree of trust declines rapidly when the number of intermediators increases. When this happens, it will be a better way to base a trust decision on reputation ratings. The main point is to gather a number of others' ratings on the target, and aggregate these ratings to draw conclusions about the target's trustworthiness. A brief overview of such systems is presented in [10]. The eBay's feedback forum is a successful de facto example of server-based centralized online reputation system. A considerable fraction of P2P trust management schemes belong to this category. Typical examples include P-Grid [11], P2Prep [12], EigenTrust [13], PeerTrust [14] and RobustRep [15], etc.

3 Model Description

3.1 Rating Matrix

Suppose there are n peers in a P2P community, and m items rated by them. Here the items could be peers or resources (for example, files), depending on the specific application and requirements.

Let \mathbf{R} be the matrix of the peers' ratings, where R_{ij} is the rating given by peer i to item j ($i \in \{1, 2, \dots, n\}, j \in \{1, 2, \dots, m\}$). We set $R_{ij} = 0$ if peer i has not rated item j , and require the actual ratings to be non-zero. A row vector \bar{R}_i consists of peer i 's ratings for all the m items. Note that this vector represents peer i 's personal judgment from direct experiences. Typically \mathbf{R} is a *sparse* matrix with many missing ratings, especially for a large-scale P2P community perhaps with millions of peers.

When a peer a wants to evaluate the trustworthiness of an item x which it has not rated, it will have to ask for recommendations from others who have rated the item.

Those recommendations, however, are probably not all honest and reliable. On the one hand, malicious peers may give misleading recommendations for personal or commercial gains. For example: a "badmouthing" peer may give bad ratings to all the others; a bunch of malicious peers may collude to give good ratings to each other and bad ratings to outside peers; etc. On the other hand, personal experiences may differ. For example: peers may feel differently about a file and make subjective judgments on its "trustworthiness" which could be different or even contradicting; some peers may act honestly to reputable peers but dishonestly to newcomers; etc. Therefore, peer a needs some algorithm to have these recommendations properly aggregated so that it can work out a best evaluation of the target x 's trustworthiness.

Typical aggregating algorithms include threshold decision, majority voting and weighted summation. Threshold decision is a kind of heuristic algorithm. Some preset threshold values are required, and this usually makes the most difficult part in application. We refer the readers to [11] for an example. Majority voting is an intuitive method to deal with large number of ratings or votes. It works pretty well if honest peers are the majority. But if the total number of malicious peers exceeds that of honest ones, then a reputation system based on majority voting can be easily subverted. An improvement of majority voting is weighted summation, where recommendations from honest peers are honored while those from dishonest peers are punished before summing up. For example, EightTrust [13] weights peers' opinions by the evaluator's trust on them, and RobustRep [15] keeps a separate recommender trust vector and weights the recommendations using the values in this vector.

Here we propose a weighted summation algorithm using the personal similarities between the evaluator and the recommenders – a CF-based aggregating algorithm.

3.2 Collaborative Filtering Algorithms

Collaborative filtering is widely adopted in recommendation applications for music, news, commodities etc. With collaborative filtering, users can get personalized predictions and suggestions to help them find what they want with a higher probability.

Most of today's collaborative filtering algorithms are based on ratings from most "similar" users. According to [16], these algorithms can be classified into two categories: memory-based algorithms and model-based algorithms.

Model-based algorithms are based on an operation called dimensionality reduction. This kind of algorithm first builds a model of user preferences (low dimension) from the rating database (high dimension), then makes predictions using this established model. Bayesian networks [16] and SVD (Singular Value Decomposition) [17] are typical examples. Because the model is usually a low dimensional one, these algorithms can be very instant in making online recommendations after the model is established. But these algorithms all need a learning phase, which may take several hours, even several days long. Therefore, these algorithms can't adapt easily to dynamic rating databases, so we won't use them in our P2P trust management system.

Memory-based algorithms, in contrast, build no prediction models but operate over the entire rating database to make each recommendation. In this case, the predicted rating of user a for item x is a weighted sum of other users' ratings, as in formula (1).

$$R_{ax}^{predicted} = \bar{R}_a + k \sum_{i=1}^n w(a,i)(R_{ix} - \bar{R}_i) \quad (1)$$

Here \bar{R}_a and \bar{R}_i is the averaged rating of user a and user i respectively, $w(a,i)$ is the weight between user a and user i , and k is a normalizing factor.

The weights $w(a,i)$ usually reflect distance, correlation or similarity between users' personal experiences. Users with high weights to a given user can be seen as "neighbors" of that user. Therefore memory-based algorithms are also called neighbor methods. There are several different ways to define the weights $w(a,i)$, typical examples are Pearson correlation and Cosine similarity [16].

Memory-based algorithms may be slower in making online recommendations because the whole rating database must be checked to find neighbors of a given user.

However, they need no learning phases, so they can adapt more easily to the dynamic rating database of a P2P trust management system. Moreover, memory-based collaborative filtering algorithms are known to be more accurate than model-based algorithms [16]. Therefore, we will choose some algorithm of this kind as our recommendation-aggregating algorithm. As for the choice of the weighting function, it will depend on the features of the specific application, such as, the density of the rating matrix, the form of the ratings (binary, discrete or continuous), etc.

4 Polling Protocol

Today's collaborative filtering systems are mostly server-based systems. User ratings are collected by a central server and stored at the server as a centralized database. The collaborative filtering algorithm is executed at the server in response to users' queries. However, things are different on P2P networks. Usually there are no central database servers or computing servers in a P2P environment. So, users' ratings are distributed in the community and computing tasks should also be fulfilled by the peers themselves or in a distributed fashion. This urges an appropriate mechanism for peers to communicate with each other and gather the data they need for trust decisions.

Some researchers propose the use of DHT (Distributed Hash Table) to store the reputation ratings [11,13,14], but DHTs seem to be not suitable for unstructured P2P systems such as BitTorrent, KaZaa and eDonkey, which are the most popular P2P applications. First, maintaining a DHT is too costly when peers frequently join and leave the system. Second, some incentive mechanism is needed to ensure that peers would like to take their responsibility as part of the DHT, but this incentive problem appears to be even more troublesome than the problem of evaluating trust itself [18]. Third but not last, strong cryptological methods must be adopted to prevent peers in the DHT from modifying the data they stored, and this can be too complicated to implement.

In contrast to DHTs, we argue that a broadcast or multicast polling protocol will fit better into an unstructured P2P system. And we've tried to develop a protocol which works in a Gnutella-like unstructured purely distributed fashion. It's a derivation of Damiani et al's polling protocol in [12]. This protocol runs as follows:

Step1: A peer (noted as A) uses the searching mechanism provided by the P2P community to find the resources he wants. When this searching finishes, A gets a list of candidate resources and/or their owners' IDs.

Step2: Peer A picks out some limited number (for fear the query message might be too long) of target items in the list from step1, and broadcasts a request-for-recommendation querying message containing the following information:

Request::=Issuer_ID | Target_SET | Desired_length
Target_SET::= SET OF {Target_ID}

Issuer_ID is the ID of peer A. Target_ID is the ID of the target items (resources and/or their owners). Desired_length is the desired number of ratings in a recommendation vector. This vector contains the recommender's ratings for several other items as well as one or more of the target ones. It is used to compute the similarity.

Step3: When another peer (noted as B) receives this query message and is willing to give a recommendation, he first checks his "blacklist" to make sure that the issuer

A is not in that list. Then he looks for the target IDs in his own transaction history log. If there are at least one hit, then he constructs a response message like the follow:

$$\begin{aligned} \text{Response} &::= \text{Responder_ID} \mid \text{Item_SET} \mid \text{Rating_vector} \\ \text{Item_SET} &::= \text{SET OF } \{ \text{Item_ID} \} \\ \text{Rating_vector} &::= \text{VECTOR OF } \{ \text{Rating} \} \end{aligned}$$

Responder_ID is the ID of peer B. Item_SET contains the IDs of the target item(s) hit in B's log, and some other randomly chosen items in order to reach the "Desired_length" in the query message. Rating_vector contains the ratings corresponding to the items in Item_SET.

Peer B encrypts the Item_SET field with A's public key and signs the whole message with B's private key, and then transmit the message to A. (The problem of identity and key management will be discussed in section 5.)

Step4: Peer A waits for responses until timeout or enough recommendations have been gathered. It checks the signature for each received response. If the signature is invalid, the message will be discarded as a fake recommendation. Otherwise, the encrypted part will be decrypted using A's private key. After that, it reconstructs a rating matrix from these recommendations and carries out a CF-based aggregating algorithm to draw conclusions of the target items' trustworthiness.

Step5: If A is satisfied about some of the targets, it will begin to requests for them directly from the owners. Otherwise, another round of querying will be carried out for some other limited number of items in the list from step1.

In this polling protocol, signatures on the response messages ensure the integrity and authenticity the recommendations. This not only prevents malicious peers from modifying other peers' response messages during transmission, but also can be a disincentive for them to give too many misleading recommendations, because that can put them into other peers' blacklists.

Encryption on Item_SET in the response message aims at protecting the recommender's privacy. Due to the encryption, only the peer issuing this query message can read the content and have some knowledge about the recommender's personal transaction history. Together with the randomness in the choice of items in his log, the recommender can feel safe that no one else can get his full transaction history.

5 Identity and Key Management

An identity and key management mechanism is necessary for the polling protocol described in section 4. We do not recommend the use of PKI public key certificates, because certificate management is too costly and inefficient for a distributed P2P system. To avoid using public key certificates, we propose a key management system based on so-called "identity-based" cryptography.

Identity-based cryptography was first introduced by Shamir in 1984 [19]. His original motivation was to simplify certificate management in e-mail systems. In an identity-based cryptosystem, instead of generating a random pair of keys and publishing the public one, a user chooses his name or other personal identification information as his public key. Because the public key is the ID itself, there will be no need for a certificate to bind the public key to the user's ID. A user can authenticate himself to

a KGC (Key Generation Center) and obtain his private key. The only purpose of a KGC is to fix the system parameters and give each user a personal “card” when he first joins the community. The KGC can be closed after all the users have got their “cards”, and the system can continue to function in a totally independent way.

Here we suppose a certain number of register servers in a P2P community. They act as a distributed KGC of an identity-based cryptosystem. The master-key of the KGC is distributed among the servers using techniques of threshold cryptography so that no single location can get the whole master-key [20].

Every peer in this community should register on one of these servers in order to get a unique ID (it is also the public key) and a corresponding private key, if he wants to participate in the trust management system. There can also be unregistered peers, but these peers can’t use the function of the trust management system.

Note that peers only need to contact a register server for once. They don’t have to do the logins and logouts each time they get online and offline, as in a centralized system like eBay. As long as the peers finish the registering, they can authenticate each other using their IDs and private keys without the interference of a central authentication server. So the existence of a register server is not a contradiction with the philosophy of P2P. Moreover, the existence of a register server can increase the cost of pseudonyms and so to some extent solve the problem of so-called “sybil attack”[21], which is a great threat to the functioning of a reputation system.

6 Simulations and Experiments

We evaluate our CF-based recommendation-aggregating algorithm by simulation experiments. These simulations are done using MatLab 5.3 on a PC machine with a P4 1.4G CPU and 256M RAM. Our concentration is on the algorithm’s effectiveness when facing various kinds of malicious users. We didn’t simulate the distributed polling protocol because it only influences the time efficiency and traffic overload but has little impact on the accuracy.

6.1 Threat Models

We consider three kinds of malicious peers (similar to those in [22]):

1. *badmouthing*: these peers always act dishonestly and give bad ratings to everyone else (good or bad).
2. *colluding*: these peers cluster into one or more collusion groups. They act dishonestly to all the good or bad peers outside their group, deliberately give good ratings to the malicious peers in their group, and give bad ratings to all the others.
3. *front peers*: so called “front peers” usually belong to some collusion group. They act honestly during transactions to gain a good reputation, and then spread their malicious ratings to subvert the functioning of the reputation system.

We make the assumption that a malicious peer always acts maliciously according to some of the above patterns. That is to say, we didn’t consider the dynamic personality of the users, or the misbehaviors caused by the users’ incidental carelessness.

6.2 Simulation Setup and Description

Our simulated P2P community consists of 1,000 peers. (We also did the same experiments with a community of 100 and 10,000 peers, and the results were largely the same.) The peers rate each other about their trustworthiness in transactions. The rating matrix \mathbf{R} is a 1000*1000 square matrix.

We randomly choose $x\%$ ($x = 10 \sim 90$) of the peers as malicious ones. We assume that good peers always act honestly in transactions and provide honest feedback afterwards, while malicious peers act according to the different threat models described in section 6.1.

We use a binary feedback system similar to that of eBay. If a peer thinks another peer is trustworthy, 1 will be given as the rating; otherwise, -1 will be the rating value. If a peer has no experience with another peer, the corresponding value in the rating matrix will be 0 meaning ignorance.

To accumulate enough ratings before implementing the aggregating algorithms, we perform 10,000 transactions between randomly chosen pairs. Because the chosen pairs may be the same in different transactions, the density of the resulting rating matrixes should be less than 2% (each transaction results in 2 ratings in the matrix). They are quite sparse matrixes meaning most of the peers do not know each other.

For each of the resulting rating matrixes, a good peer is randomly selected as the observer (or evaluator). Four algorithms are executed by the observer to evaluate the trustworthiness of all the other peers in the community:

Algorithm 1: The observer only trusts his own direct experiences. No recommendation is accepted.

Algorithm 2: For peers he has rated, the observer trusts his own ratings. And for each peer he has not rated, all the non-zero ratings in the corresponding column are summed up followed by an application of function $\text{sign}(x)$, where

$$\text{sign}(x) = 1(\text{if } x > 0) \text{ or } -1(\text{if } x < 0) \text{ or } 0(\text{if } x = 0) \quad (2)$$

This is equivalent to majority voting.

Algorithm 3: For peers he has rated, the observer trusts his own ratings. And for each peer he has not rated, a weighted summation of all the non-zero ratings in the corresponding column are computed followed by an application of function $\text{sign}(x)$. The weight of the rating in the i th row, written as $w(i)$, is the dot product of row vector $\bar{R}_{\text{observer}}$ and \bar{R}_i :

$$w(i) = \bar{R}_{\text{observer}} * \bar{R}_i \quad (3)$$

The dot product reflects the similarity between the two peers' opinions. Thus, algorithm 3 can be seen as an application of neighbor-based CF technology.

Algorithm 4: For peers he has rated, the observer trusts his own ratings. And for each peer he has not rated, the function $\text{sign}(x)$ is applied to the dot product of the corresponding column vector and the row vector $\bar{R}_{\text{observer}}$. This operation stands for a weighted summation by the observer's direct trust placed on the recommenders.

After the evaluation, an aggregated trust vector is compared with the peers' real reputations (1 for good peers, and -1 for malicious peers) to compute an error rate (er-

ror ratings divided by total number of peers) and a coverage rate (non-zero ratings divided by total number of peers).

A brief overview of the simulation setup is presented in Table 1.

Table 1. Simulation setup

Decription	value
Number of peers in the community	1,000
Percentage of malicious peers	10% ~ 90%
Percentage of front peers in a collusion group (if any)	10%
Number of transactions	10,000
Number of aggregating algorithms compared	4

6.3 Results and Discussion

The results are presented in figure1 to figure 3 for the three different threat models described in section 6.1. All the results have been averaged over ten runs of the experiments.

First look at the left part of figure1 through figure 3. The least error rate is achieved by algorithm 1 under all the three threat models, and the error rate remains stable when the percentage of malicious peers increaces. The performance of algorithm 3 and 4 is also not bad in the terms of error rate. On the contrary, the error rate of algorithm 2 increases rapidly when the percentage of malicious peers increases, and it is even worse off if malicious peers collude (figure2 and figure3). This implies that one’s own experiences are the most valuable and reliable foundation when making trust decisions, especially in an untrustworthy environment where majority rating will lead to error results.

Then look at the right part of these figures. Although algorithm 1 achieves the lowest error rate, its coverage rate is the lowest, too. In contrast, the error rate of algorithm 4 is similar to that of algorithm 1, but the coverage rate is considerably increased. The best coverage rate is achieved by algorithm 2 and 3. We know that algorithm 1 means depending only on one’s own judgement, while the other algorithms mean asking for advices from other peers if one can’t make a judgement by oneself. This tells us that personal experiences are always limited, so experience-sharing is necessary for a successful trust management system.

Coverage rate is a very important metric for a trust management system. Algorithm 1 is the most conservative one, and algorithm 4 makes only one step further. They both suffer from a low coverage rate (less than 40%). That means they fail to give an evaluation result in most of the cases. Algorithm 2 and algorithm 3 outperform them as far as this is concerned.

On the other hand, if we compare the performance of algorithm 2 and algorithm 3, we can see that: firstly, our CF-based algorithm (algorithm 3) achieves a comparable coverage rate with majority voting (algorithm 2) even when the rating matrix is very sparse (with a density less than 2%); secondly, it performs far better than majority voting in terms of error rate when malicious peers make the majority. The error rate of our CF-based algorithm increases very slowly (figure 1) or remains stable (figure

2,3) when the percentage of malicious peers increases. It achieves an error rate lower than 10% even when 90% of the peers are malicious. The advantage of our CF-based algorithm is even more remarkable when malicious peers collude (figure 2,3).

Here we can reach the conclusion that our CF-based algorithm can reach a good balance between error and coverage, thus is the best one among the above 4 kinds of algorithms.

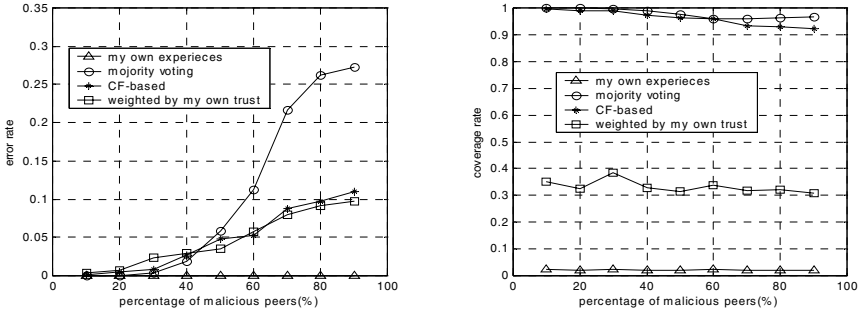


Fig. 1. Badmouthing malicious peers

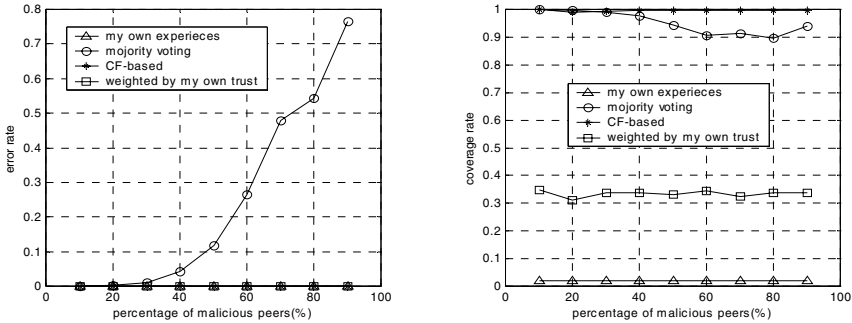


Fig. 2. Colluding malicious peers

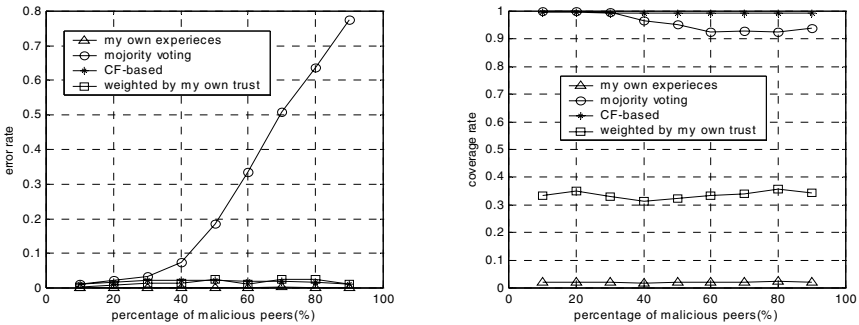


Fig. 3. Colluding malicious peers with 10% of the malicious peers as front peers

7 Conclusion

In this paper, we present a trust management scheme for distributed P2P communities. This scheme includes three main parts: a CF-based recommendation-aggregating algorithm, a polling protocol for gathering recommendations, and an identity-based key management system to ensure the confidentiality, integrity and authenticity of trust recommendations.

The application of identity-based cryptography can greatly simplify key management in a distributed environment. We believe that it is a promising technology not only in e-mail systems, but also in trust management systems on P2P networks.

Our polling protocol makes use of identity-based encryption and signatures. Together with a randomization mechanism, users' privacy can be protected while they make recommendations.

We evaluate our CF-based recommendation-aggregating algorithm by simulation experiments. And the results show that it performs pretty well even when malicious peers make the majority.

We haven't test out scheme in a real P2P application. Neither have we simulated all the possible threat models (for example, whitewashing, dynamic personalities, etc.). The traffic overload induced by the polling also need further study. We will examine these problems in a future paper.

References

1. CacheLogic. <http://www.cachelogic.com/research/index.php>
2. Suryanarayana G, Taylor R N. "A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications". Technical report, UC Irvine, (2004)
3. Blaze M, Feigenbaum J, Lacy J. "Decentralized Trust Management". In: Proc. of IEEE Conference on Security and Privacy, Oakland, CA (1996)
4. Blaze M. "The KeyNote Trust Management System (Version 2)". RFC2704. <http://www.crypto.com/papers/rfc2704.txt>, (1999)
5. Khambatti M, Ryu K, Dasgupta P. "Efficient Discovery of Implicitly formed Peer-to-Peer Communities". International Journal of Parallel and Distributed Systems and Networks, 5(4) (2002), pp155-164.
6. Pujol J, Sanguesa R, et al. "Extracting reputation in multi agent systems by means of social network topology". In: Proc. of First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy (2002).
7. Capkun S, Buttyan L, Hubaux JP. "Small worlds in security systems: an analysis of the PGP certificate graph". In: Proc. of ACM New Security paradigms Workshop'02, Virginia (2002), pp28-35
8. Capkun S, Buttyan L, Hubaux JP. "Self-organized public-key management for mobile ad hoc networks". IEEE Transactions on Mobile Computing, 2(1) (2003), pp52-64
9. Richardson M, Agrawal R, and Domingos P. "Trust management for the semantic web". In: Proc. of ISWC 2003, Sanibel Island, Florida (2003)
10. Resnick P, Zeckhauser R, Friedman E, et al. "Reputation Systems: Facilitating Trust in Internet Interactions". Communications of the ACM, 43(12) (2000), pp45-48
11. Aberer K, Despotovic Z. "Managing Trust in a Peer-2-Peer Information System". In: Proc. of CIKM 2001, Atlanta, Georgia (2001), pp310-317

12. Damiani E, Vimercati SDC, Paraboschi S, et al. "Managing and Sharing Servents' Reputations in P2P Systems". *IEEE Transactions on Knowledge and Data Engineering*, 15(4) (2003), pp840-854
13. Kamvar SD, Schlosser M T, Garcia-Molina H. "The EigenTrust Algorithm for Reputation Management in P2P Networks". In: *Proc. of WWW2003, Budapest, Hungary* (2003)
14. Xiong L, Liu L. "PeerTrust: Supporting reputation-based trust in peer-to-peer communities". *IEEE Transactions on Knowledge and Data Engineering*, 16(7) (2004), pp843-857
15. Buchegger S, Boudec J Y. "A Robust Reputation System for P2P and Mobile Ad-hoc Networks". In: *Proc. of P2PEcon 2004, Harvard University*, (2004)
16. Breese JS, Heckerman D, Kadie C. "Empirical analysis of predictive algorithms for collaborative filtering". Technical report, Microsoft Research, (1998)
17. Sarwar BM, Karypis G, Konstan JA, et.al. "Application of dimensionality reduction in recommender system – a case study". In: *Proc. ,ACM WebKDD'2000 Web Mining for E-Commerce Workshop, Boston, MA* (2000)
18. Lai K, Feldman M, Stoica I, et.al. "Incentives for Cooperation in Peer-to-Peer Networks". In: *Proc. of 1st Workshop on Economics of Peer-to-Peer Systems, UC Berkeley* (2003)
19. Shamir A. "Identity-based Cryptosystems and Signature Schemes". In: *Advances in Cryptology – Crypto'84, LNCS 196, Springer-Verlag*, pp47-53 (1984).
20. Boneh D, Franklin M. "Identity-based Encryption from the Weil Pairing". In: *Pro. of Crypto 2001, LNCS 2139, Springer-Verlag*, pp213-229 (2001).
21. Douceur JR. "The Sybil Attack". In: *Proc. of IPTPS'02, (2002)*
22. Marti S, Garcia-Molina H. "Limited Reputation Sharing in P2P Systems". In: *Proc. of ACM EC'04, New York* (2004), pp91-101