Enhancing Collusion Resilience in Reputation Systems

Haiying Shen*, *Senior Member, IEEE*, Yuhua Lin, *Student Member, IEEE*, Karan Sapra, *Student Member, IEEE*, Ze Li

Abstract—Real-world applications, such as peer-to-peer (P2P) networks, e-commerce and social networks, usually employ reputation systems to provide guidance in selecting trustworthy node for high system reliability and security. A reputation system computes and publishes reputation score for each node based on a collection of opinions from others about the node. However, collusion behaviors impair the effectiveness of reputation systems in trustworthy node selection. Though many reputation calculation methods have been proposed to mitigate collusion's influence, little effort has been devoted to specifically tackling collusion. Based on the important collusion behavior characteristics in reputation evaluation and influence on reputation values, we propose a basic collusion detection method to specifically detect suspicious collusion behaviors in pairs. We further optimize the method by reducing the computing overhead. We also propose two pre-processing methods to firstly identify partial reputation raters of a node that are more likely to be colluders before applying the collusion detection method on them, thus reducing the collusion detection overhead. Extensive experimental results show that our proposed methods can significantly enhance the capability of existing reputation systems to detect collusion with low overhead. Also, the pre-processing methods are effective in reducing the collusion detection overhead without affecting the collusion detection accuracy.

Index Terms—Reputation Systems; Trust Systems; Peer-to-Peer Networks; Collusion Detection;

1 Introduction

In real-world applications, such as peer-to-peer (P2P) networks, e-commerce and social networks, nodes or users without preexisting trust relationships interact with each other for different purposes (e.g., resource sharing, transaction and communication). Reputation system is a widely used approach in these applications to enhance collaborations and ensure reliability. In a reputation system, reputation manager(s) computes and publishes global reputation score for each node based on a collection of local reputation ratings from others about the node in order to provide guidance in selecting trustworthy nodes. Despite the effectiveness of the reputation systems, they are generally vulnerable to collusion [1, 2], which impairs their effectiveness in trustworthy node selection. In collusion, two or more malicious nodes conspire to give each other high local reputation values and (or) give all other nodes low local reputation values in order to gain high global reputation [3]. Colluders usually offer low quality of service (QoS) and receive low ratings from nodes outside of the colluding collective [1, 2]. Recent studies [4–7] indicate that collusion behaviors commonly exist in online rating systems.

- ** Corresponding Author. Email: shenh@clemson.edu; Phone: (864) 656 5931; Fax: (864) 656 5910.*
- *Haiying Shen, Yuhua Lin, Karan Sapra and Ze Li are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634.*
  E-mail: {shenh, yuhual, ksapra, zel}@clemson.edu

Current methods that can indirectly deal with collusion focus on how to calculate node reputations to mitigate the influence of collusion. They can be generally classified into two groups: (1) a node calculates others' reputations based on its own experience [8]; and (2) a node includes the feedback of pretrusted nodes and (or) assigns weights to nodes' feedback according to their global reputation [3, 9]. These methods can reduce the impact of collusion when determining node trustworthiness, but they do not specifically tackle collusion in reputation calculation. Some works [10, 11] detect suspicious collusion based on the incorporated online social network in the system. However, these methods are only suitable for the systems that incorporate an online social network. Most peer-to-peer and other distributed applications do not incorporate social networks. Also, building and maintaining such social networks cost high system overhead. Unlike the previous works, in this paper, we directly detect suspicious collusion behaviors based on user interaction history according to the behavior characteristics of colluders in reputation evaluation. Our proposed methods can enhance the capabilities of existing reputation systems to detect and combat collusion. As far as we know, this work is the first that specifically detects suspicious collection behaviors in distributed systems that do not incorporate social networks.

Previous study shows the important behavior characteristics of colluders in reputation evaluation. That is, the suspected colluders (1) gain high global reputations [3], (2) frequently give each other high reputation values and (or) give other nodes low reputation values [3], (3) receive low ratings from other nodes [1, 2], and (4) tend to conspire in bidirectional pairs rather than in a group of more than 2 nodes [1]. According to the behavior characteristics and influence, we propose a basic collusion detection method to detect suspicious collusion behaviors in pairs by directly monitoring user interaction history with low overhead. Based on characteristic (4), we focus on collusion in pairs in this paper and will briefly explain how to extend our method for a collusion group with more than 2 nodes later on though it is rare. In these methods, the reputation manager(s) detects collusion based on collected rating values and rating frequency between nodes. If two high-reputed nodes give high ratings to each other at a high frequency, while receive low ratings from other nodes, the two nodes are suspected colluders. We discuss how to conduct collusion detection in centralized reputation systems using a single reputation manager, and in decentralized reputation systems using a number of reputation managers. We further optimize the method by reducing computing overhead.

We also propose two pre-processing methods to firstly identify partial reputation raters of a node that are more likely to be colluders before applying the collusion detection method on them, thus reducing the collusion detection overhead. One method is based on the random cuts in a graph. A node's graph is formed by its raters and edges from the raters to itself with the reputation rating as the edge weight. If the reputation contribution from a random cut is within a reasonable level, the raters in the cut do not need to be checked for collusion. The other pre-processing method is based on the k-means clustering algorithm. Based on the collusion detection criteria, the k-means clustering algorithm clusters all raters of a node. Then, only the clusters matching the collusion criteria need to be checked. The contribution of this work can be summarized below:

- A basic collusion detection method to detect suspicious collusion behaviors by directly monitoring user interaction history.
- An optimized collusion detection method that reduces computing overhead.
- Random-cut based data pre-processing for collusion detection to reduce computing overhead.
- K-means clustering algorithm based data pre-processing for collusion detection to reduce computing overhead.

We conduct extensive experiments to compare the proposed collusion detection methods with other reputation systems capable of handling collusion and the effectiveness of the two pre-processing methods. Experimental results show that our proposed collusion detection methods significantly enhance the capability of existing reputation systems in deterring collusion with low overhead. Also, the two pre-processing methods reduce the collusion detection overhead while maintain collusion detection accuracy.

In this paper, we use P2P networks as an example for real-world applications that need to use reputation systems to incentivize node cooperation. Our proposed methods can be applied to large-scale distributed systems and applications with large-scale distributed users (e.g., e-commerce and social networks). Today's Internet is still driven by a multitude of large-scale distributed systems that efficiently deliver a variety of data to end users; everything from files via BitTorrent, streaming videos via PPLive, to content via Akamai. Given their reliance on peer-delivery, these systems are vulnerable to the significant impact from selfish nodes. Our work can encourage cooperative node behaviors and create a trustworthy cyber environment, which could have an impact on the real-world applications.

The remainder of this paper is as follow. Section 2 describes our proposed collusion detection methods. Section 3 presents the performance evaluation of the proposed method compared to existing reputation systems and the effectiveness of the pre-processing methods. Section 4 introduces related work in reputation systems and in collusion deterrence. Section 5 concludes the paper with remarks on our future work.

2 Collusion Detection Methods

2.1 Background

Our proposed methods can be built on any reputation system to enhance its capacity to combat collusion. In a centralized reputation system, such as the one in Amazon, a resource manager collects the ratings of all nodes and calculates the reputation values of all nodes. The decentralized reputation systems are more complex. We use EigenTrust [3] as an example to explain how a decentralized reputation system works. EigenTrust forms a number of high-reputed power nodes into a Distributed Hash Table (DHT) for reputation aggregation and calculation. These power nodes are reputation managers. We use ID_i to represent the DHT ID of node n_i, which is the consistent hash value [12] of node n_i's IP address. The reputation manager of reputation ratings on node n_i is the DHT owner of ID_i. A node uses DHT function `Insert`(ID_i, r_i) to send the rating of node n_i to its reputation manager, and uses `Lookup`(ID_i) to query the reputation value of node n_i from its reputation manager. A reputation manager periodically collects the ratings and computes the global reputation values of its responsible nodes. The joins and departures of

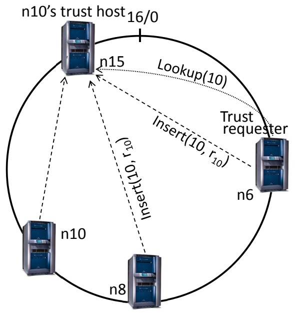


Fig. 1: A distributed reputation system.

reputation managers are handled by the DHT-based mechanism for node joins and departures [13].

Figure 1 presents a 4-node reputation system built on top of the Chord DHT [13] with 4-bit circular hash space. Other nodes report to $n15$ about $n10$'s local reputation by `Insert(10,`r_{10}`)`. Node $n15$ calculates $n10$'s global reputation value. When a node, say $n6$, wants to select a server from several candidates, it queries for the reputation values of the servers. For example, it uses `Lookup(10)` to query $n10$'s reputation value, denoted by R_{10}.

There are many ways to calculate global reputation values of nodes [3]. We use the local reputation calculation method in eBay [14] and EigenTrust [3] as an example in this paper. That is, the local reputation rating for each interaction for a node is -1, 0 and 1. A node's final reputation is the sum of all its received reputation evaluation values. Reputation systems usually specify a reputation threshold T_R. Nodes whose $R \geq T_R$ are considered as trustworthy while nodes with $R < T_R$ are considered as untrustworthy. To apply our method to the reputation systems that use different reputation calculation methods, we regard local reputation rating with $\geq T_R$ as 1, and local reputation rating with $< T_R$ as -1.

Our collusion detection methods are based on the observations of the collusion characteristics in reputation evaluation in the previous study in [1–3] as shown below:

Characteristic 1 (C1). *Collusion leads to high reputation of the colluders [3].*

C2. *Among the high-reputed nodes, colluders receive more low reputations than non-colluders [1, 2].*

C3. *Colluders frequently submit very high ratings for their conspirators [3].*

C4. *Most collusion behaviors are in pairs, and the collusion of multiple colluders in one group rating each other is very rare [1].*

2.2 Basic Collusion Detection Method

Based on the above-mentioned characteristics of collusion in reputation evaluation (C1-C4), we build a collusion model shown in Figure 2 that incorporates all the characteristics. In the collusion, two nodes (C4) frequently (C3) rate high reputation for each other (C3) in order to increase their global reputation values (C1), but offer low-QoS to other nodes and receive low ratings from other nodes (C2). Based on C4, we only focus on collusion in pairs in this paper. Our goal is to enhance the collusion resilience by mitigating the effect from most collusion rather than from all collusion. We will also briefly explain how to extend our method for a collusion group with more than 2 nodes later on though it is rare.

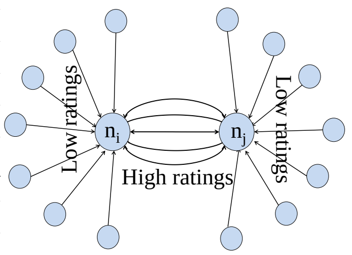


Fig. 2: Collusion model.

TABLE 1: A list of notations in the paper.

Δt	the time period for updating global reputations
N_i	# of all ratings for n_i in Δt
$N_{(i,j)}$	# of ratings from n_j for n_i in Δt
$N_{(i,-j)}$	# of ratings from all nodes except n_j for n_i in Δt
$N^+_{(i,j)}$	# of positive ratings from n_j for n_i in Δt
$N^+_{(i,-j)}$	# of positive ratings from all nodes except n_j for n_i in Δt
$N^-_{(i,j)}$	# of negative ratings from n_j for n_i in Δt
$N^-_{(i,-j)}$	# of negative ratings from all nodes except n_j for n_i in Δt
a	percent of positive ratings in all ratings from n_j for n_i
b	percent of positive ratings from all nodes except n_j for n_i
T_a	threshold of the a value
T_b	threshold of the b value
T_N	threshold of rating frequency
T_R	threshold of reputation to check (un)trustworthy nodes
I	average interactions for node n_i in some time period T
r_{max}	average maximum reputation contribution during a unit period

We define a number of notations shown in Table 1 for a pair of nodes n_i and n_j. We use a to denote the percent of positive ratings in all ratings from n_j for n_i, and use b to denote the percent of positive ratings in all ratings from all nodes except n_j for n_i. We specify a threshold T_a for a and a threshold T_b for b. T_a and T_b can be determined by the historical data of a and b of pairs of nodes with high interaction frequency. For example, in our crawled data [15], for those suspicious colluders found, using the threshold=20 for the average number of transactions of a seller-buyer pair per year, the average a=98.37 and average b=1.63. If we want to reduce the false negatives in collusion detection, we can decrease T_a and increase T_b. On the other hand, if we want to reduce the number of false positives in collusion detection, we can increase T_a and decrease T_b. We use Δt to denote the time period for updating global reputations and use $N_{(i,j)}$ to denote the number of ratings from n_j for n_i in Δt . We also specify rating frequency threshold T_N for $N_{(i,j)}$ to show how frequently n_j rates n_i. For example, based on our trace data [15], T_N=20/year.

In our proposed collusion detection method, resource manager(s) relies on reputation values and frequencies of ratings between a pair of nodes for collusion detection according to the collusion model in Figure 2. We first describe the method in a centralized reputation system, and then describe how the method works in a decentralized reputation system.

Centralized Reputation System. Based on C1, since colluders usually have high reputation values, which is their objective by colluding, and reputation systems regard nodes whose $R \geq T_R$ as high-reputed nodes, we can only check high-reputed nodes to detect colluders. For each node in the system, the centralized reputation manager keeps track of the number of ratings ($N_{(i,j)}$) and the number of positive ratings ($N^+_{(i,j)}$) during Δt of every other node to this node. The reputation manager builds an $n \times n$ matrix, where n is the number of nodes in the network. The matrix records the reputation ratings for nodes whose $R \geq T_R$. If node n_i's reputation value $R_i \geq T_R$, matrix element h_{ij}=$< ID_i, R_i, N^+_{(i,j)}, N_{(i,j)} >$ $(1 \le j \le n)$. Otherwise, h_{ij}=empty.

The manager periodically updates the matrix with its collected information and detects collusion according to the characteristics in the collusion model. In collusion detection, the manager scans each row in the matrix in the top-down manner, and scans elements in each row from the left to the right. For high-reputed node n_i (C1) in a row (non-empty line), the manager checks h_{ij} from every other node n_j (each column) for n_i. If $R_j {\ge} T_R$ and $N_{(i,j)} {\ge} T_N$, which means n_j also has a high reputation (C1) and n_j rates n_i frequently (C3), then $N^+_{(i,j)}$ is further checked. If $N^+_{(i,j)}/N_{(i,j)} {\ge} T_a$, which means a large portion of n_j's ratings are positive (C3), then the ratings from all other nodes except n_j (all other columns) are checked. The manager scans each element in the line of n_i except h_{ij} and calculate the sum of all positive ratings, $N^+_{(i,-j)}$, and the sum of all ratings, $N_{(i,-j)}$. If $N^+_{(i,-j)}/N_{(i,-j)} {<} T_b$, which means a large portion of ratings from other nodes except n_j are negative (C2), then we can conclude that n_i's high reputation is mainly caused by n_j's frequent ratings that are deviated from most others' rating values. As collusion is usually a mutual rating behavior based on C4, then the reputation manager checks whether node n_j is a high-reputed node and whether its high reputation is mainly caused by frequent ratings from n_i that are deviated from most others' rating values. Specifically, the manager finds the row of n_j in the matrix and repeats the same process for n_j to check if n_j's high reputation is also mainly caused by n_i's ratings. If $N_{(j,i)} {\ge} T_N$, $N^+_{(j,i)}/N_{(j,i)} {\ge} T_a$ and $N^+_{(j,-i)}/N_{(j,-i)} {<} T_b$, n_i and n_j are very likely to be colluding. Then, in the periodical global reputation updating, the reputation manager can reduce the reputation values of nodes n_i and n_j to punish their colluding behaviors. During the checking process, after an h_{ij} is checked, the manager marks h_{ij} and h_{ji} to indicate that the two elements no longer need checking.

Decentralized Reputation System. Decentralized reputation systems distribute the role of the centralized resource manager to a number of trustworthy nodes. We use M_i to denote the reputation manager of node n_i. As mentioned, a reputation manager M_i of node n_i keeps track of all ratings of other nodes for n_i. Thus, using the same way as the centralized reputation systems, each reputation manager builds an $\tilde{n} \times n$ matrix, where $\tilde{n}$ is the number of its responsible nodes. For reputation manager M_i, for each of its responsible node n_i with $R_i {\ge} T_R$, if $N_{(i,j)} {>} T_N$, $N^+_{(i,j)}/N_{(i,j)} {\ge} T_a$ and $N^+_{(i,-j)}/N_{(i,-j)} {<} T_b$, n_i is suspected to collude with n_j. Then, if M_i is the reputation manager of node n_j, it uses the same method in the centralized reputation system for the collusion detection. Otherwise, M_i contacts n_j's reputation manager M_j by the DHT function `Insert(j,msg)`. Then, M_j checks R_j and ratings from n_i for n_j. If n_j has high reputation and its reputation is mainly caused by n_i's frequent ratings that deviate from most others' ratings, i.e., $R_j {\ge} T_R$, $N_{(j,i)} {\ge} T_N$, $N^+_{(j,i)}/N_{(j,i)} {\ge} T_a$ and $N^+_{(j,-i)}/N_{(j,-i)} {<} T_b$, M_j sends a positive response to M_i indicating that n_i and n_j are likely to be colluders. Then, in the periodical global reputation updating, reputation managers M_i and M_j can reduce the reputation values of nodes n_i and n_j, respectively, to punish their colluding behaviors. Algorithm 1 shows the pseudo-code of the collusion detection method.

Algorithm 1 Pseudo-code of the collusion detection method in a centralized reputation system.

```
1: /*Return pairs of nodes that are detected as colluders*/
2: for each $n_i$ with reputation value $R_i{\ge}T_R$ do
3:   for each reputation rater $n_j$ of $n_i$ do
4:     if $n_j$'s reputation value $R_j{\ge}T_R$ then
5:       if CheckCollusion($n_i$, $n_j$) && CheckCollusion($n_j$, $n_i$) then
6:         Record $n_i$ and $n_j$ as colluders
7: Function CheckCollusion($n_i$, $n_j$)
8: if rating frequency from $n_i$ for $n_j$ is $\ge T_N$ then
9:   if percent of positive ratings from $n_i$ for $n_j$ is $\ge T_a$ then
10:    if percent of the positive ratings from other nodes is $< T_b$
11:      return true
12: return false
```

We use m to denote the number of high-reputed nodes and n to denote the total number of nodes in the system.

Proposition 2.1: In the collusion detection method, the computation complexity to identify colluders in the P2P system is O(mn^2).

Proof: For each high-reputed node n_i $(1 \le i \le m)$, at most n elements should be checked. For each checking, at most n elements are scanned. Thus, the computation complexity to identify colluders in the system is O(mn^2). □

The collusion detection method can effectively identify the colluders based on the characteristics of collusion behaviors in reputation evaluation. However, in order to calculate $N^+_{(i,-j)}$ and $N_{(i,-j)}$, for each rater n_j, each element in matrix line i should be scanned, generating a high computing cost. Therefore, we propose an optimized collusion detection method that produces much lower computation cost.

2.3 Optimized Collusion Detection Method

In the optimized collusion detection method, a resource manager does not need to scan each element in matrix line i for each $N_{(i,j)}$ $(1 \le j \le n)$. Each manager detects collusion only based on the global reputation value of each of their responsible nodes n_i and frequency of ratings of each of other nodes for n_i. According to the global reputation value calculation method, for a given pair of nodes n_i and n_j, we can get:

$$\begin{cases} b \cdot (N_i - N_{(i,j)}) = N^+_{(i,-j)} \\ (1-b) \cdot (N_i - N_{(i,j)}) = N^-_{(i,-j)} \\ a \cdot N_{(i,j)} = N^+_{(i,j)} \\ (1-a) \cdot N_{(i,j)} = N^-_{(i,j)} \\ N^+_{(i,-j)} + N^+_{(i,j)} - N^-_{(i,-j)} - N^-_{(i,j)} = R_i \end{cases}$$
$$\Rightarrow R_i = 2b(N_i - N_{(i,j)}) + 2aN_{(i,j)} - N_i. \quad (1)$$

Figure 3 visualizes Formula (1) when $1 \geq a \geq T_a$ (i) and $T_b \geq b \geq 0$ (ii). The surface in the figure shows the range of the reputation values of a suspected colluder corresponding to given values of $N_{(i,j)}$ and N_i, given different set of a and b. Conditions (i) and (ii) mean that a large portion of ratings from n_j are positive, whereas a large portion of ratings from other nodes are negative. In this case, n_i's high reputation is mainly caused by node n_j's ratings. Thus, we suspect that nodes n_i and n_j collude with each other. Since $(N_i - N_{(i,j)}) > 0$ and $N_{(i,j)} > 0$, based on Equation (1) and the conditions (i) and (ii), we can derive

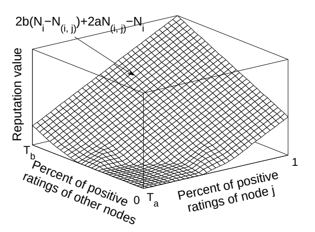


Fig. 3: Reputations of colluders.

$$2T_b(N_i - N_{(i,j)}) + 2N_{(i,j)} - N_i \geq R_i \geq 2T_a N_{(i,j)} - N_i. \quad (2)$$

Thus, if the values of $N_{(i,j)}$, N_i and R_i conform Formula (2), node n_i and n_j are very likely to collude with each other. Therefore, to detect possible collusion, each reputation manager M_i first identifies nodes whose $R \geq T_R$. For each of such node n_i, the manager then checks the rating frequency of each rater of n_i, $N_{(i,j)}$. If $N_{(i,j)} \geq T_N$, the manager then uses Formula (2) to check whether n_i's high reputation is possibly due to its collusion with n_j.

In the case that Formula (2) is satisfied, if M_i is n_j's reputation manager, it checks whether n_j's high reputation is possibly due to its collusion with n_i using the same way. Otherwise, M_i uses `Insert(j,msg)` to contact the reputation manager of n_j. Reputation manager M_j conducts the same process to check if n_j's high reputation is possibly due to its collusion with n_i. If $R_j \geq T_R$, $N_{(j,i)} \geq T_N$ and Formula (2) are also satisfied, n_i and n_j are very likely to be colluding with each other. Algorithm 2 shows the pseudo-code of the optimized collusion detection method.

Proposition 2.2: In the optimized collusion detection method, the computation complexity to identify colluders in the P2P system is O(mn).

Proof: For each high-reputed node n_i ($1 \leq i \leq m$), at most n elements should be checked for collusion detection. Thus, the computation complexity to identify colluders in the system is O(mn). □

Discussion of the Effectiveness of Our Method. Our method is proposed specifically based on the observations of the collusion characteristics in reputation evaluation from previous studies [1–3] listed in Section 2.1. Therefore, if a collusion does not show these characteristics, our method will not be effective in detecting it. However, if collusion nodes do not show these characteristics, their purpose of conducting collusion may not be reached. We will discuss how nodes can avoid showing each characteristic, and the effectiveness of our method in each case in the next paragraph. To achieve high effectiveness of our method, it is important to determine appropriate values for the thresholds (including T_N, T_a and T_b) to reduce both false negatives and false positives. Strict settings will find the colluders that are not so aggressive but generate more false positives, while loose settings will generate more false negatives. Our method aims to enhance collusion resilience by mitigating the effect from most collusions instead of from all collusions. Our experimental results show that our method can reach more than 99% precision rate, recall rate and F1 measurement.

Algorithm 2 Pseudo-code of the optimized collusion detection method in a decentralized reputation system.

```
1: /*Return pairs of nodes that are detected as colluders*/
2: for each responsible node $n_i$ do
3:   if $R_i \geq T_R$ then
4:     for each rater $n_j$ of $n_i$ do
5:       if $n_j$'s rating frequency is no less than $T_N$ ($N_{(i,j)} \geq T_N$)
6:         if $N_{(i,j)}$, $N_i$ and $R_i$ satisfy Formula (2) then
7:           Ask reputation manager of $n_j$ to check the collusion
8:
9: if Receive a positive collusion detection result for $n_i$ and $n_j$ then
10:   return $n_i$ and $n_j$
11: end if
12:
13: /*Receive a request to check the collusion of $n_j$ and its responsible node $n_i$*/
14: if $R_i \geq T_R$ then
15:   if $n_i$'s rating frequency is no less than $T_N$ ($N_{(i,j)} \geq T_N$) then
16:     if $N_{(i,j)}$, $N_i$ and $R_i$ satisfy Formula (2) then
17:       Send positive msg to $M_j$ indicating $n_i$ and $n_j$ are colluders
```

Discussion of Underlying Basis. Our collusion detection method finds nodes whose behaviors match the characteristics of collusion in reputation evaluation (C1-C4). Then, an attacker can use evasion techniques to remain undetected by avoiding exhibiting these characteristics in their behaviors. It is very unlikely for an attacker to avoid C1 because the purpose of collusion exactly is to gain high reputation. If colluders try to avoid C2, they need to try to receive fewer low reputation ratings; the only way for this is to be cooperative. Then, the nodes do not have to be colluders since they can earn high reputation by being cooperative. If colluders try to avoid C3, they must not rate each other frequently with high ratings, which however is the basic behavior of collusion. In order to offset non-colluders' low ratings and to gain high reputation, colluders must rate each other at a relatively high frequency and with a relatively high rating. Therefore, it is also difficult to avoid C3. To avoid C4, colluders can rate each other in a group with more than two nodes and avoid collusion in pairs. It is indicated in [1] that most group collusions are in pairs and groups of three of more are rare in a file sharing system. Thus, in this paper, we focus on collusion in pairs in observing collusion. We can easily extend our methods to observe possible collusion behaviors in a group such as $n_i \rightarrow n_j \rightarrow n_k \rightarrow n_i$, where $\rightarrow$ means highly frequent ratings with high reputation values. Using our method, we can find $n_i \rightarrow n_j$, $n_j \rightarrow n_k$ and $n_k \rightarrow n_i$. Then, if we find these observed behaviors form a loop, it is likely to be a group collusion. We leave the details of the detection of group collusion as our future work.

Discussion of Possible Attacks. The proposed collusion detection method may be abused by attackers to reach their malicious goals. For example, a colluder can submit fake high ratings with high frequency for a normal node, with the purpose of misleading the system into falsely identifying the normal node as a colluder. However, this attack will not succeed under the proposed collusion detection method. As according to Algorithm 1, only when a normal node receives a small portion of positive ratings from other nodes besides the colluder (i.e., $N^+_{(j,-i)}/N_{(j,-i)} < T_b$), this normal node will be identified as a colluder. Also, only when two nodes mutually rate each other with high ratings frequently, they will be considered as potential colluders. Thus, if a normal node behaves good and receives positive ratings from other nodes and it does not rate highly and frequently for the colluders, this attack fails to mislead the system into falsely identifying it as a colluder.

2.4 Pre-processing using Random Cuts

In the optimized collusion detection method, for each node with $R \geq T_R$, to sequentially find nodes with $N_{(i,j)} \geq T_N$ generates high overhead, especially in a large-scale P2P system with thousands or even millions of nodes. To increase the collusion detection scalability with lower overhead, we introduce a random cut based pre-processing method that first identifies the possible colluders before applying the optimized collusion detection method on these nodes.

In the random cut based pre-processing method, each reputation manager draws a graph $G = (V, E)$ based on its received reputation feedback. Figure 4 shows an example of such a graph. In the graph, V consists of the manager's responsible nodes and their raters (i.e., reputation reporting nodes) and E is a set of edges. A uni-directional link e_{ij} is built from node n_j to node n_i if n_j has reported n_i's reputation. Each edge e_{ij} has a weight w_{ij}, which equals the accumulated value of node n_j's reported reputation values for n_i (i.e., reputation contribution from n_j to n_i) during Δt . Then, n_i's reputation equals:

$$R_i = \sum_{k \subset |E_i|} w_{ik}, \tag{3}$$

where E_i denotes all edges pointing to n_i.

Recall that if n_i and n_j are colluding, a large portion of ratings for n_i from n_j are positive, whereas a large portion of ratings from other nodes for n_i are negative. That is, if we cut the edge from n_j to n_i, then R_i drops sharply. For example, in Figure 4, the reputation contributions (link weights) from reporting nodes for n_1 starting at n_{12} in the clockwise direction are $\{1, -1, 26, 4, -2, 0, 1, -1\}$, and those for n_2 starting at n_5 in the clockwise direction are

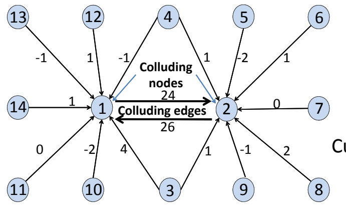


Fig. 4: A graph $G = (V, E)$ showing reputation contribution between different nodes.

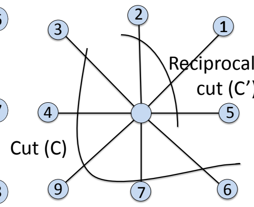


Fig. 5: Graph cut.

$\{-2, 1, 0, 2, -1, 1, 24, 1\}$. Then, $R_1 = 28$ and $R_2 = 26$. When we remove the edge e_{12} and e_{21}, then their reputations drop significantly, i.e., $R_1 = 2$ and $R_2 = 2$. This means n_1 and n_2 contributes greatly to the high reputations of each other and are possibly colluding to boost each other's reputation values.

Based on this phenomenon, for each node n_i with $R_i \geq T_R$, as shown in Figure 5, we cut a randomly selected group of neighboring edges (denoted by C) pointing to n_i. If R_i does not drop sharply, C is unlikely to contain colluder(s), then only the remaining subgraph (i.e., reciprocal cut denoted by C′) needs to be checked. In this way, many nodes do not need to be checked compared to the optimized collusion detection method, thus saving overhead. We introduce the details of this algorithm below.

Node n_i's graph is formed by n_i and other nodes with edges pointing to n_i, denoted by G_i. On G_i, we perform a cut on randomly selected neighboring $p \cdot d_{G_i}$ edges, where p is a percent constant and d_{G_i} is the degree of n_i in graph G_i. We use $R(G)$ to denote the reputation contribution to n_i from nodes in graph G. Then, $R_i(G_i) = R_i(C) + R_i(C')$. We set a threshold $r_t = \eta r_{max}$, where r_{max} is the average maximum reputation contribution during a unit time period from a node in the historical data and η is a percent determined by the collusion detection strictness. If $R_i(C) \leq r_t d_C$, which means that nodes in C gave reasonable ratings with high probability, then we do not need to further check nodes in C. Otherwise, we continually repeat the same random cut operation on C. The reciprocal cut C' is handled in the same manner. As the subgraph C with $R_i(C) > r_t d_C$ is repeatedly cut to a smaller subgraph, finally the possible colluders are located that gave n_i higher rating than the reasonable level. In the worst case, for node n_i, the random cut operation needs to check all neighbors of n_i to find its possible colluders. Thus, the computation complexity of the random cut pre-processing is $O(n d_{max})$, where d_{max} is the maximum degree of all nodes.

Let's use O_i to denote the group of observed possible colluders of n_i. For each node $n_j \in O_i$, n_i's reputation manager checks whether n_j's O_j contains n_i. If yes, the manager uses Formula (2) to determine if n_i and n_j are colluders. If n_j is managed by a different manager, then n_i's manager needs to contact n_j's manager to get n_j's O_j using the method we introduced previously. Therefore, the random cut based pre-processing method helps produce a much smaller scope of suspected colluders before using Formula (2) in the optimized collusion detection method, thus greatly reducing overhead.

Algorithm 3 Pseudo-code of the random cut based pre-processing executed by the reputation manager for a node.

```
Input: G: graph of node $n_i$
Output: O: a vector of suspected colluders of $n_i$
1: Function CheckCollusion(G,O)
2: (C,C′) = GetRandomCut(G) ($\frac{d_C}{d_G} = p$)
3: if $R_i(C) > r_t d_C$ then
4:   if $d_C > 1$ then
5:     CheckCollusion(C,O)
6:   else
7:     Insert(O,C) //insert possible colluder C into vector O
8: if $R_i(C') > r_t d_{C'}$ then
9:   if $d_{C'} > 1$ then
10:    CheckCollusion(C′,O)
11:  else
12:    Insert(O,C′) //insert possible colluder C′ to vector O
```

Algorithm 3 shows the pseudocode for the random cut based pre-processing method executed by a reputation manager. The reputation manager of node n_i (M_i) executes $CheckCollusion(G_i, O_i)$ to obtain the suspected colluders of node n_i in O_i. It is a recursive algorithm and can be divided into two steps. In the first step, M_i obtains a random cut (C) and a reciprocal cut (C') in the graph (line 2). In the second step, M_i checks the possible colluders in C (lines 3-9) and C' (lines 10-16), respectively. When checking the possible colluders, M_i checks if the reputation contribution from C is greater than a reasonable level. If yes, there possibly exist colluders of C and then M_i recursively calls this function with C as the parameter. Otherwise, M_i checks C'. Similarly, if C' contains possible colluders, M_i recursively calls this function with C' as the parameter. Otherwise, no further checking is needed. Each reputation manager executes this algorithm for each of its responsible nodes.

2.5 Pre-processing using K-Means Clustering

We also propose a pre-processing method using k-means clustering to reduce the scope of nodes for collusion checking in order to improve the performance of computationally intensive collusion detection methods in large-scale P2P systems.

In data mining, k-means clustering algorithm [16] is a method of cluster analysis, which aims to partition a set of observations into k cluster such that each observation belongs to the cluster with the nearest mean. K-means clustering algorithm can also work in a multiple-dimensional space.

In Section 2.2, we indicated that for reputation manager M_i, for each of its responsible node n_i with $R_i \geq T_R$, if $R_j \geq T_R$, $N_{(i,j)} > T_N$, $N^+_{(i,j)}/N_{(i,j)} \geq T_a$ and $N^+_{(i,-j)}/N_{(i,-j)} < T_b$, n_i is suspected to collude with n_j. Sequentially checking each rater for a given node n_i produces high overhead. To resolve this problem, reputation manager M_i uses the k-mean clustering algorithm to cluster the raters of n_i based on the above criteria to identify only the clusters of nodes that are likely to be n_i's colluders, and then uses the optimized collusion detection method for further checking.

Thus, we set four dimensions: R_j, $N_{(i,j)}$, $N^+_{(i,j)}/N_{(i,j)}$ and $N^+_{(i,-j)}/N_{(i,-j)}$. That is, each rater of n_i has a 4-dimensional dataset (a tuple of 4 elements). Reputation manager M_i performs k-means clustering and obtains k clusters. The value of k is predefined. It should be set to a value as small as possible in order to reduce the computation time but without compromising the collusion detection performance. We show the performance of different k values in Section 3. Each cluster is represented by the 4-dimensional coordinates of its centroid. M_i then uses the optimized collusion detection method on the clusters of interests (COI), in which most nodes satisfy the aforementioned conditions. We set thresholds αT_R, αT_N, αT_a and βT_b, where $\alpha < 1$ and $\beta > 1$ are parameters determined based on the dispersed degree of the actual criterion values. Suppose a cluster's centroid is (x_1, x_2, x_3, x_4); if $x_1 > \alpha T_R$, $x_2 > \alpha T_N$, $x_3 > \alpha T_a$ and $x_4 < \beta T_b$, this cluster is COI. All nodes belonging to a COI need to be checked, thus reducing all raters of n_i to a smaller group of nodes for collusion checking and hence improving the efficiency of the collusion detection method. The computation complexity of the k-means clustering algorithm is $O({d_{max}}^{4k+1} \log d_{max})$ [16], so the complexity of the pre-processing method using the k-means clustering is $O(n{d_{max}}^{k} \log d_{max})$.

3 Performance Evaluation

Network model. Since this work is the first that specifically detect suspicious collection behaviors in distributed systems that do not incorporate social networks, there is no collusion detection method we can use to compare with our methods. Then, to e-valuate our collusion resilient methods, we measure the performance of EigenTrust [3] and PowerTrust [17] with and without our collusion resilient methods. We consider a generic P2P resource (e.g., file) sharing network in which the nodes are able to issue resource queries and resource offers with different qualities directly between each other. We built an unstructured P2P network with 200 nodes unless otherwise specified. The ratio of the number of individual node's interests to the number of all interest categories is similar to the actual ratio in our crawled trace data from Overstock [11]. Specifically, we assume there are 20 interest categories in the system. The number of interests a node has is randomly chosen from [1,5], and the interests are randomly chosen from the 20 interests. In the P2P network, nodes with the same interest are connected with each other in a cluster. A node with z interests is in z clusters. Each node in the system has maximum 50 units of capacity (i.e., it can handle 50 requests simultaneously per query cycle). For a request of a file of an interest, a node queries all of its neighbors in the cluster of the interest, and

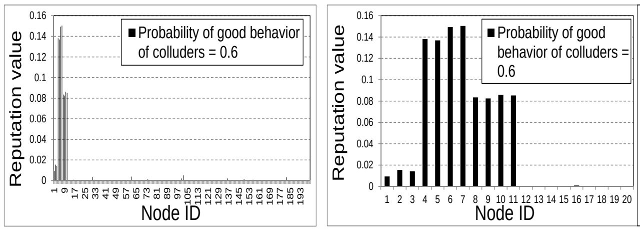


Fig. 6: Reputation distribution in EigenTrust when B=0.6 (Pretrusted node IDs 1-3, colluder IDs: 4-11).

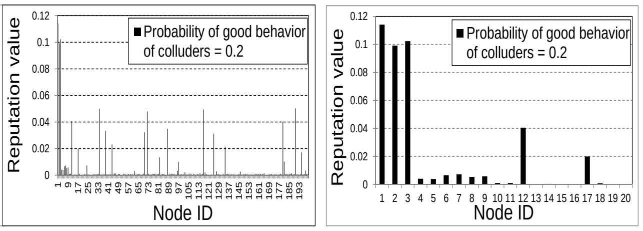


Fig. 7: Reputation distribution in EigenTrust when B=0.2 (Pretrusted node IDs 1-3, colluder IDs: 4-11).

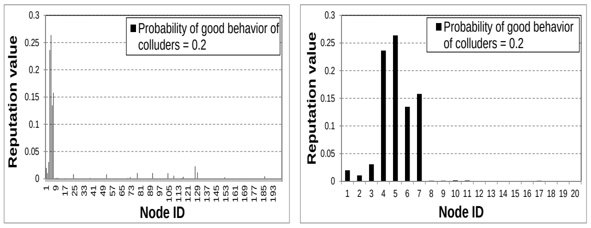


Fig. 8: Reputation distribution in EigenTrust with compromised pretrusted nodes when B=0.2 (Pretrusted node IDs 1-3, colluder IDs: 4-11).

chooses its highest-reputed neighbor with available capacity greater than 0. If a number of options have an identical reputation value, then the client randomly selects a node as a server.

Node model. We consider three types of nodes: pretrusted nodes, colluders and normal nodes unless otherwise indicated. The pretrusted nodes always provide authentic files to the requesters. Normal nodes provide inauthentic files with a default probability of 20% unless otherwise specified. We use B to denote the probability that a colluder offers an authentic file (i.e., good behavior). We randomly chose 3 pretrusted nodes and 8 colluders. In order to show the results more clearly, the IDs of the pretrusted nodes were set to 1, 2 and 3, and the IDs of the colluders were set to 4-11. The weight of the ratings from pretrusted nodes was set to 0.5 in EigenTrust.

Simulation execution. In the simulation process, a node can only send out a file request when it is active. The probability that a node is active is randomly chosen from [0.3, 0.8]. The simulation proceeds in simulation cycles. Each simulation cycle is subdivided into 20 query cycles. In each query cycle, each node issues a query if it is active. Each experiment has 20 simulation cycles. Each experiment ran 5 times and we report the average of the results finally.

Collusion model. In addition to functioning as normal nodes, colluders also mutually rate each other with positive values in order to boost the reputations of each other. We paired up two colluders and let them rate each other 10 times per simulation cycle.

Reputation model. The initial reputation value of each node is 0. Similar to the rating mechanism used in Amazon and Overstock, a client gives a server rating 1 when it receives an authentic file and rating -1 when it receives an inauthentic file. A node's local reputation value equals to the sum of all ratings. Each node updates reputations once after each simulation cycle based on the EigenTrust reputation calculation method. The reputation threshold T_R was set to 0.05, and the rating frequency threshold T_N was to 100 ratings per simulation cycle. Other parameter settings include T_a=90% and T_b=30%. These are empirical values that can better identify uncooperative nodes in our experimental settings. With a collusion-resilient reputation system, we expect to see that the nodes with IDs 4-11 (i.e., colluders) have extremely low reputation values and the normal nodes have comparably higher reputation values.

3.1 Effectiveness of EigenTrust

Figure 6 shows the reputation distribution of all nodes and nodes with IDs in $1-20$ when $B = 60\%$ in EigenTrust. We see that high-reputed nodes are skewed at pretrusted nodes with IDs in $1-3$ and colluders with IDs in $4-11$. We also can see that some normal nodes have relatively higher reputations than others. The reputations of pretrusted nodes are higher than normal nodes, but are significantly lower than colluders. Since the colluders behave well with probability of 60%, they gain a certain number of high ratings though they have 40% probability to receive negative ratings. Furthermore, colluders increase the reputations of each other greatly through collusion, which helps them attract many file requests to further increase their reputations. The results show that EigenTrust cannot detect the collusion behavior and its generated reputations cannot accurately reflect the trustworthiness of nodes.

Figure 7 shows the reputation distribution of all nodes and nodes with IDs in $1-20$ when the colluders offer authentic files at a probability of 20%. From this figure, we can see that some normal nodes have higher reputations while others have lower reputations. This is because at first when all reputation values are 0, nodes randomly choose servers. Since the chosen servers earn reputation, they will have higher probability to be chosen and to further increase their reputations later on. Comparing this figure with Figure 6, we can see that EigenTrust is able to reduce the reputation values of the colluders when $B = 20\%$. Though colluders can try to increase the reputation of each other, the reputations they receive from many other nodes are very low since they only have 20% good behavior probability. Therefore, they are unable to greatly boost the reputation of each other due to the low weight of their ratings. Therefore, EigenTrust can reduce the influence of collusion behaviors in the system when the colluders offer low QoS services at most of the time.

In above experiment, we assume that the pretrusted

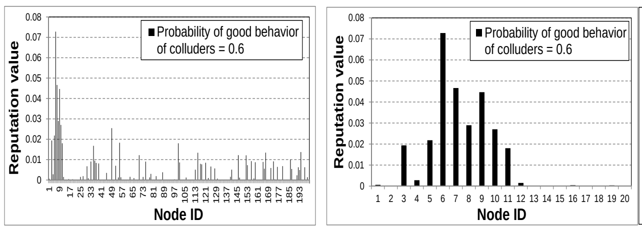


Fig. 9: Reputation distribution in PowerTrust when B=0.6 (Colluder IDs: 4-11).

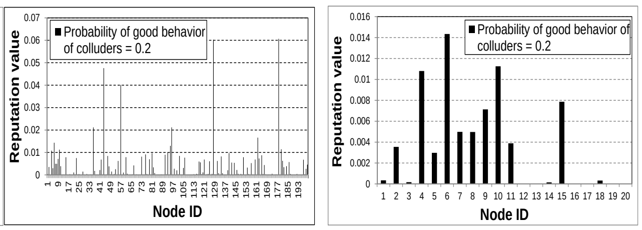


Fig. 10: Reputation distribution in PowerTrust when B=0.2 (Colluder IDs: 4-11).

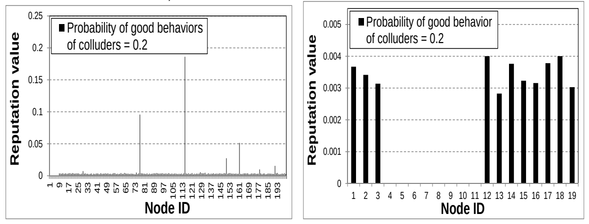


Fig. 11: Reputation distribution in our proposed collusion resilient methods when B=0.2 (Colluder IDs: 4-11).

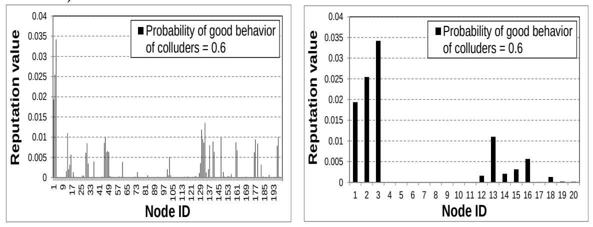


Fig. 12: Reputation distribution in EigenTrust employing our proposed methods when B=0.6 (Pretrusted node IDs 1-3, colluder IDs: 4-11).

nodes are trustable and are not involved in the collusion. In this experiment, we assume that pretrusted node n_1 colludes with node n_4 and pretrusted node n_2 colludes with n_6. Nodes $n_4 - n_{11}$ are still paired up and collude with each other. Colluders offer authenticate files with probability of 0.2. Figure 8 shows the reputation distribution of all node and the first 20 nodes. We see that the high-reputed nodes are skewed among pretrusted nodes and colluders, and the reputations of nodes $n_4 - n_7$ are boosted while the reputation value of nodes $n_8 - n_{11}$ are much lower compared to Figure 7. The reason is that because pretrusted nodes n_1 and n_2 rate highly on node n_4 and node n_6, since EigenTrust assigns more weight to the ratings of pretrusted nodes, n_4 and n_6's reputations are significantly increased. Since node n_4 and node n_6 rate highly on their colluding partners n_5 and n_7, respectively, the reputations of n_5 and n_7 also increase greatly. Because nodes always choose the highest-reputed nodes with available capacity, the reputations of these colluders continually increase and ultimately even exceed the pretrusted nodes' reputations. The result implies that compromising pretrusted nodes will exacerbate the negative impact of collusion on the reputation system, and EigenTrust cannot effectively deal with such malicious behaviors.

3.2 Effectiveness of PowerTrust

PowerTrust can mitigate the influence of collusion. It selects most reputable nodes as power nodes. The global reputation of a node is aggregated from local ratings weighted by the global reputation of the raters. PowerTrust updates the global reputations of nodes with a decaying factor and adds an additional value to the global reputations of power nodes. Thus, PowerTrust significantly improves global reputation accuracy. As PowerTrust does not use pretrusted nodes, we did not deploy pretrusted nodes in the experiments for PowerTrust. Figures 9 shows a whole picture and partial details of the reputation distribution in the system when $B = 60\%$. We can see that colluders gain higher reputation values than others. Since the colluders have 60% probability to behave well, they gain a certain number of high ratings from authentic transactions. The results show that PowerTrust is not effective in deterring the collusion behavior and its generated reputations cannot accurately reflect the trustworthiness of nodes.

Figure 10 shows the reputation distribution of all nodes and the first 20 nodes when $B = 20\%$. The distribution shows a similar trend as that in Figure 9 due to the same reason. The colluders in Figure 10 achieve lower reputations than those in Figure 9. As colluders have 80% probability to offer fraudulent transactions, which damages their reputation.

3.3 Effectiveness of Our Proposed Methods

Our Proposed Methods. We then test the collusion detection effectiveness of our proposed basic collusion detection method (denoted by *Unoptimized*) and optimized collusion detection method (denoted by *Optimized*). In this experiment, there are no pretrusted nodes. After the methods detect the colluders, they set their reputations to 0. Since *Unoptimized* and *Optimized* are only used for collusion detection and they use the same reputation value calculation, their final reputation distributions of the nodes are the same. Thus we only use one figure to show the results of both *Unoptimized* and *Optimized*. We show the results of both *Unoptimized* and *Optimized* in Figures 11 with the probability of good behavior of colluders equals to 20%. They shows that both *Unoptimized* and *Optimized* can detect all colluders, which indicates that the methods can effectively detect the colluders based on their contact frequency and reputation values.

EigenTrust with Our Proposed Methods. Next, we test how *Unoptimized* and *Optimized* can help enhance EigenTrust's capability in detecting collusion. Since *Unoptimized* and *Optimized* generate the same results in collusion detection, we use *Optimized* to represent both and use EigenTrust+*Optimized* to denote Eigen-

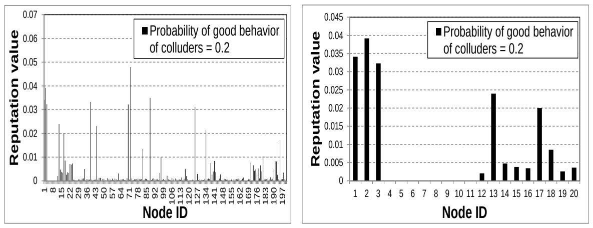


Fig. 13: Reputation distribution in EigenTrust employing our proposed methods when B=0.2 (Pretrusted node IDs 1-3, colluder IDs: 4-11).

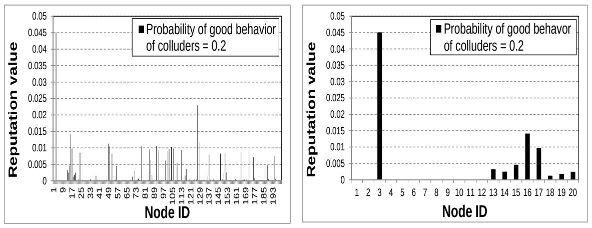


Fig. 14: Reputation distribution in EigenTrust employing our proposed methods with compromised pretrusted nodes when B=0.2 (Pretrusted node IDs 1-3, colluder IDs: 4-11).

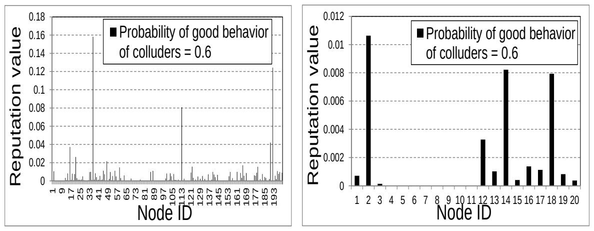


Fig. 15: Reputation distribution in PowerTrust employing our proposed methods when B=0.6 (Colluder IDs: 4-11).

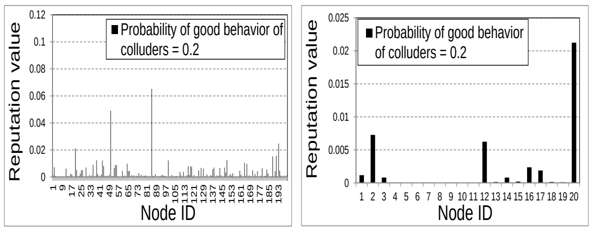


Fig. 16: Reputation distribution in PowerTrust employing our proposed methods when B=0.2 (Colluder IDs: 4-11).

Trust employing *Optimized*. Figure 12shows the reputation distribution of all nodes and nodes IDs in $1-20$ in EigenTrust+*Optimized* when the colluders offer authentic files with probability of 60%. Comparing to Figure 6, we find that EigenTrust+*Optimized* leads to increased average reputation values for many normal nodes. The results demonstrate the effectiveness of *Optimized* in detecting collusion.

Figure 13 shows the reputation distribution of all nodes and the first 20 nodes in EigenTrust+*Optimized* when the colluders offer authentic files with probability of 20%. Comparing to Figure 7, we observe that EigenTrust+*Optimized* increases the reputations of normal nodes by a greater amount than EigenTrust. The results confirm the effectiveness of *Optimized* in collusion detection based on rating patterns.

Figure 14 shows the reputation distribution of the nodes in EigenTrust+*Optimized* in the same scenario as Figure 8. Comparing Figure 14 and Figure 8, we see that EigenTrust+*Optimized* increases the reputations of normal nodes in EigenTrust. By compromising pretrusted nodes, colluders can receive much higher reputations than pretrusted nodes in EigenTrust. While in EigenTrust+*Optimized*, both colluders and compromised pretrusted nodes receive 0 reputation values. Since the pretrusted node with ID=3 does not involve in the collusion, its reputation value is still high. In conclusion, our proposed *Unoptimized* and *Optimized* collusion detection methods can greatly enhance the collusion detection capability of EigenTrust.

PowerTrust with Our Proposed Methods. Figure 15 shows the reputation distribution in PowerTrust+*Optimized* when $B = 60\%$. It shows that PowerTrust+*Optimized* can reduce the negative impact of collusion by reducing the high reputations of colluders with IDs $4-11$ to 0. Figure 16 shows the reputation distribution of the system in PowerTrust+*Optimized* when $B = 20\%$. Comparing to Figure 10, we observe that PowerTrust+*Optimized* deter collusion by reducing colluders' reputation values to 0. The results confirm the effectiveness of our proposed methods in collusion detection based on the rating patterns.

Comparison between Different Methods. We then use precision rate and recall rate to show the effectiveness of our proposed method in collusion detection. EigenTrust and PowerTrust do not have collusion detection mechanisms, so we used two thresholds and consider nodes with reputations below the threshold as colluders. One threshold equals the average reputation value of all nodes (i.e., 0.005) and the other threshold equals the highest reputation among the real colluders. EigenTrust using the two thresholds are denoted by EigenTrust-0.005 and EigenTrust-highest, respectively. The same to PowerTrust. We use EigenTrust /w and PowerTrust w/ to represent EigenTrust and PowerTrust with our proposed method. As in [18], we randomly chose a certain percent of nodes as colluders and varied the percent value in testing. Colluders offer authenticate files with probability of 20%.

Table 2 shows the precision rate, recall rate and F_1 measure in colluder detection in different methods with different percents of colluders. We see that when the percent of colluders equals to 10%, EigenTrust-0.005 and PowerTrust-0.005 perform the worst among different methods, with precision rates of 11.2% and 10.4%, respectively. Though EigenTrust-highest and PowerTrust-highest improve the precision rates, they are still very low (lower than 14%). With our proposed collusion detection method, EigenTrust and PowerTrust achieve precision rates as high as around 90%. We also see that as the percent of colluders increases, our method makes EigenTrust and PowerTrust maintain their precision rates at approximately 90%, and the precision rates of other methods increase since more colluders have reputations below the threshold. These results confirm the effectiveness of our method in colluder detection.

EigenTrust-0.005 and PowerTrust-0.005 yield the

TABLE 2: Precision/Recall/F_1 measure of collusion detection.

% of colluders	Precision			Recall			F_1 measure		
	10	20	30	10	20	30	10	20	30
EigenTrust-highest	13.7	24.6	33.4	100	100	100	24.1	39.5	50.1
EigenTrust-0.005	1.9	9.6	11.2	7.4	25.1	27.3	3	13.9	15.9
EigenTrust w/	99.1	98.9	99.3	100	92.8	83.3	99.5	95.8	90.6
PowerTrust-highest	10.2	21.4	32.7	100	100	100	18.5	35.3	49.3
PowerTrust-0.005	1.5	7.3	10.4	5.3	24.7	26.8	2.3	11.3	14.9
PowerTrust w/	99.2	99.3	99.5	99.4	93.3	82.8	99.3	96.2	90.4

lowest recall rates since colluders can get high reputations even with the strategies of EigenTrust and PowerTrust. EigenTrust-highest and PowerTrust-highest exhibit the best recall rates because we chose the highest reputation value of colluders as the threshold, but at the cost of low precision rates. Our proposed method can increase the recall rates of EigenTrust and PowerTrust significantly. We see that as the percent of colluders increases, the recall rates of EigenTrust-highest and PowerTrust-highest keep at 100% due to our selected threshold, those of EigenTrust-0.005 and PowerTrust-0.005 increase since more colluders increase the probability of a colluder having reputation lower than the threshold. The recall rates of EigenTrust w/ and PowerTrust w/ show a slight decrease. When there are more colluders in the system, the number of undetected colluders increases faster than the number of detected colluders in our method because the percent of colluders that satisfy the collusion checking conditions decreases. As a result, the recall rates with our method decreases.

F_1 measure is the harmonic mean of precision and recall. We see that both EigenTrust w/ and PowerTrust w/ generate much higher F_1 scores than other methods. The result indicates that our proposed method is effective in identifying colluders.

3.4 Performance Comparison

We use *Random cut*, *K-means* and *K-means+Random cut* to denote the optimized collusion detection method with the pre-processing using random cuts, with the pre-processing using k-means clustering, and with the pre-processing using k-means clustering first and then using random cuts. We set p to 0.3 and set r_t to 0.4 in our random cut method.

Figure 17 shows the percent of the file requests sent to the colluders in the total number of requests in the system versus the number of colluders in the system in *Unoptimized*, *Optimized*, *EigenTrust*, *Random cut*, *K-means* and *K-means+Random cut*. In this experiment, the setting of EigenTrust is identical to Figure 7. If the computed global reputation values can more accurately reflect the actual behavior of nodes, the number of requests sent to the colluders should be smaller. Since EigenTrust is not effective at collusion detection, colluders have high reputations, they receive many file requests from other nodes. The results indicate that our methods are more effective in thwarting collusion, and our proposed pre-processing methods do not compromise accuracy of collusion detection.

We plot Figure 18 without EigenTrust to see the differences between all our methods. We see that *K-means*, *Random cut* and *K-means+Random cut* generate a slightly higher percentage of requests sent to colluders. As *K-means* partitions all raters of a node to different clusters, there is a possibility that a colluder lies within a non-COI cluster. In *Random cut*, a colluding edge might be missed in checking if r_t value is large.

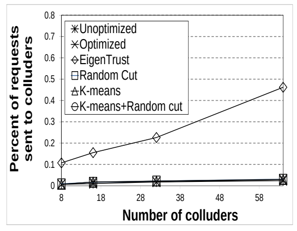


Fig. 17: Effectiveness of thwarting collusion (with EigenTrust).

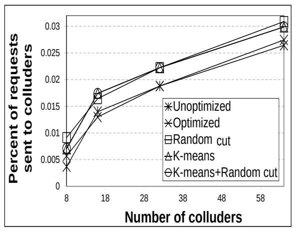


Fig. 18: Effectiveness of thwarting collusion (without EigenTrust).

Figure 19 shows the percent of file requests sent to colluders versus the number of nodes in the system. The total number of colluders was set to 64 in this experiment. Both X and Y axes are in a logarithmic scale. We see a decline in the percentage of requests sent to colluders in all methods when there are more nodes in the system. EigenTrust is not as effective as our methods in thwarting collusion because even when the number of the nodes in the system increases, the colluders still have high reputation and thus a large number of request are sent to them. We also see that *K-means* has higher percent and it decreases more slowly than our other methods as the number of nodes in the system increases due to the reason that a colluder may lie within a non-COI cluster as explained previously.

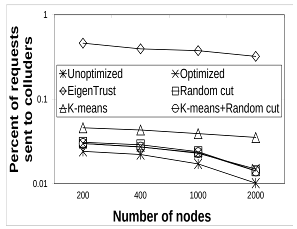


Fig. 19: Effectiveness of thwarting collusion with different network size.

We define *operation cost* as the number of computer cycles for thwarting collusion. The operation cost of EigenTrust includes the cost for calculating all global reputations for all nodes, and the operation cost of our methods includes the cost for information analysis and computation, and the pre-processing. Figure 20 shows the operation costs of *Unoptimized*, *Optimized*, *EigenTrust*, *Random cut*, *K-means* and *K-means+Random cut*. The figure shows that *Unoptimized* generates significantly higher operation cost than others. This is because for each high-reputed node in the system, *Unoptimized* needs to scan all raters for rating values and frequency for each rater. The operation cost in EigenTrust is caused by the recursive matrix calculation, which is determined by the number of the nodes rather than the number of colluders in the system. Hence, the operation cost of EigenTrust is constant as the number of colluders in the system increases.

In order to see the differences between other methods, we draw Figure 21 without *Unoptimized*. We see that EigenTrust produces higher operation cost

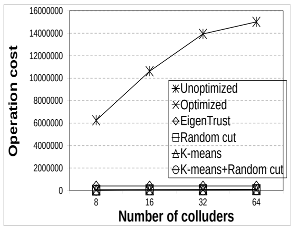


Fig. 20: Operation cost of thwarting collusion.

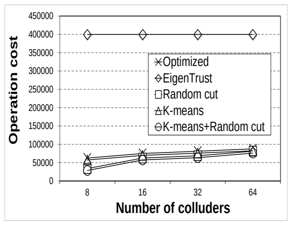


Fig. 21: Operation cost of thwarting collusion (without *Unoptimized*).

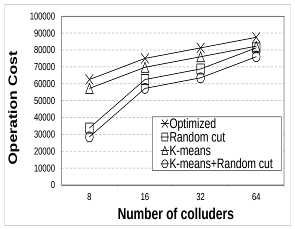


Fig. 22: Operation cost of thwarting collusion (without *Unoptimized* or EigenTrust).

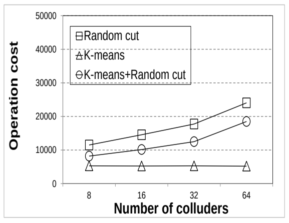


Fig. 23: Operation cost for the pre-processing step.

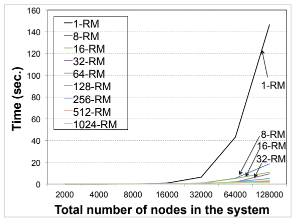
than *Optimized* and other methods. The operation cost of *Optimized* is very low because there is no need to scan the ratings from a node's raters. The only operation cost of *Optimized* is caused by checking the high contacting frequency between high-reputed nodes and collusion inference. We see that *Random cut*, *K-means* and *K-means+Random cut* produce similar operation cost as *Optimized*, which indicates that the additional cost of pre-processing does not exceed the collusion detection cost saved by pre-processing. We plot Figure 22 to show the operation cost of *Optimized*, *Random cut*, *K-means* and *K-means+Random cut*. We notice that *K-means* produces lower operation cost than *Optimized* due to the reason that *K-means* clustering algorithm reduces the scope of suspected colluders for collusion checking.

3.5 Performance of Pre-processing

We plot Figure 23 to show the operation cost for only the pre-processing step (without collusion detection cost) in *K-means*, *Random cut* and *Random cut+K-means*. Compared to Figure 22, we see that the pre-processing step in different pre-processing methods generate very low cost. These additional costs are lower than the cost saved from collusion detection by these pre-processing methods; that is, they bring about more benefits than cost. We also see that as the number of colluders in the system increases, the operation cost of *Random cut* and *Random cut+K-means* increase steadily while that of *K-means* remains relatively constant. This is because the cost of *K-means* is independent of the number of colluders since it only creates clusters for the given number of nodes, while more colluders make *Random cut* to have more recursive operations.

Next, we present the efficiency of the pre-processing method with different number of reputation managers and network sizes. We use the k-means clustering as an example. In order to show the efficiency measured by real processing time, we used the Palmetto Supercomputer [19] in Clemson University for experiments. The Palmetto Supercomputer has 15,120 computing cores and we used each core to simulate a node or a reputation manager.

Figure 24 shows the computation time for pre-processing using k-means clustering with varying number of reputation managers and nodes in the system. Because we aim to test the efficiency of the k-means clustering, we assume that each node has $R \geq T_R$ so that the clustering is on all nodes. In



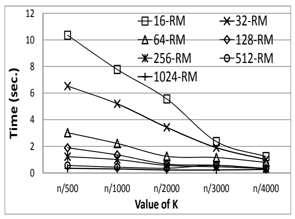


Fig. 24: Computation time for k-means clustering vs. the number of nodes.

Fig. 25: Computation time for k-means clustering vs. the value of k.

the figure, "m-RM" means there are m reputation managers in the system. "1-RM" represents the centralized reputation system. In the system, each node has received a reputation rating from every other node, and k was set to $n/1000$ (n is the number of nodes in the system) in order to keep the number of clusters to be the same for different network sizes. From the figure, we see that the run time of the k-means clustering increases as the number of nodes in the system increases or as the number of reputation managers decreases. The time increases exponentially in "1-RM". As more nodes in the system generate more ratings, the k-means clustering needs more time to create clusters. Also, we observe that the run time growth in a fully decentralized system is much slower than that of the centralized method. Therefore, the decentralized reputation system is more suitable than the centralized method in large-scale P2P systems. Figure 25 shows the computation time for the system with 128,000 nodes and different number of RMs when k is varied from $n/500$ to $n/4000$. We see that computation time decreases when k decreases due to the same reason as in Figure 24. Overall, the computation time is short especially when there are no less than 128 RMs, which indicates that the k-means clustering based pre-processing is suitable for large-scale P2P systems.

4 Related Work

Recent studies show that collusion behaviors commonly exist in online rating systems [4–7]. Allahbakhsh *et al.* [4] studied the impact of collusion attacks using the logs of Amazon online rating system, and showed that current rating systems are vulnerable to collusion attacks. You *et al.* [5] found that sellers in consumer-to-consumer (C2C) markets use collusion attacks to manipulate their reputations by studying a dataset from Taobao, the largest C2C market in China. You *et al.* [6] studied the damage of collusion attacks to crowdsourcing systems by analyzing anonymized product ratings obtained from a large e-commerce organization. They found that colluders are able to boost the ratings of their products by cooperating with others. Allahbakhsh *et al.* [7] showed that users can give unfair evaluations to products by studying the MovieLens dataset from a famous online movie rating systems. Therefore, it is important to enhancing collusion resilience in reputation rating systems.

Many previous reputation systems focus on how to accurately calculate the global reputation value of nodes. PowerTrust [17] dynamically selects a small number of most reputable nodes using a distributed ranking mechanism. The work in [8] focuses on first-hand reputation calculation, in which a node only believes its own observations. Li *et al.* [20] used a time attenuation function to calculate local trust rating, and then applied a scalable feedback aggregating overlay to compute trustworthiness of nodes. Koutrouli *et al.* [21] proposed a credit-based recommendation exchange mechanism. Ammar *et al.* [22] studied the problems to rank items based on partial user inputs. Yao *et al.* [23] indicated that previous trust inference approaches largely ignore other important properties and proposed a multi-aspect trust inference model. Akavipat *et al.* [24] presented a framework for enhancing lookups in redundant DHTs by tracking how well other nodes service lookup requests. Costa and Furtado *et al.* [25] presented a reputation-based task scheduling strategy for distributed database systems.

However, existing reputation systems are vulnerable to collusion [1]. Current methods that can indirectly deal with collusion focus on how to calculate node reputations to mitigate the influence of collusion. EigenTrust [3] breaks collusion collectives by using the feedback of pretrusted nodes. Fan *et al.* [26] recommended reputation value to evaluate the resource service behavior, and recommending reputation value to evaluate the trust recommendation behavior of nodes. RCA [27] uses tamper-evident logs for all messages to discover all misreporting and protocol violations. Some works [28, 29] calculate the similarity among users who provide ratings for suspicious products (that are suspected to receive abnormal ratings) to detect potential colluders.

Some works leverage social networks to combat collusion. SocialTrust [11] adaptively adjusts the weight of ratings based on the social distance and interest relationship between nodes to combat collusion. In Sorcery [10], a client labels a content provider as a colluder if the provider's votings on some items do not agree with that of the its friends. SocialLink [30] uses a weighted transaction network to manage the trust among non-friends, and it can thwart collusion because nodes need to upload files in order to receive files from non-friends.

Sybil attacks share similar features as collusion and many works use the social network clustering feature to handle the sybil attacks. The works in [31, 32] use sybil region and random walk strategy to determine whether a node is potentially a sybil attack node. The works in [33–38] restrict the number of attack edges between honest regions (i.e., the regions including all non-malicious nodes) and sybil regions (i.e., the regions with all sybil nodes). The work in [39] uses landmark routing-based techniques to efficiently approximate credit payments over large networks to reduce the overhead on computationally expensive network analysis in the previous sybil tolerance systems.

Unlike the previous works that either adjust the global reputation calculation method to mitigate the influence of collusion or rely on social networks to detect collusion which is not suitable for systems without social networks or brings extra overhead, our proposed collusion-resilient methods directly detect suspicious collusion behaviors based on the behavior characteristics of colluders in reputation evaluation with low overhead.

5 Conclusion

Though many reputation systems try to reduce the influence of collusion on calculating reputation values, there are few works that specifically tackle collusion. According to collusion behavior characteristics in reputation evaluation, we propose collusion detection methods to thwart collusion behavior and further optimize the method by reducing the computing cost. We also propose two pre-processing methods to shrink the scope of nodes to apply the collusion detection methods in order to reduce the overhead, which is especially important in large-scale networks. Experimental results show the higher capacity of the methods in combating collusion in comparison with the EigenTrust and PowerTrust reputation systems, and the effectiveness of our proposed pre-processing methods in reducing overhead. In our future work, we will investigate how to detect a collusion collective having more than two nodes, and consider other collusion patterns (i.e, repetition, spam account and traffic concentration) [1] in collusion detection.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, and Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proc. of ICPP'12 [15].

References

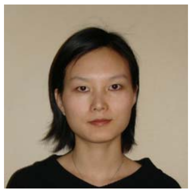
[1] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li, "An empirical study of collusion behavior in the maze P2P file-sharing system," in *Proc. of ICDCS*, 2007.

[2] S. Zhao and V. Lo, "Result verification and trust-based scheduling in open peer-to-peer sharing systems," in *Proc. of P2P*, 2005.

[3] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in P2P networks," in *Proc. of WWW*, 2003.

[4] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S. Beheshti, N. Foo, and E. Bertino, "Detecting, representing and querying collusion in online rating systems," *CoRR, abs/1211.0963*, 2012.

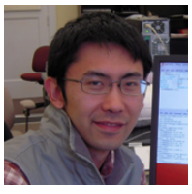
[5] W. You, L. Liu, M. Xia, and C. Lv, "Reputation inflation detection in a chinese c2c market," *Electronic Commerce Research and Applications*, vol. 10, no. 5, pp. 510–519, 2011.
[6] A. KhudaBukhsh, J. Carbonell, and P. Jansen, "Detecting non-adversarial collusion in crowdsourcing," in *Proc. of HCOOMP*, 2014.
[7] M. Allahbakhsh and A. Ignjatovic, "An iterative method for calculating robust rating scores," *TPDS*, vol. 26, no. 2, pp. 340–350, 2015.
[8] A. A. Selcuk, E. Uzun, and M. R. Pariente, "A reputation based trust management system for P2P networks," *International Journal of Network Security*, 2008.
[9] K. Walsh and E. G. Sirer., "Experience with a distributed object reputation system for peer-to-peer filesharing," in *Proc. of NSDI*, 2006.
[10] E. Zhai, R. Chen, Z. Cai, L. Zhang, E. K. Lua, H. Sun, S. Qing, L. Tang, and Z. Chen, "Sorcery: Could we make P2P content sharing systems robust to deceivers?" in *Proc. of P2P*, 2009.
[11] Z. Li, H. Shen, and K. Sapra, "Leveraging social networks to combat collusion in reputation systems for peer-to-peer networks," *IEEE Transactions on Computers (TC)*, vol. 62, no. 9, pp. 1745 – 1759, 2013.
[12] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and P. R., "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," in *Proc. of STOC*, 1997.
[13] I. Stoica, R. Morris, D. Liben-Nowellz, D. R. Kargerz, M. F. Kaashoekz, F. Dabekz, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE TON*, 2003.
[14] "ebay," http://www.ebay.com [accessed in August 2015].
[15] Z. Li, H. Shen, and K. Sapra, "Collusion detection in reputation systems for peer-to-peer networks," in *Proc. of ICPP*, 2012.
[16] M. R. Anderberg, *Cluster Analysis for Applications*. Academic Press, 1973.
[17] R. Zhou and K. Hwang, "Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing." *IEEE TPDS*, 2007.
[18] L. Canon, E. Jeannot, and J. Weissman, "A Scheduling and Certification Algorithm for Defeating Collusion in Desktop Grids," in *Proc. of ICDCS*, 2011.
[19] "Palmetto Supercomputer - Clemson University," http://citi.clemson.edu/palmetto/[accessed in August 2015].
[20] X. Li, F. Zhou, and X. Yang, "Scalable Feedback Aggregating (SFA) Overlay for Large-Scale P2P Trust Management," *TPDS*, 2012.
[21] E. Koutrouli and A. Tsalgatidou, "Credible recommendation exchange mechanism for P2P reputation systems," in *Proc. of SAC*, 2013.
[22] A. Ammar and D. Shah, "Efficient rank aggregation using partial data," in *Proc. of SIGMETRICS*, 2012.
[23] Y. Yao, H. Tong, X. Yan, F. Xu, and J. Lu, "Matri: a multi-aspect and transitive trust inference model," in *Proc. of WWW*, 2013.
[24] R. Akavipat, M. Al-Ameen, A. Kapadia, Z. Rahman, R. Schlegel, and M. Wright, "ReDS: A Framework for Reputation-Enhanced DHTs," *TPDS*, vol. 1, no. 99, pp. 1–10, 2014.
[25] R. Costa and P. Furtado, "Elections and reputation for high dependability and performance in distributed workload execution," *TPDS*, vol. 1, no. 99, pp. 1–10, 2014.
[26] X. Fan, M. Li, J. Ma, Y. Ren, H. Zhao, and Z. Su, "Behavior-based reputation management in P2P file-sharing networks," *J. Comput. Syst. Sci.*, 2012.
[27] P. Aditya, M. Zhao, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, and B. Wishon, "Reliable Client Accounting for P2P-Infrastructure Hybrids," in *Proc. of NSDI*, 2012.
[28] Y. Liu, Y. Yang, and Y. Sun, "Detection of collusion behaviors in online reputation systems," in *Proc. of ACSSC*, 2008.
[29] Y. Yang, Y. Sun, S. Kay, and Q. Yang, "Defending online reputation systems against collaborative unfair raters through signal modeling and trust," in *Proc. of SAC*, 2009.
[30] K. Chen, G. Liu, H. Shen, and F. Qi, "SocialLink: Utilizing Social Network and Transaction Links for Effective Trust Management in P2P File Sharing Systems," in *Proc. of P2P*, 2015.
[31] A. Mohaisen, N. Hopper, and Y. Kim, "Keep Your Friends Close: Incorporating Trust into Social Network-based Sybil Defenses," in *Proc. of INFOCOM*, 2011.
[32] W. Wei, X. Fengyuan, C. Chiu, and L. Qun, "SybilDefender: Defend Against Sybil Attacks in Large Social Networks," in *Proc. of INFOCOM*, 2012.
[33] T. Nguyen, L. Jinyang, S. Lakshminarayanan, and S. Chow, "Optimal Sybil-Resilient Peer Admission Control," in *Proc. of INFOCOM*, 2011.
[34] J. L. N. Tran, B. Min and L. Subramanian, "Sybil-Resilient Online Content Voting," in *Proc. of NSDI*, 2009.
[35] H. Yu, M. Kaminsky, and A. D. Flaxman, "Sybilguard: Defending against sybil attacks via social networks," in *Proc. of SIGCOMM*, 2006.
[36] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *Proc. of IEEE SP*, 2008.
[37] W. Wei, F. Xu, C. Tan, and Q. Li, "Sybildefender: Defend against sybil attacks in large social networks," in *Proc. of INFOCOM*, 2012.
[38] L. Shi, S. Yu, W. Lou, and Y. Hou, "Sybilshield: An agent-aided social network-based sybil defense among multiple communities," in *Proc. of INFOCOM*, 2013.
[39] B. Viswanath, M. Mondal, and K. Gummadi, "Canal: Scaling social network-based sybil tolerance schemes," in *Proc. of EuroSys*, 2012.



Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the Department of Electrical and Computer Engineering at Clemson University. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and cloud computing. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE, and a member of the ACM.

Yuhua Lin received both his BS degree and MS degree in Computer Engineering from Sun Yat-sen University, China in 2009 and 2012 respectively. He is currently a Ph.D student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include social networks and reputation systems.

Karan Sapara received the BS degree in computer engineering from Clemson University, in 2011, and is currently working toward the PhD degree in the Department of Electrical and Computer Engineering at Clemson University. His research interests include P2P networks, distributed and parallel computer systems, and cloud computing. He is a student member of the IEEE.



Ze Li received the BS degree in Electronics and Information Engineering from Huazhong University of Science and Technology, China in 2007, and the Ph.D. degree in Computer Engineering from Clemson University. His research interests include distributed networks, with an emphasis on peer-to-peer and content delivery networks. He is currently a data scientist in the MicroStrategy Incorporation.