A Social Network based Reputation System for Cooperative P2P File Sharing

Kang Chen, *Student Member, IEEE*, Haiying Shen*, *Senior Member, IEEE*, Karan Sapra and Guoxin Liu

Abstract—Current reputation systems for peer-to-peer (P2P) file sharing networks suffer from high overhead on reputation querying. Also, purely relying on a threshold to detect malicious nodes may make a high-reputed node be reluctant to further increase its reputation in these reputation systems. On the other side, the social network concept of ”friendship foster cooperation” can be utilized to alleviate the high overhead in reputation systems. However, the limited number of friends limits the availability of file resources in these approaches.To overcome the drawbacks, we propose a social network based reputation system, namely SocialTrust, that synergistically leverages the social network connections and traditional credit based reputation system to provide efficient reputation management for P2P file sharing. In SocialTrust, each node favors friends for service transactions, which are resulted from both real life acquaintance and online partnership established between high-reputed and frequently-interacted nodes. When no friends are available for a request, a node chooses the server with the highest reputation. The benefits of friendship and partnership on file sharing and cost saving encourage nodes to be continuously cooperative. Further, SocialTrust considers the number of friends/partners and reputation of a node in reputation rewarding/punishment in order to realize accurate reputation evaluation. SocialTrust can also prevent certain attacks such as deny of service and collusion. Extensive trace-driven simulation demonstrates the effectiveness of SocialTrust.

Index Terms—Social networks, Reputation system, Peer-to-peer, File sharing

1 Introduction

Due to the open nature of the peer-to-peer (P2P) environment, P2P file sharing systems are prone to have selfish and misbehaving nodes. Selfish nodes are not cooperative in providing files, but still would like other nodes to comply to their requests [1], [2]. Misbehaving nodes can distribute tampered files, corrupted files or files with malicious code into the system, which could be further spread by unsuspecting users. For example, 85% of Gnutella users are selfish nodes sharing no files, and 45% of files downloaded through the Kazaa file sharing application contain malicious code. Therefore, incentives are needed to encourage cooperation in P2P networks. Reputation system, as a cooperation incentive method, has been widely studied in recent years [3]–[8]. In a reputation system, a node's reputation is built based on a collection of feedbacks from other nodes. A pre-defined reputation threshold is used to classify nodes to reputed or selfish nodes. However, a clever node can sustain in the system by maintaining its reputation just above the threshold and take this advantage for uncooperative behaviors. Further, frequent reputation querying can easily overload the reputation center, leading to degraded service quality in P2P systems.

Recently, emerging P2P file sharing systems have been proposed to exploit social network connections to enhance service cooperation [9]–[12] or malicious node detection [13] by leveraging the social property of “friendship fosters cooperation” [14]. Naturally, such an idea can alleviate the necessity of reputation querying and reduce the load on reputation centers. However, the friendship network of a node usually only consists of a small part of the entire P2P network, which means that a client may not be able to find a server among its friends. Thus, these social network based approaches, if imported into P2P file sharing systems, limit the objective of widely sharing of files.

In this paper, we propose a novel reputation system, namely SocialTrust, that synergistically exploits both traditional credit based reputation systems and social networks to overcome their individual shortcomings with three main components:

(1) *Social networks (friend network and partner network)*. Each node maintains two lists of mutual-trusted nodes, friend-list containing friends from real life (i.e., real-world acquaintance) and partner-list containing frequently-interacted high-reputed nodes (i.e., online acquaintance). They both represent mutual trust relationship and encourage nodes to be cooperative. Firstly, nodes do not want to damage friendship easily to maintain their real-life reputations. Secondly, nodes would like to have more partners to gain more benefits. To avoid abusing social connections for attacks, nodes set high standard for friendship and partnership establishment, as introduced in Section 4.1.

(2) *Lightweight reliable server selection.* Given a number of server options, a client chooses a friend or partner, if available, without querying their reputations. When multiple friends exist, the client chooses the one with the highest local ranking, which is decided by the rating for services previously received from each friend.

(3) *Reputation evaluation.* We define a node's reputation as the amount of credits it accumulates through service providing or rating. As aforementioned, both friendship and partnership represent mutual trust in SocialTrust. Therefore, the social degree, i.e., the number of friends

- * *Corresponding Author. Email: shenh@clemson.edu; Phone: (864) 656 5931; Fax: (864) 656 5910.*
- *The authors are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634.*
  E-mail: {kangc, shenh, ksapra, guoxinl}@clemson.edu

and partners, of a node represents its level of trust from friends/partners [15]. On the other side, the reputation of a node represents its trust from non-friends/partners. Then, SocialTrust defines a node's impact factor by considering both its social degree and reputation and uses it to adjust reputation reward or punishment after a transaction, e.g., giving lower weight to ratings from lower-impact nodes and vice versa.

In SocialTrust, when a node becomes non-cooperative in either service providing or service rating, the file sharing system will be affected. Then, as in previous reputation systems [4], [6], [7], we assume that a node's trust determines its willingness to be cooperative in both activities. In other words, we do not separate the serving trust and rating trust but use a union value to represent its cooperation level in the system. Therefore, a node's trust (social connections and reputation) is determined based on both file serving and rating, which encourages nodes to be cooperative in both activities.

The above designs lead to several benefits to the P2P file sharing system. Firstly, the first two components reduce the reputation querying cost and service delay, and meanwhile still supports the objective of open and free P2P services. This encourages nodes to be continuously cooperative in order to keep their friends and partners. Secondly, with the third component, SocialTrust assigns reputation punishment proportional to a server's impact factor. This means that a node's reputation is built gradually, but drops in proportional to its impact factor, which helps to prevent nodes with a high reputation or social degree from gaining unfair advantages [16]. As a result, SocialTrust can ensure high-quality file sharing among nodes and potentially, enhance the popularity of these P2P file sharing systems.

To show SocialTrust's effectiveness on providing cooperation incentives, we use game theory to analyze node behaviors in Section 4.4. We also briefly discuss how to prevent certain attacks in SocialTrust in Section 4.6. In summary, SocialTrust has below contributions compared to previous reputation systems [3]–[8].

(1) SocialTrust exploits both social networks and reputation systems for efficient reputation management, i.e., reducing the reputation querying cost without constraining server options.
(2) SocialTrust proposes to consider both social degree and reputation of a node to adjust the weight of its rating for others and the degree of punishment for it. Thus, node reputation is more accurately evaluated and high-reputed nodes can hardly take advantage of its reputation for misbehavior, thereby motivating them to be constantly cooperative.
(3) SocialTrust considers both service rating and providing in reputation evaluation, which encourage nodes to be cooperative in both activities.

The purpose of SocialTrust is not to combine P2P file sharing services with online social networking services. Rather, SocialTrust exploits the trust within social networks (i.e., among friends and partners) for efficient reputation management, thereby enabling users to obtain the right file more efficiently. From this perspective, users would be willing to maintain certain social relationships for their own benefits. Actually, the Maze [17] P2P file sharing system has already provided an online social networking component to allow creating friendships for file sharing. Therefore, the proposed SocialTrust can be accepted by users in practical systems.

SocialTrust only needs users to update the client software to support the social network component and the server to update the reputation evaluation function. More importantly, in addition to the P2P file sharing applications, e.g., BitTorrent [18] and Akaimai [19], SocialTrust can also be applied to other applications where transactions are conducted in a peer-to-peer manner, such as video streaming through PPTV [20], routing in mobile ad hoc networks [21], and online business (e.g., eBay [22] and Amazon [23]). In these systems, nodes can follow the principles in SocialTrust to build social networks to facilitate the reputation management.

The remainder of this paper is arranged as follows. Section 2 presents the related work. Section 3 introduces the system background. Section 4 introduces the design of SocialTrust and how to prevent certain attacks. In Section 5 and Section 6, the performance of SocialTrust is evaluated with real trace and PlanetLab testbed. Section 7 concludes this paper with future work.

2 Related Work

In recent years, numerous research works have been conducted on reputation systems [3]–[8] in P2P networks. These works focus on how to aggregate reputation ratings and calculate the reputation efficiently and accurately. Though these systems are effective, they fail to utilize the social network properties in P2P file sharing to reduce reputation querying cost and encourage continuous cooperation, as mentioned in the introduction. Zhou and Hwang [3] observed a power-law distribution in user feedbacks in eBay and proposed the PowerTrust reputation system that selects few most reputable nodes to aggregate reputation feedbacks in order to improve the global reputation accuracy and aggregation speed. Zhang *et. al* [4] presented a reputation system built upon the multivariate Bayesian inference theory. It offers a theoretical basis for clients to predict the reliability of candidate servers based on self-experiences and feedbacks from peers. GossipTrust [5] uses randomized gossiping and power nodes to enable fast aggregation and fast dissemination of global reputation. PeerTrust [6] includes a coherent adaptive trust model for quantifying and comparing the trust of peers based on a transaction-based feedback system, which combines multiple parameters such as feedback a peer receives from other peers and the total number of transactions a peer performs. EigenTrust [7] computes a global reputation value for a peer by calculating the left principal eigenvector of a matrix of normalized local reputation values, thus taking into consideration the entire system's history with each peer and increasing the accuracy of global reputation. SFTrust [8] separates trust between the service providing and feedback in order to take full advantages of peers' service abilities for high performance.

In these reputation systems, a node must query the reputation values of the server options to select suitable servers, which generates high cost. Also, few systems leverage node trust in the social network for accurate global reputation evaluation.

There are also a number of works that leverage social networks for reliable services in P2P networks [9]–[13], [24] based on the property of "friendship foster cooperation" [14]. Since a user's friends are usually trustworthy and share similar interests with it, Chen *et al.* [24] exploited the friend relationships to perform reputation estimation, i.e., a node selects a file based on its friends' evaluation on the file. TRIBLER [9] is a social-based P2P file sharing system, which enables fast, trusted content discovery and recommendation by allowing nodes to retrieve files from taste groups, friends and friends-of-friends. Turtle [10] builds its overlay on top of pre-existent trust relationships among its users in order to withstand most of the denial of service attacks and allow both data sender and receiver anonymity. MyNet [11] is a P2P middleware platform and user interaction tool that allows everyday users to easily and securely access and share their devices, services, and content in real time. In the F2F system [12], a node chooses its neighbors (the nodes with which it shares resources) based on existing social relationships. This approach provides incentives for nodes to cooperate. SybilGuard [13] exploits the property that social connections usually represent trust relationships to detect Sybil nodes. It is based on the property that Sybil nodes usually have disproportionately small amount of connections with honest nodes.

These methods mainly use the trust among social relationship directly for certain services. However, since the friends of a node only cover part of the whole nodes in the system, by constraining the options of servers to friends, the objective of widely sharing of individual resources, even between strangers, cannot be realized in P2P file sharing applications. SocialTrust solves this problem by considering both social networks and reputation system for reputation system, thereby supporting widely and freely P2P file sharing.

3 Background

3.1 System Environment

SocialTrust is based on a traditional credit based reputation system, which aggregates service feedbacks, calculates reputation reward or punishment, update node reputation values, and replies to reputation queries. We assume that nodes can gradually build social connections with its friends (i.e., real-world acquaintances) and partners (i.e., online acquaintances) in the P2P network. This can be easily supported on client software, as in the Maze [25] system. Like previous social network based server selection methods [9]–[12], [24], SocialTrust considers friends/partners in the designed social network as trustable. Requesting for services from friends can enhance service quality since people do not want to damage their real-life reputations and instead would wish to build their real-life reputations through cooperation.

3.2 Trust Representation in SocialTrust

In SocialTrust, we define a node's trust as its cooperation level in service providing/rating, i.e., how likely it is to provide high-quality file and honest rating.

A node's trust is reflected in both the designed social network and the reputation evaluation system. As introduced in Sections 4.1 and 4.2, friendship and partnership are built based on mutual trust for file sharing. Therefore, the social degree, i.e., the number of friends and partners, of a node represents its level of trust from friends/partners. On the other side, as introduced in Section 4.3, a node's reputation is accumulated through service providing or rating with non-friends/partners. Therefore, the reputation value of a node represents its level of trust from non-friends/partners.

Considering a node with high reputation or high social degree is more likely to be trusted and be selected as the server, SocialTrust further defines the impact factor for each node based on both social degree and reputation. It is used to evaluate the reliability of a node's rating and adjust the punishment to a node, the details of which are introduced later in Section 4.3.

As a result, the punishment for misbehavior in SocialTrust lies in two aspects: its reputation will be reduced or its friendship/partnership may be terminated (as in Section 4.1). When a node's reputation falls below a certain threshold, the TA will notify all nodes to isolate the node, i.e., its file requests will be rejected directly. When a node has no friends/partners, it cannot take the benefits of friendship/partnership for file sharing.

3.3 Service Process

We consider a generic P2P file sharing environment in this paper. We define the node that receives a file as *client*, denoted by N_c, and the node that offers the file as the *server*, denoted by N_s. We define the reputation center as *Trust Authority (TA)* in SocialTrust. It is running on the server of the P2P file sharing system and is maintained by the owner of the P2P file sharing system. The system owner would be willing to run the TA since it can ensure high-quality file sharing, thereby providing better user experiences to nodes.

Then, the process of an interaction in such a system can be summarized as:

(1) A client generates a file request and identifies available servers for the request from peers in the system.
(2) The client selects one server based on their trustworthiness (i.e., probability of providing good services), which are obtained from the TA, and sends the request to the selected server.
(3) Upon receiving the file request, the server decides to offer or not to offer the requested file.
(4) Upon receiving the file, the client rates the quality of the received file and sends the rating to the TA to update the reputations of the two nodes.

In SocialTrust, we do not consider a general concept of Quality of Service (QoS) but define it as the quality of the obtained file, i.e., whether it is complete and correct. Therefore, nodes rate the received file by checking whether it is complete and correct. As in [3]–[8], the feedbacks from nodes are essential to evaluate node reputation. Then, the goal of our work is to improve the cost and effectiveness of previous reputation systems. We follow their assumption that the feedback from nodes can generally reflect the QoS, though ratings from different users for the same service may fluctuate. In other words, we accept minor fluctuations in the rating. In case the rating is dishonest, i.e., far from the actual value, the server can file a claim to the TA to disagree with the rating, like in the he-said-she-said attack. The TA then checks whether the rating is honest or not. The details on how to handle this is introduced in Section 4.6.1.

For easy description, we regard file sharing as a kind of **service** in the system, i.e., the server node provides the file service to the client node. Then, the cooperative and non-cooperative behaviors of the server and client node are defined as below.

Server node: A server node is cooperative when it provides the service with high quality for the received request. In contrast, a server node is non-cooperative when it refuses to provide service or even acts maliciously.

Client node: A client node is cooperative when it rates received service honestly. In contrast, a client node is non-cooperative when it purposely distorts its rating (i.e., giving poor/good ratings to good/bad services).

4 THE DESIGN OF SOCIALTRUST

SocialTrust is a credit based reputation system for P2P file sharing systems. It has three main modules.

Social networks (friendship and partnership networks): In the friendship network, a user's neighbors are his/her friends (i.e., real-world acquaintance). In the partnership network, a node's neighbors are high-reputed and frequently-contacted nodes (i.e., online acquaintances). Both relationships represent mutual trust. Nodes set high standard for them to avoid abusing them for attacks.

Lightweight reliable server selection: The friendship and partnership networks are exploited to alleviate the reputation querying cost on nodes and the reputation system. A client directly selects its friends or partners, if available, from the server option list, as the servers for its requested service. Only when the list does not include any friend nor partner, the client queries for the reputation values of available servers. As a result, many reputation querying operation can be avoided.

Reputation evaluation: Since both the reputation and social degree of a node reflect its trustability when it behaves cooperatively and the level of potential harm when it is non-cooperative (high-reputed and high-degree nodes have high probability to be selected as servers). We consider both a node's social degree and reputation to adjust the credibility of its service rating and reputation punishment. We present the details of each module in the following.

4.1 Social Networks

The social relationships of a node in SocialTrust include both offline friends and online partners. Trust relationship is not bi-directional in reality. However, we require that the friendship and partnership to be bidirectional to represent mutual trust for file sharing and ensure the fairness among nodes. Since friends are connected by certain social relationships in a social community, they would offer high QoS to each other with the intention of building high real-life reputations in their social communities (e.g., research lab and department). Thus, the friendship network motivates nodes to be cooperative continuously. A node's partners also have high probabilities to offer high QoS to the node according to their previous interaction (collaboration) records. Therefore, in order to maintain the partnership, which is a reflection of its trustability in file sharing, nodes would not arbitrarily decrease the QoS of their services.

Each node also maintains local ranks for friends and partners. For each friend/partner, the node calculates the average rating for services obtained from it and then ranks them in descending order. Such a local ranking is used to further compare the trust of different friends/partners for server selection, which will be introduced later in Section 4.2.

4.1.1 Friendship Maintenance

The friendship update (i.e., addition and deletion) in SocialTrust is an user-dependent behavior, and users are responsible for the consequence of adding a new friend. In other words, rational users would be cautious in accepting friend invitations. Consequently, the friendship network can reflect trustable and stable social relationship in real lives. Since the friendship is user centric, each user maintains its own friend-list. When a node, say N_i, wants to add another node, say N_j, into its friend-list, it sends a friend invitation to N_j. If N_j accepts the invitation, they become friends of each other.

A node may remove a friend when it receives uncooperative service from a friend or is notified by the TA that the node is a malicious node. If a user deletes a friend, they remove each other from their friend-lists. This process is enforced by fairness consideration of individual node that when a node finds that another node stops taking it as a friend (i.e., no longer trusting it), it would naturally remove the other node from its friend list (stop trusting it too).

4.1.2 Partnership Maintenance

A bi-directional partnership is established when both below conditions are satisfied.

(1) The interaction frequency between the two nodes is larger than a threshold, denoted by T_f.
(2) Each node's reputation value is larger than the partnership threshold, denoted by T_r.

The above two requirements match how people build partners in their businesses, i.e., entities with good and frequent past transaction records are more reliable.

The interaction frequency of a node, say N_i, with another node, say N_j, is calculated by

$$F(N_i, N_j) = \gamma F_{old}(N_i, N_j) + (1-\gamma)F_{new}(N_i, N_j)$$

where F_{new} and F_{old} denote the frequency measured in the current and the previous period, respectively, and $\gamma \in [0, 1]$ is an adjusting factor. The update period is the

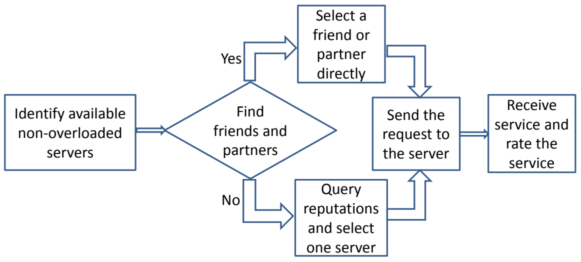

Identify available non-overloaded servers
Find friends and partners
Yes
Select a friend or partner directly
No
Query reputations and select one server
Send the request to the server
Receive service and rate the service

Fig. 1: The process of a client's server selection.

average time between two interactions among nodes in the system multiplied by a factor. In order to avoid periodical reputation querying for timely partnership update, the partnership of each node is maintained on the reputation system. After each reputation update, the reputation system notifies each node the change of partnerships,i.e., adding new partners or removing existing partners. Similar to the friendship, a node would not delete partners arbitrarily to save reputation querying cost and receive reliable services.

4.2 Lightweight Reliable Server Selection

Since the friendship and partnership represent certain trust, we exploit this property to alleviate the reputation querying cost. Specifically, a node directly selects friends or partners, when available, as the server for requested services without querying their reputations.

Figure 1 shows the process of a client's operation in server selection. When a client needs a service, it first identifies the available servers (i.e., server candidates) for the requested file through the file lookup function provided by the system. In this process, overloaded servers are excluded from the available server list to realize load balance and avoid large delay on file sharing. To realize this, the file lookup function simply does not check whether the overloaded servers contain the requested file. With such a scheme, the node with a high reputation and many friends will not receive too many requests and get overloaded. As a result, file requests are distributed to more servers in the system, leading to more balanced load and higher overall efficiency.

The client node then checks whether any of its friend(s) or partner(s) are in the list. If yes, it skips the step of querying the available servers' reputations and selects the friend or partner with the highest local ranking as the server directly. Each node records its average rating for services received from each friend/partner and ranks them in descending order of the average rating. If there is no friend or partner in the server list, it queries the reputation value of each server candidate from the reputation system, and chooses the one with the highest reputation. After the transaction is completed, the client sends the service rating to the reputation system. If the service is from a friend or partner, the node records the rating for local ranking.

4.3 Reputation Evaluation

The reputation system updates node reputations based on received service rating. In order to deter reputed nodes from conducting occasional misbehavior while still being regarded as reputed, we follow the principle that reputation is built gradually, but drops in proportional to its ability to bring about harm [16]. The probability that a misbehaving node is chosen as a server determines its potential harm to the system performance. Recall a node either selects its friend/partner or the available server with the highest reputation as the server for its request. As a result, the probability is determined by the node's reputation and social degree, which is the number of friends and partners. Therefore, we propose a metric called *Impact Factor* (denoted by $T \in [0,1]$) that integrates both reputation and social degree:

$$T(i) = \beta \frac{R(N_i)}{R_{max}} + (1-\beta)\frac{D(N_i)}{D_{max}}, \tag{1}$$

where $R(N_i)$ is the reputation of node N_i, R_{max} is the maximal reputation value allowed in the system, $D(N_i)$ is the social degree of N_i, D_{max} is the maximal number of friends and partners a node can have in the system, and β is an adjusting factor. Then, the impact factor of a node represents its harm when it is non-cooperative and the credibility of its service rating when it is cooperative.

SocialTrust considers both file serving and service rating as a part of a node's willingness to be cooperative in the system. Each node rates the server that has provided a file to it. Though this brings about extra costs, it is necessary to ensure the correctness of the reputation system and protect nodes from non-cooperative or malicious nodes, as in all previous reputation systems proposed by other researchers [3]–[8]. There are also ways that can reduce the load on clients and meanwhile ensure the collection of ratings for correct reputation evaluation. For example, we can let each node set a default rating for good services. Then, nodes do not need to manually input rating for the service provider unless it is not satisfied with the service. Due to page limit, we leave the investigation on such methods to future work.

We name the file sharing process between two nodes as an interaction between them. Each interaction is a game, in which each node selects a strategy for the sharing of a file: cooperative and non-cooperative. We use T_c and T_s to represent the impact factor of the client node and the server node, respectively. In below discussion, we assume file quality/rating accuracy can be correctly checked by the TA. We discuss how to check whether the rating and actual file quality are consistent, i.e., preventing he-said-she-said attack, in Section 4.6.1.

4.3.1 *Cooperative Server and Cooperative Client*

In this case, the server provides requested service to the client, which then rates the quality of the received service honestly. Then, the rating is always larger than 0: $Y_c \in (0, 10]$. The reputation system decides whether the rating is honest by asking the server whether it accepts the rating. If the server accepts the rating, the reputation credits given to the server node ($R_{s1} \in [0,1]$) and the client node ($R_{c1} \in [0,1]$) are

$$\begin{cases} R_{s1} = (1 + T_c * Y_c/10) * \alpha \\ R_{c1} = \alpha \end{cases} \tag{2}$$

In Equation (2), $\alpha \in [0, 0.05]$ is the unit reputation credit for file serving and service rating. Thus, the amount of reputation credits earned by the server depends on the rating and the impact factor of the client. The rationale behind such a design is that the higher impact factor the client has, the more trustable its rating is, and more reward should be assigned to the server. Equation (2) also shows that a server prefers to serve clients with high impact factor in order to earn more reputation, which motivates nodes to always maintain high impact factor.

4.3.2 Cooperative Server and Non-cooperative Client

In this situation, the server is cooperative in providing service but the client gives a bad feedback or does not provide any feedbacks. Then the server refuses to accept the rating (or no rating) given by the client and files a claim. When the non-cooperative behavior of the client is confirmed, the reputation credits assigned to the server ($R_{s2} \in [0,1]$) and the client ($R_{c2} \in [-1,0]$) are as below. We explain how the reputation system investigates whether the client rates honestly later.

$$\begin{cases} R_{s2} = \alpha \\ R_{c2} = -(1+T_c) * \alpha \end{cases} \tag{3}$$

We only assign the unit credit to the server since the rating provided by the client is dishonest and thus cannot reflect the QoS of the service provided by the server. The non-cooperative client is punished based on its impact factor: the higher impact factor it has, the more punishment it receives. Such a design reduces the reputation of nodes with higher impact factor more quickly when they are non-cooperative, preventing them from taking advantage of the high impact factor for non-cooperative behavior. Comparing Equation (3) with Equation (2), we see that the server earns $T_c * \alpha$ less reputation credits, which is the cost of choosing a dishonest client. Thus, servers are further motivated to provide service to high-reputed nodes, which in turn, motivates nodes to be cooperative in serving and rating.

4.3.3 Non-cooperative Server and Cooperative Client

In this case, the server provides malicious service and the client node rates the service honestly (i.e., giving bad feedback). We consider how to check whether the service and rating are consistent in Section 4.6.1. Therefore, the rating is smaller than 0: $Y_c \in [-10,0)$. Then, the reputation credits assigned to the server ($R_{s3} \in [-1,0]$) and the client node ($R_{c2} \in [0,1]$) are

$$\begin{cases} R_{s3} = -(1+T_c * |Y_c|/10) * (1+T_s) * \alpha \\ R_{c3} = \alpha \end{cases} \tag{4}$$

Above design follows the rationale that a non-cooperative server with high impact factor receives more punishment to prevent it from exploiting high reputation for misbehavior. The unit credit (α) is assigned to the client to encourage honest rating.

4.3.4 Non-cooperative Server and Non-cooperative Client

This situation occurs in collusion, where the server provides service with low quality or does not provide service at all, but the client still gives good feedback in order to boost the server's reputation. We first assume that collusion is not detected. In this case, the reputation credits assigned to the server node ($R_{s4} \in [0,1]$) and the client ($R_{c4} \in [0,1]$) are the same as those in Equation (2). But in reality, collusion generally is detected with certain probability under different detection algorithms. When the collusion is detected, involved nodes are punished. Then the actual credits assigned to collusion nodes can be expressed as

$$\begin{cases} R_{s4} = D_d * R_{s1} = D_d * (1+T_c * Y_c/10) * \alpha \\ R_{c4} = D_d * R_{c1} = D_d * \alpha \end{cases} \tag{5}$$

where $D_d \in [-1,1)$ is the adjusting factor considering the possible punishment for detected collusion behaviors. It is determined based on the probability of successful detection and the amount of punishment for collusion.

The collusion group can be detected based on their behavior pattern, i.e., always providing good feedbacks to each other and bad service/rating to outside nodes. We have proposed a method to detect colluding nodes in Section 4.6.4. Our previously proposed method [26] and other methods listed in the paper for combating collusion can also be used to deal with collusion in SocialTrust. Due to page limit, we leave the work on how to determine D_d to future work.

4.4 Game Theory Analysis

We analyze the behavior of both server and client in SocialTrust with game theory. We assume that nodes are rational, i.e., they always select the strategies that can maximize their benefits. When two nodes conduct a transaction, each node needs to either provide a file or rate the file. Therefore, we do not explicitly consider the cost for service providing or rating in the game theory analysis. We first define notations in an interaction between a client node and a server node.

(1) L_c: The loss of the client when receiving non-cooperative service, which includes the cost for requesting the service.
(2) B_c: The benefit of the client when receiving cooperative service.
(3) L_s: The loss (i.e., resource consumption) of the server node when providing cooperative service.
(4) B_s: The benefit of the server when it is non-cooperative, which includes the saved resource consumption and the overall profit from its non-cooperative behavior.
(5) l_r: The loss of a node when its reputation credit decreases by one unit, i.e., the decrease of the number of partners and the increase of the number of reputation queries and service rejections.
(6) b_r: The benefit of a node when its reputation credit increases by one unit, i.e., the increase of the number of partners and the decrease of the number of reputation queries and service rejections.

Clearly, L_c and B_c depend on the QoS of the service received by the client and vary on different scenarios. We can regard them as two fixed values if we only consider cooperative and non-cooperative QoS cases, as in previous reputation systems. Similarly, L_s and B_s can also be regarded as fixed values.

4.4.1 *Analysis of the Interactions among Nodes*

Based on above design, the overall benefits for the client node and the server node after one interaction are shown in Table 1. In the table, Co., Non., and Cli. denote cooperation, non-cooperation, and Client respectively.

TABLE 1: Benefit matrix for interactions among nodes in SocialTrust

		Server	
		Co.	Non.
Cli.	Co.	$B_c + R_{c1}b_r$, $R_{s1}b_r - L_s$	$R_{c3}b_r - L_c$, $B_s + R_{s3}l_r$
	Non.	$B_c + R_{c2}l_r$, $R_{s2}b_r - L_s$	$R_{c4}b_r - L_c$, $B_s + R_{s4}b_r$

We first analyze the action of the client. When the server is cooperative, the benefit of the client is $(B_c + R_{c1}b_r)$ when being cooperative and is $B_c + R_{c2}l_r$ when being non-cooperative. Since $R_{c1} = \alpha$ and $R_{c2} = -(1 + T_c)\alpha < 0$, $(B_c + R_{c1}b_r)$ is always larger than $B_c + R_{c2}l_r$. Thus, the client will always choose the cooperative strategy in order to maximize its benefit. When the server is non-cooperative, the benefit of the client is $R_{c3}b_r - L_c$ when being cooperative and $R_{c1}b_r - L_c$ when being non-cooperative. Since $D_d \in [-1, 1)$, $R_{c3} = \alpha > D_d * \alpha = R_{c4}$, the client would always choose to be cooperative. Therefore, cooperation is the dominate strategy for client since this strategy leads to more benefit no matter which strategy the server chooses.

We next analyze the action of the server. When the client is cooperative, the best strategy of the server depends on the punishment for non-cooperative service. If the punishment is large enough that $R_{s1}b_r - L_s > B_s + R_{s3}l_r$, cooperation brings more benefit to the server than the non-cooperation strategy. Clearly, we can set $\alpha b_r > L_s$ and $\alpha l_r > B_s$ to satisfy this. The condition can be transformed to $(1 + T_cY_c/10)\alpha b_r - L_s > B_s - (1 + T_c|Y_c|/10)(1 + T_s)\alpha l_r$, which is satisfied when $\alpha b_r > L_s$ and $\alpha l_r > B_s$. When the client is non-cooperative, the reward for the server when it is cooperative and non-cooperative is α and $D_d(1 + T_cY_c/10)\alpha$, respectively. Therefore, the best strategy for the server depends on whether the collusion detection is effective enough to make $D_d(1 + T_cY_c/10) < 1$. If yes, cooperation is the dominate strategy for the server since it leads to the highest benefit no matter what the client chooses.

In conclusion, we first configure α so that it satisfies $\alpha b_r > L_s$ and $\alpha l_r > B_s$. Then, if $D_d(1 + T_cY_c/10) < 1$, strategy (Co., Co.) is both the Nash equilibrium point (NEP) [27] and Pareto-optimal [28]. If $D_d(1 + T_cY_c/10) \geq 1$, strategy (Co., Co.) is not the Pareto-optimal but is the NEP since the client would always choose cooperation as discussed previously, and the server would also choose the cooperative strategy in order to gain the maximal benefits. This result justifies that nodes would choose to be cooperative with SocialTrust.

4.5 Improvement over Previous Reputation Systems

We further analyze how SocialTrust realizes the contributions claimed in Section 1 in below.

4.5.1 *Reducing Reputation Querying Cost*

Recall that SocialTrust utilizes the social network to reduce the reputation querying cost, as introduced in Section 4.2. We then check the percentage of reputation queries (P_{sc}) that can be avoided in SocialTrust.

We assume that the servers that can satisfy a request are evenly distributed among all nodes. We use M to denote the total number of nodes in the system. Let m_i be the number of service requests generated by node i, K_s be the average number of available servers for a request, and n_{fp_i} be the number of friends and partners node N_i has. Then, the probability that none of the available servers is a friend or partner (P_ϕ) can be calculated by $P_\phi = (1 - n_{fpi}/M)^{K_s}$. Then,

$$P_{sc} = \sum_{i=1}^{M}((1 - (1 - n_{fpi}/M)^{K_s})K_s m_i) / \sum_{i=1}^{M} K_s m_i \quad (6)$$

As shown in Equation (6), when nodes build friendships/partnerships in the network, P_{sc} is larger than 0. Furthermore, since a node is more likely to find requested services from its friends or partners, the actual P_{sc} should be larger than that calculated by Equation 6. This means SocialTrust can indeed reduce the reputation querying cost in P2P file sharing systems.

Note that the extra costs brought about by SocialTrust, i.e., those for service rating and rating accuracy/comfirmation check, commonly exist in reputation systems for P2P file sharing systems. Therefore, the claim that SocialTrust can save cost is made on the comparison with other reputation systems.

4.5.2 *Providing Accurate Reputation Evaluation*

Traditional reputation systems do not consider social network properties in reputation reward or punishment. In some of these systems (e.g., EigenTrust), the reputation of a rater is used as the weight to measure the credibility of its reputation feedback. Then, the reputation update in these methods in the four scenarios during the reputation evaluation can be shown as:

$$\begin{cases} R'_{s1} = (1 + R_c * Y_c) * \alpha, & R'_{c1} = \alpha \\ R'_{s2} = \alpha, & R'_{c2} = -\alpha \\ R'_{s3} = -(1 + R_c * Y_c) * \alpha, & R'_{c3} = \alpha \\ R'_{s4} = D_d * (1 + R_c * Y_c) * \alpha, & R'_{c4} = D_d * \alpha \end{cases} \quad (7)$$

where R_c denotes the reputation value of the client (rater). Comparing Eq. (7) with Eq. (2) and Eq. (4), we see that in calculating the reward and punishment for the server, rather than using R_c as in traditional methods, SocialTrust uses the client's impact factor, which is determined by both its reputation (R_c) and social degree. SocialTrust leverages the social network property that higher-degree or higher-reputed nodes tend to be more trustable in measuring the credibility of a rater's reputation feedback. Thus, it offers more accurate reputation measurement and mitigates the negative influence on reputation evaluation caused by dishonest ratings, as shown later in the experiment (Figures 5 and 6).

4.5.3 *Encouraging Continuous Cooperation*

SocialTrust can encourage nodes to be continuous cooperation through both the designed social network and the reputation reward/punishment system.

Firstly, Eq. (6) indicates that the more friends/partners a node has, the more reputation queries (i.e., cost) it can save. As stated in Section 4.1, the friendship is usually stable while the partnership is built dynamically. If a node has a high reputation, it can pass the reputation thresholds of more nodes and be accepted as the partner of more nodes. As a result, a node would not maintain its reputation value barely above the threshold but would like to accumulate its reputation as high as possible to save more cost on reputation querying.

Secondly, comparing Eq. (7) with Eq. (3) and Eq. (4), we see that unlike SocialTrust, traditional methods do not relate the punishment for a non-cooperative node to its reputation. Thus, strategic high-reputed nodes can occasionally be non-cooperative while maintaining reputed status to gain unfair advantages. On the contrary, SocialTrust relates the punishment to a node's impact factor, which means a node's reputation increases slowly but drops quickly if it behaves maliciously. Such a design can motivate nodes to be continually cooperative.

4.6 Preventing Attacks

In this section, we further discuss some attacks in current reputation systems, i.e., he-said-she-said attack [29], denial of service (DoS) [30], whitewashing and traitor attack [31], and collusion [31]. We introduce how SocialTrust prevents these attacks in below sections.

4.6.1 He-said-she-said Attack

In the he-said-she-said attack [29], the claims from the server and client are not consistent. For example, the server may claim that it has provided a file correctly while the client claims that it only received a faulty or malicious file. To prevent this attack, we need to check whether the claim from the client, i.e., rating, match with the claim from the server, i.e., actual quality of the service. This judgement is also needed by the reputation evaluation process, as mentioned in Section 4.3.

In order for the TA to judge correctly, we use the public and private key technique to ensure the authenticity of the requests and files provided by servers. When a node generates a request or provides a file, it encrypts the checksum of the request or file with its private key, and sends the encrypted checksum along with the request or the file. After the file provision interaction, if either side files a claim to the TA, the TA requires the server to submit its received request from the client and requires the client to submit its received file from the server. The TA then checks the authenticity of the submitted request and file based on the attached checksum using the public keys of the client and the server. Based on the request and provided file, the TA can determine whether the file is authentic and satisfies the request, and then makes a judgement of the claim. We will explain the details for this judgement later on. In order not to be detected, the non-cooperative client or server may fabricate its received request or file or may not submit their received request or file. If any side that fails to provide its received request/file or its provided file/request cannot be verified by the checksum using the corresponding public key, it is regarded as non-cooperation.

The checksum can ensure that the submitted request/file is not fabricated. However, the client may give a dishonest rating or the server may provide a faulty file or reject a honest rating. Thus, the TA may need to further check whether the quality of the offered file satisfies the request. As in [4], [6]–[8], we focus on the reputation system rather than how to check the consistency between the rating and service. Therefore, we give a general guidance in the paper. Specifically, the TA compares the file with its previously verified high-quality files for the request, i.e., these that have received high ratings for the same request, to determine whether the file satisfies the request. In case such a verification file does not exist, the TA uses voting from trustable third parties, e.g., selected high-reputation nodes, to check the quality of the offered file. Automatic file quality detection algorithms proposed by other researchers can also be adopted by SocialTrust to realize this purpose.

After the checking, the TA detects which side has lied about the service and conduct corresponding punishment. Finally, the he-said-she-said attack can be prevented in SocialTrust and server reputation can still be evaluated correctly since nodes cannot lie about their received ratings or services.

4.6.2 Deny of Service

In the Denial of Service attack [30], attackers attempt to cause the TA to become overloaded (e.g. by sending many reputation value queries) to prevent the calculation or dissemination of reputation values and cause denial of services on the TA. Attackers may also attempt to lower the reputation of victim nodes, so their service requests will be rejects by other nodes, thus causing denial of services on nodes. Since the latter case has already been discussed in Section 4.6.1, we focus on how to prevent malicious nodes from overloading the TA and consequently intervening the calculation or dissemination of reputation values in this subsection.

In the Denial of Service attack, malicious nodes query node reputation values (misbehavior 1) or create false claims on the QoS of servers or the ratings of clients (misbehavior 2) frequently as mentioned in the previous subsection, which require significant network or computational resources. Consequently, the TA would be overloaded and cannot provide services regularly, thereby leading to denial of services. To prevent such an attack, we enforce two additional strategies in the TA to deter the two misbehaviors, respectively.

- Misbehavior 1: The TA sets a quota, N_{qr}, for the number of reputation queries each node can generate in a unit time, e.g., an hour, to limit the number of reputation queries on the TA. When the number of a node's reputation queries exceeds the assigned quota, its queries are rejected by the TA directly.
- Misbehavior 2: The TA gives severe punishment to nodes that are detected to have such a misbehavior, i.e., provide incorrect claiming about the quality of service or the rating. Section 4.6.1 introduces how TA can detect such a misbehavior.

Both the two strategies constrain the overhead on the TA, thereby thwarting the denial of service attack. In the first strategy, N_{qr} is decided by the normal maximal number of queries TA can handle in a unit time divided by the total number of nodes in the system. Further, the second strategy encourages nodes to be honest on providing services or rating received services.

4.6.3 *Whitewashing and Traitor Attack*

In the whitewashing and traitor attack [31], malicious users purposely exploit the reputation system to recover their reputation values. In the whitewashing attack, a low-reputed user can simply discard its current ID and registers for a new ID to gain the initialization reputation. In the traitor attack, a user can behave uncooperatively until its reputation drops right above the threshold for malicious behavior detection, and then behave cooperatively to earn some reputation. Then, the user is always regarded as cooperative node.

The whitewashing attack can be alleviated by 1) assigning a low initial reputation, i.e. slightly over the threshold, 2) increasing the difficulty of acquiring a new ID, e.g., requiring references from reputed users, and 3) binding user ID with its other attributes (e.g., IP or MAC address) to limit the number of IDs a user can create. As a result, users are encouraged to behave cooperatively rather than relying on creating new IDs to be able to provide or receive services in the system.

The traitor attack can be naturally thwarted in SocialTrust. In SocialTrust, once a node's reputation value becomes low, its friendships or partnerships with others would be removed. Recall that nodes prefer to select friends or partners as the server for their requests. Then, the node has a low probability of being selected as the server. This means that the node will have few opportunities to earn credits to recover its reputation. As a result, the traitor attack is not effective in SocialTrust since once a node's reputation value becomes low, it is very difficult to recover it.

4.6.4 *Collusion*

In collusion attack [31], a group of nodes purposely provide positive ratings to each other to boost their reputation values. Previous studies showed that colluders tend to provide low QoS to others [25], [32]. Thus, collusion would make the reputation system malfunction as clients are misled to choose colluders with high reputations as servers, which actually will provide low QoS.

In SocialTrust, a node creates new partners based on reputation values. Then, with collusion, nodes will also be misled to choose colluders as their partners. Also, since nodes add friends based on its their decisions, they may also add colluding nodes as friends mistakenly. In order to restrain this negative impact of the collusion, we enforce a strategy to remove friends/partners in SocialTrust. Specifically, when a node finds that the percentage of good services (gs) it has received from a friend/partner, is lower than a predefined threshold T_{gs}, it will remove the friend/partner. We can set T_{gs} to a large value (e.g., 80%) since friends/partners are supposed to provide good services to each other.

With such an additional strategy, colluders can hardly be friends or partners of non-colluders since they rarely behave cooperatively when interacting with non-colluders [25], [32]. They may only build friendships/partnerships with other colluders. Since a client

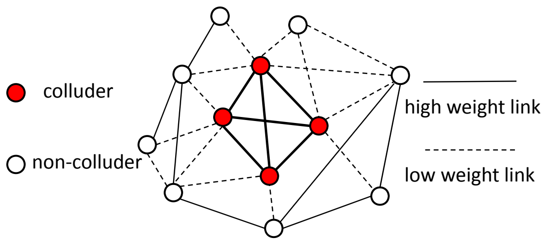


Fig. 2: Demonstration of collusion group in reputation rating graph.

node always firstly selects its friends/partners for services, thus colluders have few opportunities to receive service requests and provide faulty files to others, though they have high reputation values. This means that colluders cannot benefit from the boosted high reputation values. Eradicating the benefits of collusion discourage nodes to be colluders and prevent the collusion in SocialTrust. Note that such an additional strategy is enabled in addition to the friendship/partnership maintenance strategy in Section 4.1 only when colluder prevention component is selected for SocialTrust.

We further introduce a method to detect colluders. Since colluders tend to provide high ratings to each other frequently and offer low QoS to non-colluders (hence receiving low ratings from non-colluders), we can draw a reputation rating graph to detect colluding groups. In the reputation rating graph, each user is represented by a vertex, and two vertices are connected if they have had interactions. The weight of each link is the average rating between the two connected nodes in a unit time period. Then, since colluders tend to provide high ratings to each other and receive low ratings from nodes outside the colluding group, colluders are connected with high weight links and low-weight links with outside nodes, as shown in Figure 2. If we remove the links with weights lower than a threshold, T_w, from the reputation rating graph, the colluding groups are shown as isolated connected clusters in the group. We can then identify such clusters as the suspicious colluding groups.

As a result, nodes can still receive high QoS services from "colluders" or isolate "colluders" from providing services to them, which greatly alleviates the negative effects of collusion in SocialTrust.

5 Performance Evaluation

We used the trace from LiveJournal [33] in the test. LiveJournal is a free online community with more than 10 million members. We randomly selected a medium social network with 10,500 users from the trace for our test. We use the friend relationships in the trace to construct the friendship links in SocialTrust. Such a setting reflects real world friendship networks. Partnerships among nodes are built gradually in the test. We randomly selected 520 nodes as non-cooperative nodes in the test.

We adopted a simple model to simulate the willingness to be cooperative. We assigned each node (both cooperative and non-cooperative) a level (L) randomly in range [1, 10]. For a cooperative (non-cooperative) node, the higher its level (L) is, the more likely it behaves cooperatively (non-cooperatively). We set the probabilities of levels 1 to 10 linearly in the range [0.55, 1], i.e., the start probability is 0.55 for level 1 and the maximal probability is 1 for level 10. Then, the probability of a cooperative node to behave cooperatively (P_g) and the probability for a non-cooperative node to behave non-cooperatively (P_b) are defined as $P = 0.55 + 0.05 * (L - 1)$. Such a setting means that the higher level a node is, the more likely it behaves cooperatively or non-cooperatively.

As in most reputation system, the reputation value of a node lies in the range of [-1,1]. Initially, all nodes are regarded as cooperative nodes by setting their initial reputation value in range [0,1]. Nodes with negative reputation values are identified as non-cooperative nodes, and their services requests are always rejected by others. The reputation threshold of partnership (T_r) of a node was randomly chosen from a medium range of [0.3, 0.8]. Based on the average interaction frequency in the trace, the contact frequency threshold (T_f) was set to 1 transaction every 50 rounds. Also, based on analysis of the trace, the reward unit (α) was set to 0.04.

In the test, each experiment consists of 200 rounds. In each round, we reasonably assume each node has 60% probability to generate a request. We assume that there are m nodes containing the requested file for each request. Considering the total number of nodes is not huge, m is randomly selected from [3, 6]. We let the file quality fall in range [-10,10]. For the server, the QoS of its service is randomly selected from (0,10] and [-10, 0]when it is cooperative and non-cooperative, respectively. If the client is cooperative, it honestly gives rating $QoS + r$, where r represents the rating bias and is chosen from [-1,1] randomly. The final rating value is in the range [-10, 10]. If the client is non-cooperative, it gives rating randomly selected from [-10,0] for cooperative service and (0,10] for non-cooperative service.

We compared SocialTrust with SocialOnly [9], [10], [12] and EigenTrust [7]. They rely on social networks and user feedbacks for reputation management, respectively. SocialOnly is adapted from previous social network based P2P services, e.g., TRIBLER [9], Turtle [10], and F2F [12]. It uses the same concept in these works, i.e., friendship fosters cooperation, to deduce server reputations by letting each node query the reputation values of available servers from its friends. A node considers a server as non-cooperative when at least four of its friends report negative reputation value for the server. If none of a node's friends knows the reputation of available servers, the node randomly selects a server. EigenTrust [3] calculates the reputation of a node based on the feedbacks from other nodes weighted by the rater's reputation, while SocialTrust weights the feedback based on impact factor. We set the confidence interval to 95% in the experiment.

5.1 Efficiency of Reputation Systems

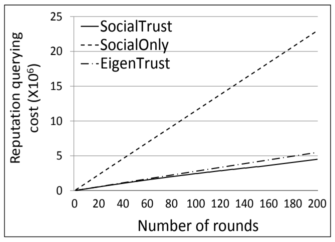
Figure 3(a) shows the reputation querying cost, measured by the total number of generated reputation queries, in the three reputation systems. We see that the results follow SocialOnly>EigenTrust>SocialTrust, and SocialOnly generates much higher cost than SocialTrust and EigenTrust. For SocialOnly, each node needs to query all its friends for each available server, leading to a large amount of reputation queries. In SocialTrust, friendship and partnership are exploited to alleviate the reputation query for available servers. In EigenTrust, the service requester needs to query the reputation of all available servers. Therefore, SocialTrust produces lower cost than EigenTrust. This result justifies that SocialTrust realizes the goal of reducing reputation querying cost by leveraging friendship and partnership. Though the cost saving is resulted from selecting friends/partners for service directly, we can see later that such a design does not sacrifice the effectiveness of SocialTrust in guiding trustworthy server selection.

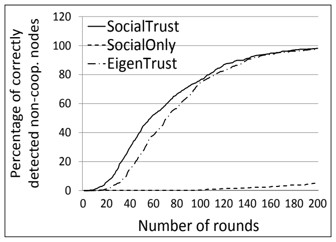
5.2 Effectiveness of Reputation Systems

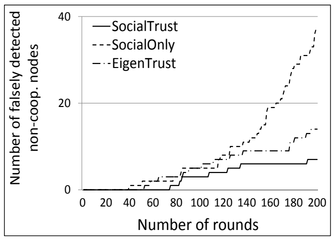
Figure 3(b) illustrates the percentage of correctly detected non-cooperative nodes throughout the test. We find that the results follow SocialTrust>EigenTrust>> SocialOnly in the first 180 rounds. Also, both SocialTrust and EigenTrust can detect almost all non-cooperative nodes in the end while SocialOnly can only identify a small portion of non-cooperative nodes. SocialTrust decides the punishment for a non-cooperative node based on its impact factor. Thus, it can decrease the reputation of non-cooperative nodes with high impact factor quickly. Moreover, SocialTrust adjusts the rating based on the impact factor of the rater. As a result, its reputation evaluation can more accurately reflect the behavior of nodes, leading to better detection of non-cooperative nodes. EigenTrust only considers the rater's reputation for the credibility of its reputation feedback. Therefore, the reputation evaluation in EigenTrust is not as accurate as that in SocialTrust, especially for high reputation nodes, leading to a slow detection. For SocialOnly, each node usually has limited interaction records and thereby a node's friends can provide very limited and updated reputation information of available servers. As a result, SocialOnly can only detect a small amount of non-cooperative nodes.

Figure 3(c) presents the number of falsely identified non-cooperative nodes. We see that the results follow SocialOnly >EigenTrust>SocialTrust. SocialOnly has the most false detections due to its local reputation calculation with limited interaction records. For SocialTrust, it generates fewer false detections than EigenTrust since it adjusts the reputation punishment by impact factor. Therefore, when a node has low impact factor, the punishment for its non-cooperative behavior is relatively low, which prevents it from falling into non-cooperation category occasionally. The consideration of the rater's impact factor also leads to more accurate reputation evaluation in SocialTrust.

Figure 3(d) plots the number of non-cooperative services received by all nodes. We see that the result follows SocialOnly >EigenTrust>SocialTrust. This is because the abilities of the three methods to exclude non-cooperative nodes follow SocialOnly>EigenTrust>SocialTrust, as shown in Figure 3(b). The more unidentified non-cooperative nodes in the system, the more non-cooperative services. This result further verifies the efficiency of SocialTrust in excluding non-cooperative nodes


(a) Reputation querying cost.


(b) Percentage of correct detections.


(c) Number of false detections.

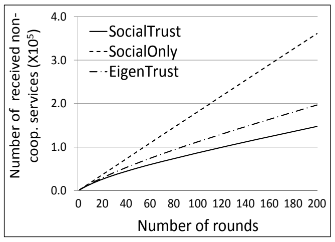

(d) Number of received non-cooperative services.

Fig. 3: Efficiency and effectiveness of the three reputation systems.

and preventing non-cooperative services in P2P system. The result also shows that the scheme in SocialTrust that selects friends directly when available doesn't compromise the performance since friends are more likely to be cooperative with each other [9]–[12], [14].

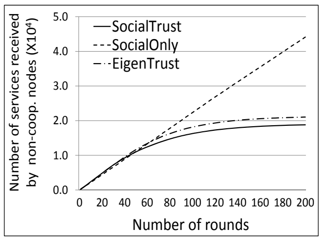
Figure 4(a) shows the number of cooperative services received by non-cooperative nodes. As the number of rounds increases, SocialOnly cannot prevent non-cooperative nodes from receiving services, while EigenTrust and SocialTrust successfully exclude non-cooperative nodes from receiving services. SocialTrust can prevent more services for non-cooperative nodes than EigenTrust. This is because the number of cooperative services received by non-cooperative nodes is proportional to the number of non-cooperative nodes that are not identified. As demonstrated in Figure 3(b), the ability to exclude non-cooperative nodes follow SocialTrust>EigenTrust>SocialOnly.

Figure 4(b) plots the number of rejected service requests from cooperative nodes. We see that the number follows the same relationship as in Figure 3(c) (i.e. SocialOnly>EigenTrust>SocialTrust). This is because the number of rejected service requests from cooperative nodes is proportional to the number of falsely detected non-cooperative nodes. When cooperative nodes are falsely detected as non-cooperative nodes, their subsequent requests are rejected. Therefore, SocialTrust generates the least rejected service requests, and EigenTrust produces fewer rejected service requests than SocialOnly. This further justifies that SocialTrust is more reliable than EigenTrust and SocialOnly on reputation evaluation.

In the end of the test, each node has 8.9 friends and 4.3 partners on average. This is because the LiveJournal trace is not from a very active social network and we only had 200 rounds of interactions in the experiment. However, such a limited number of friends/partners can save around 20% reputation query cost, as shown in Figure 3(a). This is because, as indicated by previous research [34]–[36], friends and partners are more likely to share files with each other. We also see that with the social network, SocialTrust can better ensure cooperative file sharing than other methods. Therefore, if we use a more active trace with more social relationships, the efficiency improvement can be better. In summary, social relationships can positively enhance the overall efficiency of the P2P file sharing system.

5.3 Accuracy of Reputation Evaluation

In this experiment, we evaluate the accuracy of the reputation systems in measuring how cooperative or how un-


(a) Number of services received by non-cooperative nodes.

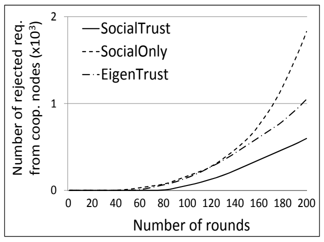
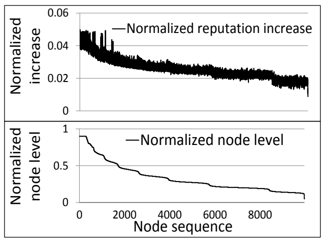

(b) Number of rejected requests from cooperative nodes.

Fig. 4: Effectiveness of reputation systems.


(a) SocialTrust.

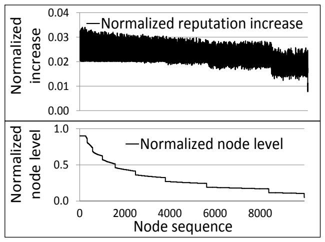
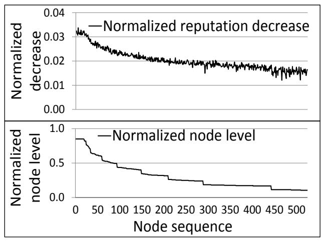

(b) EigenTrust.

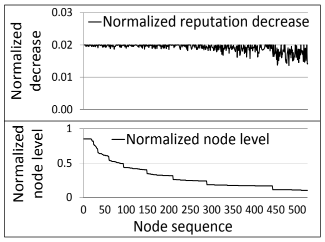
Fig. 5: Accuracy in reputation evaluation of cooperative nodes.

cooperative a node is. The degree of a server's QoS and a rater's rating honesty are decided by their impact factors. If the selected server is cooperative in serving, the QoS of its service is calculated as $r + 10 * T_i/T_{Max}$, where T_i is the node's impact factor, and T_{Max} is the maximal impact factor. If the calculated QoS is less than 0 or larger than 10, it is set to 0.1 or 10, respectively. If the client is cooperative, it gives rating by: $QoS + r * (1 - D_i/D_{Max})$, where its social degree D_i controls the deviation from the honest rating. If the client is non-cooperative, it's rating is randomly selected from range [-10,0]. Here, we do not set the rating fluctuation associated with the rater's social degree because we only want to test the effect of punishing nodes based on their impact factors. If the selected server is non-cooperative, the QoS of its service is calculated as $-(r + T_i/T_{Max})$, which is also confined to the range of [-10, 0]. Then, if the client is cooperative, it gives rating by: $QoS + r * (1 - T_i/T_{Max})$. Otherwise, it gives rating randomly in range (0, 10].

The normalized reputation increase refers to the average reputation increase per interaction, and it is calculated as the final reputation value increase divided by the total number of interactions. The normalized node level represents how actually cooperative a node is. Recall that the probability of a node to behave cooperatively is decided by its node level, and its QoS or rating honesty is decided by its social degree. Therefore, node i's normalized node level is defined as $N_L = 0.5 *$



(a) SocialTrust.



(b) EigenTrust.

Fig. 6: Accuracy in reputation evaluation of non-cooperative nodes.

$L/L_{max} + 0.5 * T/T_{max}$, in which L and T are the level and the average impact factor, while L_{max} and T_{max} are their maximal values, respectively.

Since SocialOnly does not provide a global reputation for each node, we only measured that of SocialTrust and EigenTrust. We rank all cooperative nodes by their normalized node levels, and show each node's normalized node level and normalized reputation increase in SocialTrust and EigenTrust in Figure 5(a) and Figure 5(b), respectively. We see that SocialTrust leads to more similar trend between the normalized reputation increase and the normalized node level. The reason is that SocialTrust considers both a node's social degree and reputation in evaluating a node's impact factor, rather than only considering reputation as in EigenTrust.

We also tested the relationship between the normalized reputation decrease and normalized node level for non-cooperative nodes. The results are shown in Figure 6(a) and Figure 6(b). The normalized reputation decrease is calculated as the average reputation decrease per interaction. We see that the normalized reputation decrease more closely approximates the normalized node level in SocialTrust than in EigenTrust. This is because SocialTrust utilizes the trust of a node to adjust the punishment for non-cooperative behaviors: high-trust nodes receive more reputation punishment. Above results demonstrate the accuracy of SocialTrust in evaluating the reputation by considering node trust.

We define the *correlation* between the normalized reputation increase and normalized node level as

$$C = \frac{N\sum RL - \sum R \sum L}{\sqrt{(N\sum R^2 - (\sum R)^2)(N\sum L^2 - (\sum L)^2)}} \tag{8}$$

where N denotes the total number of cooperative nodes, L means the normalized level of a node, and R is the normalized reputation increase of the node. Figure 7 shows the correlation every 10 rounds. We find that SocialTrust always generates much higher correlation than EigenTrust. This is because SocialTrust provides more accurate reputation evaluation, as explained previously. We also see that the correlation between the two metrics increases when the number of rounds increases. This is because with the experiment going on, the evaluated reputation of each node can more accurately reflect its level of cooperative behavior.

5.4 Effect of Friendship in SocialTrust and SocialOnly

Both SocialTrust and SocialOnly utilize social friendship in reputation evaluation. In SocialTrust, friendships and partnerships are exploited to reduce reputation queries by directly selecting friends/partners from available servers, while in SocialOnly, friendship is used to provide reputation information of available servers. We further measure the effect of friendship in the two systems.

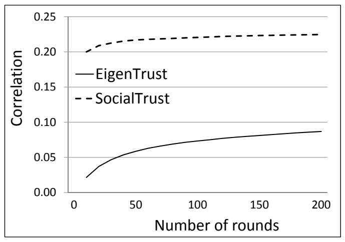
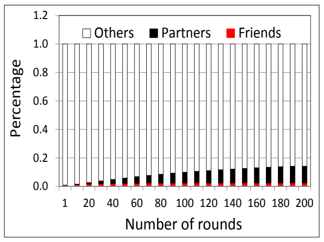
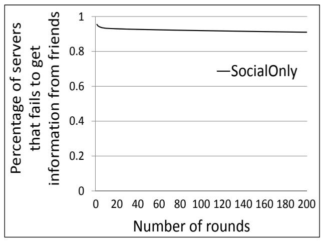


Fig. 7: Accuracy in reputation evaluation of all nodes.



(a) Distribution of selected servers in SocialTrust.



(b) Percentage of servers whose reputation cannot be provided by friends in SocialOnly.

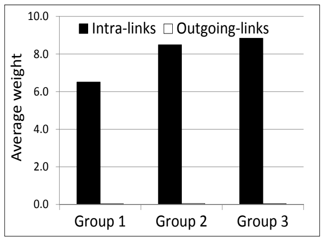
Fig. 8: Effect of friendship in SocialTrust and SocialOnly.

We plot the percentage of servers from friends, partners and others in Figure 8(a). We see that the percentage of servers from friend-list remains almost stable, and the percentage of servers from partner-list increases from 0 to about 13% from round 1 to round 200. This is because the probability that available servers are in a node's friend list is low in the beginning. Partners are accumulated as time goes on, which shows that nodes accumulate reputation from interactions and build trustable partnerships gradually to save reputation querying cost.

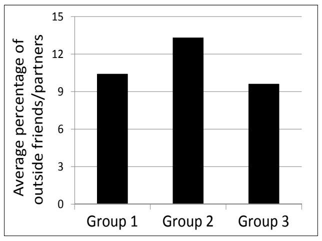
Figure 8(b) shows the percentage of servers whose reputation information cannot be provided by friends in SocialOnly. It maintains high (i.e., >90%) throughout the experiment and decreases slightly as the number of rounds increases. Though the number of nodes that a node interacts increases as the experiment runs, the number is very limited compared to the network size. Therefore, nodes have limited reputation information of other nodes in the system. The limited reputation information held by individual nodes leads to poor performance of SocialOnly, as shown in previous experiments. Combining the results in Figure 8(a) and Figure 8(b), we conclude that friendship can better serve the purpose of representing direct trust than the purpose of providing reputation information.

5.5 Collusion Detection Performance

We further measured the performance to prevent collusion attack introduced in Section 4.6.4. We purposely randomly selects 3 nodes and regard each node and its friends in the LiveJournal trace as a colluding group. Finally, the three colluding groups contain 11, 13, and 14 nodes, respectively. These colluding nodes always provide high ratings to each other but low QoS services



(a) Average weight of intra- and outgoing- links.



(b) Percentage of outside friend/partners.

Fig. 9: Effectiveness of collusion detection methods.

to nodes outside the colluding group. Other settings are the same as mentioned in the beginning of Section 5.

We define intra-links as the links connecting nodes in the colluding group and outgoing-links as these connecting a colluder with a non-colluder. We first measured the average weights of intra- and outgoing-links on the reputation rating graph, as shown in Figure 2. The weight of a link is calculated as the average rating between the two connected nodes in a unit time period. The test results for the three colluding groups are shown in Figure 9(a). We see that the average weight of outgoing-links is very small compared to that of the intra-links. Such a result verifies that the design mentioned in Section 4.6.4 can detect collusion groups.

Outside friends/partners refer to the friends/partners outside a colluding group. We further measured the average percentage of outside friends/partners of the three colluding groups, as shown in Figure 9(b). We find that they have only about 10% of outside friends/partners on average. This is because with the scheme proposed in Section 4.6.4, normal nodes terminate the friendships/partnerships with the colluders after several rounds of interactions. Such a result shows that the proposed method can isolate colluders, thereby reducing the risks of receiving faulty files from colluders.

6 Experiment on PlanetLab

We further deployed SocialTrust on the real-world PlanetLab [37] testbed. In this experiment, since there is no data on whether a PlanetLab node is willing to be cooperative or not, we simulate such a metric by a subjective property of each node, i.e., the average communication delay with all other nodes. The service latency is calculated as the period of time a node waits before receiving the requested file. We identified 100 PlanetLab nodes that have stable response latency. Because a server's clients can be any node in the system, the effect of the server-client geographical distance on the final global reputation is eliminated. We let each PlanetLab node represent 100 virtual nodes. We still use the social relationship from the LiveJournal trace [33] to build the initial social network in PlanetLab. We first measured the average delay of each node to the requests from all other nodes, and ranked the average delays in a descending order. We choose the 94th percentile value as the threshold of cooperation. Nodes with delays lower than this threshold are regarded as non-cooperative nodes, resulting in 629 non-cooperative nodes. Other settings are the same as those in the simulation.

6.1 Effectiveness of Reputation Systems

We measured the accuracy of trust evaluation in the Planetlab test: percentage of of correct detections, the number of false detections, number of non-cooperative services received by all nodes, number of good services for non-cooperation nodes. The results are shown in Figure 10(a), Figure 10(b), Figure 10(c), and Figure 10(d), respectively. We find that the three methods show the same relationship as in the corresponding simulation results shown in Figure 3(b), Figure 3(c), Figure 3(d), and Figure 4(a) due to the same reasons. This result further demonstrates the effectiveness of SocialTrust in reputation evaluation in a real environment due to the same reasons stated in previous simulation.

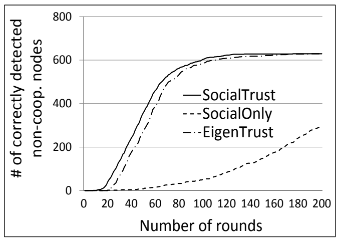
In order to further show SocialTrust's ability to reflect a node's actual behavior in providing service, we designed another algorithm, named Random, in which a node randomly picks an available server. We then measured their accumulation service latency, i.e., the accumulated file quality. The test results show that SocialTrust presents about 10% less service latency than Random, i.e., the QoS is enhanced by 10%. This is because SocialTrust can accurately reflect the actual reputation of the node, as previously explained. Guided by the calculated reputations based on real node QoS, nodes can always choose the server with low overall delay for service. Then, the overall latency can be reduced. This result confirms that SocialTrust can effectively evaluate node reputation in real environment.

7 Conclusions

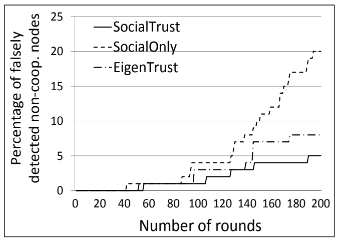
In this paper, we propose a social network based reputation system for P2P networks, namely SocialTrust, which exploits social network properties to save reputation querying cost and enhance the reputation evaluation accuracy. Since friends/partners in social networks usually trust each other, SocialTrust lets each node directly select its friends/partners, if available, as service provider without querying their reputations, thereby reducing the reputation querying cost. Further, SocialTrust integrates both social degree and reputation of a node as its impact factor for reputation clearance. Since nodes with high social degree/reputation are more likely to be selected as servers, they are more harmful when they are non-cooperative. Therefore, the impact factor is used to adjust the reputation reward and punishment, thus motivating nodes to be continually cooperative. Extensive simulation demonstrates the efficiency and effectiveness of SocialTrust. In the future, we plan to investigate the correlation between the friendship and reputation evaluation to further enhance the accuracy of reputation evaluation.

References

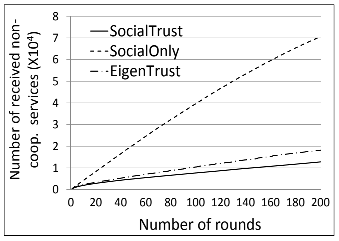
[1] D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: The bell tolls?" *IEEE Dist. Systems Online*, 2005.
[2] E. Sit and R. Morris, "Security considerations for peer-to-peer distributed hash tables," *Peer-to-Peer Systems*, 2002.
[3] R. Zhou and K. Hwang, "PowerTrust: A robust and scalable reputation system for trusted peer-to-peer computing," *IEEE TPDS*, 2007.



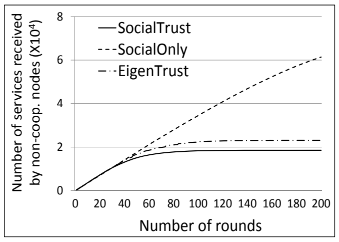
(a) Percentage of correct detections.



(b) Number of false detections.



(c) Number of received non-coop. services.



(d) Number of services received by non-coop. nodes.

Fig. 10: Efficiency of reputation systems in PlanetLab experiment.

[4] Y. Zhang and Y. Fang, "A fine-grained reputation system for reliable service selection in peer-to-peer networks," *IEEE TPDS*, 2007.

[5] R. Zhou, K. Hwang, and M. Cai, "Gossiptrust for fast reputation aggregation in peer-to-peer networks," *IEEE TKDE*, 2008.

[6] L. Xiong and L. Liu, "PeerTrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE TKDE*, 2004.

[7] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proc. of WWW*, 2003.

[8] Y. Zhang, S. Chen, and G. Yang, "SFTrust: A double trust metric based trust model in unstructured p2p system," in *Proc. of IPDPS*, 2009.

[9] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Y. 0015, A. Iosup, D. H. J. Epema, M. J. T. Reinders, M. van Steen, and H. J. Sips, "Tribler: A social-based peer-to-peer system," in *Proc. of IPTPS*, 2006.

[10] B. Popescu, B. Crispo, and A. Tanenbaum, "Safe and private data sharing with turtle: Friends team-up and beat the system," in *Proc. of SPW*, 2004.

[11] D. N. Kalofonos, Z. Antonious, F. D. Reynolds, M. Van-Kleek, J. Strauss, and P. Wisner, "MyNet: A platform for secure p2p personal and social networking services," in *Proc. of PerCom*, 2008.

[12] J. Li and F. Dabek, "F2F: Reliable storage in open networks," in *Proc. of IPTPS*, 2006.

[13] H. Yu, M. Kaminsky, and A. D. Flaxman, "SybilGuard: Defending against sybil attacks via social networks," in *Proc. of Sigcomm*, 2006.

[14] E. Pennisi, "How did cooperative behavior evolve?" *Science*, 2005.

[15] B. Viswanath, A. Post, K. Gummadi, and A. Mislove, "An analysis of social networkbased sybil defenses," in *Proc. of SIGCOMM*, 2010.

[16] M. Srivatsa, L. Xiong, and L. Liu, "TrustGuard: Countering vulnerabilities in reputation management for decentralized overlay networks," in *Proc. of WWW*, 2005.

[17] 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.

[18] "Bittorrent," http://en.wikipedia.org/wiki/Bittorrent.

[19] Akamai, http://www.akamai.com/.

[20] PPTV, http://www.pptv.com/.

[21] C. Perkins, E. Belding-Royer, and S. Das, "RFC 3561: Ad hoc on demand distance vector (AODV) routing," 2003.

[22] eBay, http://www.ebay.com.

[23] "Amazon.com," http://en.wikipedia.org/wiki/Amazon.com.

[24] R. Chen, E. K. Lua, and Z. Cai, "Bring order to online social networks," in *Proc. of INFOCOM*, 2011.

[25] 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.

[26] Z. Li, H. Shen, and K. Sapra, "Leveraging social networks to combat collusion in reputation systems for peer-to-peer networks," in *Proc. of IPDPS*, 2011.

[27] V. Srivastava, J. Neel, A. B. MacKenzie, R. Menon, L. A. Dasilva, J. E. Hicks, J. H. Reed, and R. P. Gilles, "Using game theory to analyze wireless ad hoc networks," *IEEE Communications Surveys and Tutorials*, 2005.

[28] P. D. Straffin, *Game theory and strategy*. The mathematical association of Amecica, 1993.

[29] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki, "Dandelion: Cooperative content distribution with robust incentives." in *Proc. of USENIX ATC*, 2007.

[30] K. J. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems." *ACM Comput. Surv.*, vol. 42, no. 1, 2009.

[31] Y. L. Sun and Y. Liu, "Security of online reputation systems: The evolution of attacks and defenses." *IEEE Signal Process. Mag.*, vol. 29, no. 2, pp. 87–97, 2012.

[32] Z. Li, H. Shen, and K. Sapra, "Leveraging social networks to combat collusion in reputation systems for peer-to-peer networks." in *Proc. of IPDPS*, 2011.

[33] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: Membership, growth, and evolution," in *Proc. of KDD*, 2006.

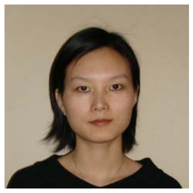
[34] A. Fast, D. Jensen, and B. N. Levine, "Creating social networks to improve peer-to-peer networking," in *Proc. of KDD*, 2005.

[35] A. Iamnitchi, M. Ripeanu, and I. T. Foster, "Small-world file-sharing communities," in *Proc. of INFOCOM*, 2004.

[36] M. Mcpherson, "Birds of a feather: Homophily in social networks," *Annual Review of Sociology*, vol. 27, no. 1, 2001.

[37] "Planetlab," http://www.planet-lab.org/.

Kang Chen Kang Chen received the BS degree in Electronics and Information Engineering from Huazhong University of Science and Technology, China in 2005, and the MS in Communication and Information Systems from the Graduate University of Chinese Academy of Sciences, China in 2008. He is currently a Ph.D student in the Department of Electrical and Computer Engineering at Clemson University. His research interests include mobile ad hoc networks and delay tolerant networks.



Haiying Shen 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 grid and cloud computing. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE and a member of the ACM.

Karan Sapara 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.

Guoxin Liu Guoxin Liu received the BS degree in BeiHang University 2006, and the MS degree in Institute of Software, Chinese Academy of Sciences 2009. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include Peer-to-Peer, CDN and online social networks.