Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

An efficient incentive scheme with a distributed authority infrastructure in peer-to-peer networks

Zhan Zhang[a], Shigang Chen[b,*], Zhen Mo[b], MyungKeun Yoon[c]

[a] Juniper Networks, Sunnyvale, CA 94089, USA
[b] Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611, USA
[c] Department of Computer Engineering, Kookmin University, Seoul, 136-702, Republic of Korea

ARTICLE INFO

Article history:
Received 1 September 2011
Received in revised form
1 August 2012
Accepted 9 August 2012
Available online 17 August 2012

Keywords:
Peer-to-peer networks
Incentive scheme
Threshold cryptography

ABSTRACT

Today's peer-to-peer networks are designed based on the assumption that the participating nodes are cooperative, which does not hold in reality. Incentive mechanisms that promote cooperation must be introduced. However, the existing incentive schemes (using either reputation or virtual currency) suffer from various attacks based on false reports. Even worse, a colluding group of malicious nodes in a peer-to-peer network can manipulate the history information of its own members, and the damaging power increases dramatically with the group size. Such malicious nodes/collusions are difficult to detect, especially in a large network without a centralized authority. In this paper, we propose a new distributed incentive scheme, in which the amount that a node can benefit from the network is proportional to its contribution, malicious nodes can only attack others at the cost of their own interests, and a colluding group cannot gain advantage by cooperation regardless of its size. Consequently, the damaging power of colluding groups is strictly limited. The proposed scheme includes three major components: a distributed authority infrastructure, a key sharing protocol, and a contract verification protocol.

1. Introduction

A node in a peer-to-peer network is allowed to consume resources from other nodes, and is also expected to share its resources with the community. However, today's peer-to-peer networks suffer from the problem of free-riders, who consume resources in the network without contributing anything in return. Originally it was hoped that users would be altruistic, "from each according to his abilities, to each according to his needs". In practice, however, altruism breaks down as networks grow larger and include more diverse users. This leads to a "tragedy of the commons", where individual players' self interest causes the system to collapse.

To reduce free-riders, the systems have to incorporate incentive schemes to encourage cooperative behavior [27]. Some recent works [15,28,1,19,7,14] propose *reputation* based trust systems, in which each node is associated with a reputation established based on the feedbacks from others that it has made transactions with. The reputation information helps users to identify and avoid malicious nodes. An alternative is *virtual currency* schemes [24,10,22,26], in which each node is associated with a certain amount of money. Money is deducted from the consumers of a service, and transferred to the providers of the service after each transaction.

Both types of schemes rely on authentic measurement of service quality and unforgeable reputation/money information. Otherwise, selfish/malicious nodes may gain an advantage based on false reports. For example, a consumer may falsely claim to have not received service in order to pay less or defame others. More seriously, malicious nodes may collude in cheating in order to manipulate their information. Several algorithms are proposed to address these problems. They either analyze statistical characteristics of the nodes' behavior patterns and other nodes' feedbacks [19,13], or remove the underlying incentive for cheating [2]. However, in order to apply these algorithms, the nodes' history information must be managed by a central authority, which is not available in typical peer-to-peer networks.

Some other works [12,9] find circular service patterns based on the history information shared among trusted nodes. Each node in a service circle has chance to be both a provider and a consumer. However, the communication overhead for discovering service circles is very high, which makes these schemes not scalable. In addition, nodes belonging to different interest groups have little chance to cooperate because service circles are unlike to form among them.

* Corresponding author.
E-mail addresses: zhanz@juniper.net (Z. Zhang), sgchen@cise.ufl.edu (S. Chen), zmo@cise.ufl.edu (Z. Mo), mkyoon@kookmin.ac.kr (M. Yoon).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.08.003

Also related is the research on sybil attacks [8]. The notion of sybilproofness is formalized in [4], which further proves that there is no symmetric sybilproof reputation function and provides the conditions for sybilproofness. A sybilproof transitive trust protocol is proposed in [16] to enable the interaction indirectly through a chain of credit or "trust" links. It assumes the common knowledge of all participants and each user maintains a trust account for every other user. This generic protocol is not specifically designed to address the free-ride problem in peer-to-peer networks. The sybilproof indirect reciprocity mechanism in [11] forms a contribution graph among nodes through links of contribution (e.g., one peer has served another). It models the contribution transitivity as a routing problem. Through transitive paths, it allows a node to contribute to a set of peers and transfer these contributions through cycles in the graph, such that it can be served by other peers. In this paper, we use a more generic model where any past contribution by a consumer can be applied to the service of any provider without the need of transitive paths.

We focus on how to design an effective incentive scheme suitable for P2P systems, which have no central authority to maintain individual nodes' history information. The major contributions are listed below.

(1) We propose a new distributed incentive scheme, which combines reputation and virtual money. It is able to strictly limit the damage caused by malicious nodes and their colluding groups. The following features distinguish our scheme from others.
   - *The benefit that a node can get from the system is limited by its contribution to the system.*
   - *The members in a colluding group cannot increase their total money or aggregate reputation by cooperation, regardless of the group size.*
   - *Malicious nodes can only attack others at the cost of their own interest.*

(2) We design a distributed authority infrastructure to manage the nodes' history information with low overhead and high security.

(3) We design a key sharing protocol and a contract verification protocol based on the threshold cryptography to implement the proposed distributed incentive scheme.

The rest of the paper is organized as follows. Section 2 provides the motivation for our work. Section 3 defines the system model. Section 4 proposes a distributed authority infrastructure. Section 5 presents our distributed incentive scheme. Section 6 studies the properties of the proposed scheme. Section 7 discusses several important issues. Section 8 evaluates the scheme by simulations. Section 9 draws the conclusion.

2. Motivation

2.1. Limitation of prior work

Any node in a peer-to-peer network is both a service provider and a service consumer. It contributes to the system by working as a provider, and benefits from the system as a consumer. A transaction is the process of a provider offering a service to a consumer, such as supplying a video file. The purpose of an incentive scheme is to encourage the nodes to take the role of providers. However, neither reputation systems [15,28,1,19,7,14] nor virtual currency systems [24,10,22] can effectively prevent malicious nodes, especially those in collusion, from manipulating their history information by using false service reports. Specifically, the existing schemes have the following problems.

Reputation inflation: In the reputation schemes, malicious nodes can work together to inflate each other's reputation or to defame innocent nodes, by which colluding nodes protect themselves from the complaints by innocent nodes as these complaints may be treated as noise by the systems.

Money depletion: In the virtual currency schemes, malicious nodes may launch attacks to deplete other nodes' money and paralyze the whole system. Without authentic reputation information, innocent nodes are not able to proactively select benign partners and avoid malicious ones.

Frequent complainer: In many incentive schemes, nodes will be punished if they complain frequently, which prevents malicious nodes from constantly defaming others at no cost. However, it also discourages innocent nodes from reporting frequent malicious acts because otherwise they would become frequent complainers.

Punishment scale: In most existing schemes, the scale of punishment is related to the service history of the transaction participants. Consequently, an innocent node may be subject to negative discrimination attacks [5] launched by nodes with excellent history.

2.2. Motivation

Punishing malicious nodes and limiting the damage caused by a colluding group are indispensable requirements of an incentive scheme that is able to deter bad behavior. There are two major kinds of bad behavior. First, a provider may *deceive* a consumer by providing less-than-promised service. Second, a consumer may *defame* a provider by falsely claiming the service is poor.

Consider how these problems are dealt with in real life. Before a transaction happens, the provider would want to know if the consumer has enough money to pay for the service, and the consumer would want to know the reputation of the provider. With such information, they can control the risk and decide whether to carry out the transaction or not. After the transaction, if the provider deceives, it will be sued by the consumer. Consequently, the malicious provider will build up a bad reputation, which prevents it from deceiving more consumers. Now consider when a consumer intentionally defames a provider. It does so only after it can show the evidence of a transaction, which requires it to pay money first. Consequently, defaming comes with a cost. The ability of the malicious consumer to defame others is limited by the amount of money it has.

Inspired by the observation above, we propose a new incentive scheme: *MARCH*, which is a combination of Money And Reputation sCHemes.

The basic idea behind the scheme is simple: each node is associated with two parameters: money and reputation. The providers earn money (and also reputation) by serving others. The consumers pay money for the service. If a consumer does not think the received service worth the money it has paid, it reports to an authority, specifying the amount of money it believes it has overpaid. If the authority can determine who is lying, the liar is punished. Otherwise, the authority freezes the money claimed to have been overpaid. The money will not be available to the provider and will not be returned to the consumer either, which eliminates any reason for the consumer to lie. If the provider is guilty, the consumer has the revenge and the provider's reputation suffers. If the provider is innocent, the consumer does it at a cost because after all it has paid the price of the transaction. In addition, the falsely-penalized provider will not serve it any more. The technical challenges are (1) how to establish a distributed authority for managing the money and reputation, (2) how to design the protocol of transaction that ensures authentic exchange of money/reputation information and allows the unsatisfied consumers to sue the providers, (3) how to analyze the properties of such a system, and (4) how to evaluate the system.

3. System model

The nodes in a P2P network fall in three categories: honest, selfish, and malicious. *Honest* nodes follow the protocol exactly, and they both provide and receive services. *Selfish* nodes will break the protocol only if they can benefit. *Malicious* nodes are willing to compromise the system by breaking the protocol even when they benefit nothing and may be punished. Selfish/malicious nodes may form colluding groups. There may exist a significant number of selfish nodes, but the malicious nodes are likely to account for a relatively small percentage of the whole network. At a certain time, all self/malcious nodes that break the protocol are called *dishonest* nodes. A node is said to be *rejected* from the system if it has too little money and too poor a reputation such that no honest providers/consumers will perform a transaction with it. We study the incentive scheme in the context of DHT-based P2P networks, e.g., [20,17,18,25]. We assume the routing protocol is robust, ensuring the reliable delivery of messages in the network [3]. We also assume the networks have the following properties.

- *Random, non-selectable identifier*: A node can not select its identifier, which should be arbitrarily assigned by the system. This requirement is essential to defending the Sybil attack [8]. One common approach is to hash a node's IP address to derive a random identifier for the node [20].
- *Public/private key pair*: Each node A in the network has a public/private key pair, denoted as P_A and S_A respectively. A trusted third party such as PKI is needed to issue public-key certificates. The trusted third party is used off-line once per node for certificate issuance, and it is not involved in any transaction.

4. Authority infrastructure

4.1. Delegation

Who will keep track of the money/reputation information in a P2P network? In the absence of a central authority for this task, we design a distributed authority infrastructure. Each node A is assigned a *delegation*, denoted as D_A, which consists of k nodes picked pseudo-randomly. For example, we can apply k hash functions, i.e., $\{h_1, h_2, \dots, h_k\}$, on the identifier of node A to derive the identifiers of nodes in D_A. If a derived identifier does not belong to any node currently in the network, the "closest" node is selected. For example, in [20], it will be the node clockwise after the derived identifier on the ring. The j-th element in D_A is denoted as $D_A(j)$.

D_A keeps track of A's money/reputation. Any anomaly in the information stored at the delegation members may indicate an attempt to forge data. The information is legitimate only if the majority of the delegation members agree on it. Therefore, as long as the majority of the delegation members are honest, the information about node A cannot be forged. Such a delegation is said to be *trustworthy*. On the other hand, if at least half of the members are dishonest, then the delegation is *untrustworthy*.

The delegation members are appointed pseudo-randomly by the system. A node cannot select its delegation members, but can easily determine who are the members in its or any other node's delegation. To compromise a delegation, the malicious/selfish nodes from a colluding group must constitute the majority of the delegation. Unless the colluding group is very large, the probability for this to happen is small because the identifiers of the colluding nodes are randomly assigned by the system and the identifiers of the delegation are also randomly assigned. Let m be the size of a colluding group and n be the total number of nodes in the system. The probability for t out of k nodes to be in the colluding group is

$$P\left(t, k, \frac{m}{n}\right) = \binom{k}{t} \left(\frac{m}{n}\right)^t \left(1 - \frac{m}{n}\right)^{k-t}$$

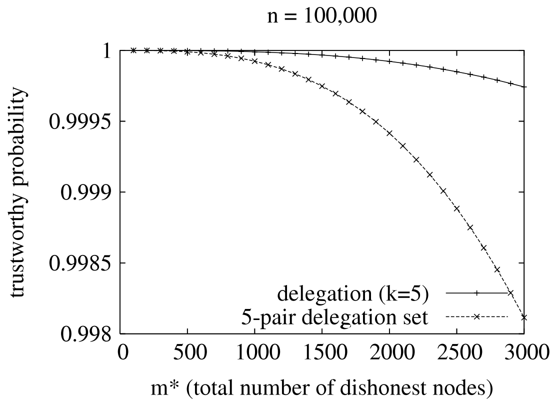

Fig. 1. When $m^* = 3000$, the trustworthy probability for a delegation and 5-pair delegation set are 99.975% and 99.815% respectively. Even if a delegation/k-pair delegation set is not trustworthy, it may not be compromised because very unlikely a single colluding group can control the majority of them.

where $P\left(t, k, \frac{m}{n}\right)$ denotes the probability of t successes in k trials in a Binomial distribution with the probability of success in any trial being $\frac{m}{n}$. Let m^* be the total number of distinct nodes in all colluding groups, also including all malicious nodes. The probability of a delegation being trustworthy is at least $\sum_{t=0}^{\lfloor \frac{k}{2} \rfloor} P\left(t, k, \frac{m^*}{n}\right)$. We plot the trustworthy probability with respect to m^* when $k = 5$ in Fig. 1 (the upper curve). In order to control the overhead, we shall keep the value of k small.

4.2. k-pair trustworthy set

A transaction involves two delegations, one for the provider and the other for the consumer. They have to cooperate in maintaining the money and reputation information, and avoiding any fraud. To facilitate the cooperation, we introduce a new structure, called *k-pair delegation set*, consisting of k pairs of delegation members. Suppose node A is the provider and node B is the consumer. The ith pair is $(D_A(i), D_B(i))$, $\forall i \in [1 \dots k]$, and the whole set is

$$\{(D_A(1), D_B(1)), (D_A(2), D_B(2)), \dots, (D_A(k), D_B(k))\}.$$

If both $D_A(i)$ and $D_B(i)$ are honest, the pair $(D_A(i), D_B(i))$ is trustworthy. If the majority of the k pairs are trustworthy, the whole set is trustworthy. It can be easily verified that the probability for the whole set to be trustworthy is

$$\sum_{t=0}^{\lfloor \frac{k}{2} \rfloor} P\left(t, k, 2\frac{m^*}{n} - \left(\frac{m^*}{n}\right)^2\right).$$

We plot the trustworthy probability for the whole set with respect to m^* in Fig. 1 (the lower curve).

5. MARCH: a distributed incentive scheme

5.1. Money and reputation

With the distributed authority designed in the previous section, the following information about a node A is maintained by a delegation of k nodes.

Total money (TM_A): It is the total amount of money paid *by* others to node A minus the total amount of money paid *to* others by A in all previous transactions. The universal refilled money (Section 7.2) will also be added to this variable.

Overpaid money (OM_A): It is the total amount of money overpaid by consumers. A consumer pays money to node A before a service. If the service contract is not fulfilled by the transaction, the consumer may file a complaint, specifying the amount of money that it has overpaid. This amount cannot be greater than what the consumer has paid.

When a new node joins the network, its total money and overpaid money are initialized to zero. From TM_A and OM_A, we define the following two quantities.

Available money (m_A): It is the amount of money that node A can use to buy services from others.

$$m_A = TM_A - OM_A. \tag{1}$$

Reputation (r_A): It evaluates the quality of service (with respect to the service contracts) that node A has provided.

$$r_A = \begin{cases} \dfrac{TM_A - OM_A}{TM_A} & \text{if } TM_A \neq 0 \\ 1 & \text{if } TM_A = 0. \end{cases} \tag{2}$$

For example, if $TM_A = 500$ and $OM_A = 10$, then A's available money is 490, i.e., $m_A = 490$, and its reputation is 0.98, i.e., $r_A = 0.98$.

To track every node's available money and reputation, we propose a set of protocols. Consider a transaction, in which Alice (A) is the provider and Bob (B) is the consumer. The transaction consists of five sequential phases.

- *Phase one: contract negotiation.* Alice and Bob negotiate a service contract.
- *Phase two: contract verification.* Through the help of their delegations, Alice and Bob verify the authenticity of the information claimed in the contract.
- *Phase three: money transfer.* The amount of money specified in the contract is transferred from Bob's account in D_B to Alice's account in D_A.
- *Phase four: contract execution.* Alice offers the service to Bob based on the contract specification.
- *Phase five: prosecution.* After the service, Bob provides feedback reflecting the quality of service offered by Alice.

5.2. Phase one: contract negotiation

Suppose Bob has received a list of providers through the lookup routine of the P2P network. Each provider specifies its reputation and its price for the service. Bob wants to minimize his risk when deciding which service provider he is going to use.

Let L_A be the price specified by Alice and G_B be the fair price estimated by Bob himself. According to the definition, r_A can roughly be used as a lower bound on the probability of Alice being honest. Intuitively, the probability for Bob to receive the service is at least r_A, and the probability for Bob to waste its money L_A is at most $(1 - r_A)$. We define the benefit for Bob to have a transaction with Alice as $G_B \times r_A - L_A \times (1 - r_A)$. We further normalize it as

$$R = r_A - \frac{L_A}{G_B}(1 - r_A). \tag{3}$$

To avoid excessive risk, Bob takes Alice as a potential provider if R is greater than a threshold value T. The use of threshold helps the system reject dishonest providers with poor reputation. Among all potential providers, Bob picks the one with the highest normalized benefit.

Both the value of L_A and the value of r_A are given by the provider A. If L_A is set too high, R will be small and the provider runs the risk of not being picked by Bob. Providers with a poor reputation can improve their R values by setting their prices low. In this way, they can recover their reputation by selling services at lower prices. If Alice lies about its r_A, she will be caught in the next phase and be punished.

Now suppose Bob chooses Alice as the best service provider. They have to negotiate a service contract, denoted as c, in the following format.

$$\langle A, B, S, Q, L, Seq_A, Seq_B, r_A, m_B \rangle$$

where A, B, S, Q, and L specify the provider, the consumer, the service type, the service quality, and the service price respectively. Seq_A and Seq_B are the contract sequence numbers of Alice and Bob, respectively. After the transaction, Alice and Bob each increase their sequence numbers by 1. The values of r_A and m_B in the contract will be verified by the delegations in the next phase.

As an example, if $S = Storage$, $Q = 200G$, and $L = 5$, the contract means that Alice offers storage with size 200G to Bob, and as return, the amount of money Bob must pay is 5.

Note that the Eq. (3) is a heuristic method recommended to end user, and does not need to be system-wide. Each peer can modify it based on its own experiences and expectation, and it does not have impact on other nodes in the system.

5.3. Phase two: contract verification

After negotiating a contract, Alice and Bob should exchange an authenticatable contract proof, so that Alice is able to activate the money transfer procedure and Bob is granted the prosecution rights. In addition, the information in the contract, e.g., r_A and m_B, should be verified by the delegations of Alice and Bob.

We use the notation $[x]_y$ for the digital signature signed on message x with key y and $\{x\}_y$ for the cipher text of message x encrypted with key y. After Phase two, if the contract is verified by the delegations, Alice should have the following contract proof

$$c_A = [c]_{S_B}.$$

c_A should not be produced by Bob, who may lie about m_B. Instead, Alice must receive c_A from Bob's delegation after the members confirm the value of m_B. Bob has k delegation members. Each of them will produce a "piece" of c_A and send it to Alice, who will combine the "pieces" into a valid contract proof. Similarly, Bob must receive the following contract proof from Alice's delegation

$$c_B = [c]_{S_A}.$$

The contract proofs will be used by Alice for money transfer and by Bob for prosecution.

It is important to ensure that either both Alice and Bob, or none of them, receive the contract proofs. Otherwise, dishonest nodes may take advantage of it. It can be shown that ensuring both or neither one receives her/his contract proof is impossible without using a third party (the delegation of Alice or Bob in this case).

Key sharing protocol

A k-member delegation is not a *centralized* third party. One possible approach for producing a contract proof by a delegation is to use threshold cryptography [6]. A (k, t) threshold cryptography scheme allows k members to share the responsibility of performing a cryptographic operation, so that any subgroup of t members can perform this operation successfully, whereas any subgroup of less than t members can not. For digital signature, k shares of the private key are distributed to the k members. Each member generates a partial signature by using its share of the key. After a combiner receives at least t partial signatures, it is able to compute the signature, which is verifiable by the public key. An important property is that less than t compromised members cannot produce a verifiable signature on a false message.

In our case, the problem is to produce c_B (or c_A) by the k-member delegation of Alice (or Bob). We employ a $\left(k, \left\lfloor \frac{k}{2} \right\rfloor + 1\right)$ threshold cryptography scheme to produce the contract proof.

Alice distributes shares $S_A(i)$ of her private key S_A to her delegation members $D_A(i)$, which will produce partial signatures $[c]_{S_A(i)}$ and forward them to Bob for combination. As long as the delegation of Alice is trustworthy, Bob will receive enough correct partial signatures to compute a verifiable contract proof, while the false partial signatures generated by the compromised delegation members will not yield any verifiable proof.

When applying threshold cryptography, we have to defend against dishonest nodes, which may intentionally distribute incorrect secret shares. The incorrect partial signatures cannot yield a valid signature. We propose a protocol for distributing the key shares. Take Alice as an example. The protocol guarantees that either all delegation members receive the correct shares, or they all detect that Alice is dishonest.

Step 1: Alice sends a key share $S_A(i)$ to each delegation member $D_A(i)$, encrypted by the member's public key $P_{D_A(i)}$. The messages are shown below.

$MSG1\ Alice \rightarrow D_A(i) : [\{S_A(i)\}_{P_{D_A(i)}}]_{S_A}, \quad \forall D_A(i) \in D_A.$

Step 2: After all members receive their key shares, they negotiate a common random number s (possibly by multi-party Diffie–Hellman exchange with authentication). Each member sends the number s as a challenge to Alice, signed by the member's private key and then encrypted by Alice's public key.

$MSG2\ D_A(i) \rightarrow Alice : \{[s]_{S_{D_A(i)}}\}_{P_A}, \quad \forall D_A(i) \in D_A.$

Step 3: Alice signs s with $S_A(i)$ and then with S_A before sending it back to $D_A(i)$.

$MSG3\ Alice \rightarrow D_A(i) : [[s]_{S_A(i)}]_{S_A}, \quad \forall D_A(i) \in D_A.$

Step 4: After authentication, if the received $[s]_{S_A(i)}$ value matches the locally computed one, $D_A(i)$ forwards the message to all other members in D_A.[1]

$MSG4\ D_A(i) \rightarrow D_A(j) : [[s]_{S_A(i)}]_{S_A}, \quad \forall D_A(j) \in D_A.$

Otherwise, $D_A(i)$ files a certified complaint to other members.

$MSG5\ D_A(i) \rightarrow D_A(j) : [\text{“}S_A(i) \text{ is invalid”}]_{S_{D_A(i)}}, \quad \forall D_A(j) \in D_A.$

Step 5: $D_A(i)$ needs to collect $[s]_{S_A(j)}$, $\forall D_A(j) \in D_A$, which are the partial signatures on s. If it receives $MSG4$ $[[s]_{S_A(j)}]_{S_A}$ from $D_A(j)$, the value of $[s]_{S_A(j)}$ is in the message. If it receives $MSG5$ from $D_A(j)$, there are two possibilities: either Alice or $D_A(j)$ is dishonest. To resolve this situation, $D_A(i)$ forwards the certified complaint to Alice. If Alice challenges the complaint, she must disclose the correct value of $S_A(j)$ to $D_A(i)$ in the following message (then $D_A(j)$ can learn $S_A(j)$ from $D_A(i)$).

$MSG6\ Alice \rightarrow D_A(i) : [\{S_A(j)\}_{P_{D_A(i)}}]_{S_A}.$

Learning $S_A(j)$ from this message, $D_A(i)$ can compute $[s]_{S_A(j)}$. After $D_A(i)$ has all k partial signatures on s, it can determine that Alice is honest if any $\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right)$ partial signatures produce the same signature $[s]_{S_A}$, which can be verified by Alice's public key. Otherwise, Alice must be dishonest.

Since the value of k is typically set small (e.g. 5) and the key distribution is performed once per node, the overhead of the above protocol is not significant.

A delegation member $D_A(i)$ can acquire another delegation member $D_A(j)$'s key share only when $D_A(j)$ dishonestly complains about Alice and in response, Alice discloses $D_A(j)$'s key share in an effort to prove that she is honest whereas $D_A(j)$ lies. Note that $D_A(i)$ cannot deceive Alice to expose the private share of $D_A(j)$ because only $D_A(j)$ itself can generate its certified complaint. Thus, the total number of distinct shares exposed by Alice is no larger than the number of dishonest members. Our security model ensures the validity of MARCH only when a majority of the delegation members are honest. On the other hand, if Alice detects that her delegation set is no longer trustworthy, she can request the trusted third party for rekeying and identification reassignment to change her delegation.

Theorem 1. *The key sharing protocol ensures that all delegation members will either obtain the correct shares of Alice's private key or detect Alice's fraud.*

Proof. First of all, any node cannot deny the messages it has sent to others or falsely declare it has received some messages from others, because all messages in the protocol are signed by the corresponding nodes with their private keys.

Consider the first case that Alice is honest. All delegation members can obtain the correct shares in Step 1. In the meantime, only if a complaint is signed by a delegation member, Alice will disclose the corresponding share to challenge the complaint. If Alice is honest, only dishonest members may issue certified complaints. Thus, the total number of distinct shares exposed by Alice is no larger than the number of dishonest members.

Next consider the second case that Alice is dishonest. Alice may try to deceive the delegation in two possible ways. One way is that Alice does not send shares to some delegation members $D_A(i)$, which can be easily detected by $D_A(i)$ when it receives $MSG3$ from Alice or $MSG4$ from other delegation members. Subsequently, $D_A(i)$ will file a certified complaint ($MSG6$). If Alice discloses the correct share ($MSG7$) to challenge the complaint, $D_A(i)$ can obtain its share from other members; otherwise, honest members are certain that Alice is dishonest, and will punish her.

The other possible way for Alice to deceive is to distribute incorrect shares to some members $D_A(i)$ in $MSG1$. There are three possible outcomes when $MSG3$ is processed by $D_A(i)$. (1) The partial signature in $MSG3$ matches the locally computed one. Subsequently, $MSG3$ is forwarded to all other members by $D_A(i)$. Then all honest members can detect Alice's fraud in Step 5 because, in addition to $[s]_{S_A(i)}$, there are $\frac{\lfloor k \rfloor}{2}$ other partial signatures that cannot be used to compute the signature $[s]_{S_A}$. (2) The partial signature in $MSG3$ does not match the locally computed one. $D_A(i)$ will detect Alice's fraud in Step 4 because of the inconsistency between $MSG1$ and $MSG3$. It will forward two inconsistent messages from Alice to all other members in $MSG5$. Consequently, all members learn the inconsistency and punish Alice. (3) Alice does not send $MSG3$ to $D_A(i)$ at all. This can be handled in a way similar to the previous case that $D_A(i)$ does not receive $MSG1$ from Alice. □

Contract verification protocol

Both Alice and Bob must register the contract with their delegations so that the money transfer and the optional prosecution can be performed through the delegations at later times. The delegations must verify the information claimed by Alice and Bob in the contract and generate the contract proofs that Alice and Bob need in order to continue their transaction. We design a contract verification protocol to implement the above requirements. The protocol consists of four steps, depicted in Fig. 2 (the left portion), and the number of messages is $O(k)$ for normal cases.

A procedure call is denoted as $x.\mathbf{y}(z)$, which means to invoke procedure y at node x with parameter(s) z. If x is a remote node, a signed message carrying z must be sent to x first.

Step 1: Alice sends the contract c and a digital signature c' to the delegation D_A for validation. c' may be a signature of the contract concatenating the identifier of the receiver, i.e., $c' = [c|D_A(i)]_{S_A}$. Bob does the same thing.

[1] Note that $D_A(i)$ knows s and learns $S_A(i)$ from $MSG1$.

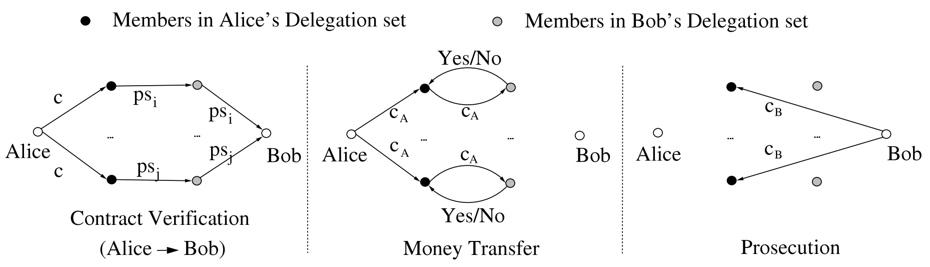

Fig. 2. Protocols for contract verification and exchange (left), money transfer (middle), and prosecution (right).

```
Alice.**SendContract**(Contract $c$)
1.    for  $i = 1$ to $k$ do
2.        compute Signature $c'$
3.        $D_A(i)$.ComputePartialProof($c, c'$)
```

In addition, we have an extra similar procedure as follows, which is used by Alice's delegation members to actively request a contract with a specific sequence number s.

```
Alice.RetrieveContract(Seq $s$, Delegation $d$)
1.    if  exists $c$ with $c.Seq_A = s$
2.        compute Signature $c'$ for $d$
3.        $D_A(i)$.ComputePartialProof($c, c'$)
4.    else send error to $d$
```

Step 2: Then the delegation member $D_A(i)$ verifies the reputation claimed by Alice in the contract (denoted as $c.r_A$), save the contract history in *ContractList* and computes a partial signature (denoted as ps_i) on the contract with its key share established by the previous protocol.

```
$D_A(i)$.ComputePartialProof(Contract $c$, Signature $c'$)
1. if $r_A \geq c.r_A$ then
2.     insert(ContractList, $c$, $c'$)
3.     $ps_i = [c]_{S_A(i)}$
4.     $D_B(i)$.DeliverPartialProof($c$, $ps_i$)
5. else punish(A)
```

Line 1 verifies whether $c.r_A$ is over-claimed or not. Please note that in order to avoid the punishment due to discrepancy between r_A (that Alice declared) and the value maintained by the delegations, Alice should declare its r_A conservatively. The *insert* operation of Line 2 saves the contract c as well as c' in *ContractList* for later use in Step 3. The signature c' will be used in a procedure *detect*(). Line 3 produces a partial signature on the contract by using $S_A(i)$. Line 4 sends the partial signature to the ith member of D_B. If $c.r_A$ is over-claimed, Alice will be punished at Line 5.

The delegation members in D_B execute a similar procedure except that the condition in Line 1 should be $m_B \geq c.m_B$.

Step 3: When $D_B(i)$ receives the contract c and the partial signature sp_i from $D_A(i)$, it executes the following procedure.

```
$D_B(i)$.DeliverPartialProof(Contract $c$, PartialSignature $ps_i$)
1. if not lookup(ContractList, $c$) then
2.     Bob.RetrieveContract($c.Seq_B$, selfid)
3. if lookup(ContractList, $c$) then
4.     Bob.ProcessPartialProof($ps_i$)
5. else
6.     detect()
```

If $D_B(i)$ cannot find c in its ContractList (Line 1), it actively queries Bob to retrieve the contract c (Line 2). Then $D_B(i)$ looks up c in its ContractList again (Line 3). Note that in *Bob*.**RetrieveContract** ($c.Seq_B$, *selfid*), Bob may send a different contract from c with the same sequence number Seq_B, and in this case $D_B(i)$ will not be able to find c in its ContractList. If c exists, Bob has announced the exact same contract as Alice does, and $D_B(i)$ forwards the partial signature ps_i to Bob. Otherwise, $D_B(i)$ believes either Alice or Bob is dishonest, and it will will invoke the detect() procedure to detect the special case of a malicious Bob forging the contract. Once Bob is detected to be dishonest, the delegation can regard him as a liar, and omit the detection procedure in the future suspicious transactions, indicating the detection procedure needs to be invoked only once for each dishonest node.

In the following, we will present the design details of the detect() procedure, and then provide the correctness proof. The delegation member $D_B(i)$, which has received two different contracts from Bob and $D_A(i)$, must handle two possible cases. One case is that $D_B(i)$ has received a contract with the sequence number $c.Seq_B$ from Bob, and the other is that $D_B(i)$ has never received such a contract from Bob. In the former case, $D_B(i)$ stops the verification procedure immediately. Then it tries to detect whether Bob is lying or not by sending Bob's signature c' to all other delegation members in D_B. If a member finds that c' is different from Bob's signature that it receives directly from Bob, it sends its version of Bob's signature to all other members. Otherwise, the member discards the signature from $D_B(i)$. In the latter case, $D_B(i)$ sends a special request, denoted *REQ*, with the sequence number $c.Seq_B$ to all other members in D_B. If a member has already received the contract from Bob with the specified sequence number, it sends the corresponding signature c' to all other members after receiving *REQ*. Otherwise, the member discards the request *REQ*. In both cases, any member that has received two different versions of Bob's signature c' punishes Bob and refuses to participate in the rest of transaction for the contract. In addition, for the latter case, $D_B(i)$ refuses to proceed with the verification if no replies are received from other members, or punishes Bob but still continues the verification procedure using the the contract retrieved from other members if all of the signatures received are the same.

We show that, having received conflicted contracts, if a delegation member simply stops the verification procedure without invoking the detect() routine, Bob will be able to break the protocol. Suppose k is equal to 3, $D_B(1)$ is a friend of Bob, and both $D_B(2)$ and $D_B(3)$ are honest. Bob can break the protocol by sending $D_B(2)$ a correct contract while sending $D_B(3)$ a forged contract (for instance, with lower price $c.L$) or not sending the contract to $D_B(3)$ at all. Because $D_B(1)$ is Bob's friend, it may forward the partial signature $ps_A(1)$ from $D_A(1)$ to Bob, but not send the partial signature $ps_B(1)$ to $D_A(1)$. Then, Bob can collect two partial signatures $ps_A(1)$ and $ps_A(2)$ because $D_B(2)$ cannot detect Bob's fraud and will forward the partial signature $ps_A(2)$ to Bob, while Alice can only get one signature $ps_B(2)$ from $D_A(2)$. Therefore, Bob can compute the contract proof signed by Alice, while Alice cannot compute the proof signed by Bob. In our protocol, this problem is addressed by $D_B(3)$ invoking the detect() routine after receiving the contract from $D_A(3)$. The detect() routine guarantees that either $D_B(2)$ detects Bob's fraud or $D_B(1)$ forwards the partial signature of the contract retrieved from $D_B(2)$.

Step 4: After Alice (Bob) receives t or more correct partial signatures, she (he) can compute the contract proof $c_A(c_B)$, which can be verified by using Bob's (Alice's) public key.

Theorem 2. *If k-pair delegation set of Alice and Bob is trustworthy, the contract verification protocol ensures that both Alice and Bob will receive the correct proofs, or neither one can receive a valid contract proof and the transaction is aborted.*

Proof. First, we prove that neither Alice nor Bob can deceive the authority. This is a symmetric protocol, so without losing generality we only consider Bob. Below we analyze the four possible ways that Bob may use to deceive the authority.

1. Bob over-claims its available money m_B. In this case, all honest members in D_B can detect Bob's fraud in Step 1, and punish Bob. In the meantime, these members will neither forward the partial signature $[c]_{S_B(i)}$ to $D_A(i)$ nor deliver $[c]_{S_A(i)}$ to Bob, and consequently the transaction will be aborted.
2. Bob modifies the contract specification, for example, by lowering the transaction price $c.L$ in order to pay less for the transaction. He sends the same modified contract to D_B. In Step 3, all members in D_B learn that the contract presented by Bob is different from that presented by Alice, and they will invoke the detect() procedure. In this case, all honest delegation members will stop contract verification immediately, but they will not punish Bob, because there is only one contract signature from Bob and either Alice or Bob may be lying.
3. Bob sends different modified contracts to the delegation members. Multiple delegation members will detect that the contracts from Bob and Alice are different, and they will invoke the detect() routine. At the end of the detection procedure, all members will learn that there are different contract signatures coming from Bob. Consequently, they will all punish Bob and abort the transaction.
4. Bob does not send the contracts to some (or all) delegation members. In this case, a member $D_B(i)$ that does not receive the contract from Bob will send the request *REQ* to all other members. If no other member receives the contract from Bob, $D_B(i)$ will receive no reply back. It will refuse to continue the verification process, but will not punish Bob because either Alice or Bob can be lying. Similarly, all other members will also stop the verification. Now if some other members have received the contracts from Bob, $D_B(i)$ will receive Bob's signatures in the replies from them, and it will continue the verification process using the contracts retrieved from other members. Therefore, no one stops the verification process.

In summary, we can see that all members in the trustworthy set will take the same action (continuing or stopping the contract verification process) in all four possible cases.

Next, we prove that dishonest members cannot deceive honest members in the trustworthy sets to stop the verification process if both Alice and Bob are honest. As we have discussed above, only in two cases will an honest delegation member, $D_A(i)$ or $D_B(i)$ stop the contract verification. One case is that the contract signatures (both Alice's and Bob's) received by the member in the detect() routine are not identical. The other case is that the member receives different contracts from Alice and Bob in Step 3. The former case happens only when Alice/Bob sign and distribute different contracts to delegation members dishonestly, while the latter case happens when both $D_A(i)$ and $D_B(i)$ are untrustworthy or when either Alice or Bob is dishonest. Therefore, if both Alice and Bob are honest, dishonest members cannot interrupt the verification process executed by other members in the trustworthy sets.

By Theorem 1 and the discussions above, if Alice and Bob are honest, both of them are able to collect no less than $\frac{\lfloor k \rfloor+1}{2}$ correct partial signatures, and compute the valid contract proofs. Otherwise, if either Alice or Bob is dishonest, the transaction is aborted. □

5.4. Phases three and four: money transfer and contract execution

Before providing the service, Alice requests its delegation to transfer money, which is shown in the middle portion of Fig. 2. Upon receiving a money transfer request from Alice, the delegation member $D_A(i)$ invokes the following procedure.

```
$D_A(i)$.**TransferMoneyProvider**(Contracts $c$, ContractProof $c_A$)
1. **if** valid($c, c_A$) and $D_B(i)$.**TransferMoneyConsumer**($c, c_A$)
2.     $TM_A = TM_A + c.L$
3. **else** verify()
```

In Line 1, both $D_A(i)$ and $D_B(i)$ need to validate the contract by using Bob's public key, which can be queried from Bob if it is not locally available. After validation, $D_A(i)$ increases Alice's earned money in Line 2.

Note that $D_B(i)$ may be malicious. If $D_A(i)$ cannot get a positive answer from $D_B(i)$, it must verify the validity of the contract further (Line 3), which can be designed as follows. $D_A(i)$ asks other members in D_A. If the majority of D_A have received a positive answer from D_B, the contract is considered to be valid ($D_B(i)$ is malicious). Otherwise, the contract is considered to be invalid and Alice is punished.

When $D_B(i)$ receives a money transfer request from $D_A(i)$, it performs the following operations.

```
$D_B(i)$.**TransferMoneyConsumer**(Contract $c$, ContractProof $c_A$)
1. **if** $valid(c, c_A)$ **then**
2.     **if** $m_B > c.L$ **then**
3.         $TM_B = TM_B - c.L$
4.         **return** true;
5.     **else**
6.         punish(B)
7.         **return** false;
8. **else return** false;
```

First, if the contract is valid (Line 1) and Bob has enough money to pay the service (Line 2), then Bob's spent money is increased and a positive answer is returned to $D_A(i)$ (Lines 3 and 4). Second, it is possible that the contract is valid but Bob does not have enough money. This happens when Alice and Bob are colluding nodes and Alice gets the contract proof c_A directly from Bob instead through her delegation. In such a case, Bob is punished and a negative answer is returned (Lines 6 and 7). Third, if the contract is invalid, a negative answer is returned (Line 8).

$D_A(i)$ and $D_B(i)$, $\forall i \in [1 \ldots k]$, perform a money transfer at most once for each contract. They keep track of the sequence numbers (Seq_A and Seq_B) of the last contract for which the money has been transferred. All new contracts have larger sequence numbers.

5.5. Phase five: prosecution

After Bob receives the service from Alice, if the quality of service specified in the contract is not met, Bob may issue a prosecution request to Alice's delegation, as illustrated in the right portion of Fig. 2. The request specifies the amount of money f that Bob thinks he has overpaid.

Upon receiving a prosecution request from Bob, if D_A cannot evaluate the service quality, it punishes both Alice and Bob by freezing the money overpaid by Bob. The procedure is given as follows.

```
$D_A(i)$.**Prosecution**(Contract $c$, ContractProof $c_B$, Overpaid $f$)
1. **if** $valid(c, c_B)$ and $f \le c.L$ **then**
2.     $OM_A = OM_A + f$
3.     notify($A$)
```

First $D_A(i)$ validates the prosecution request by checking if the contract proof is authentic (Line 1). If the contract is valid, it increases Alice's overpaid money by f (Line 2). Finally, it notifies Alice so that Alice is able to determine whether to sell service to Bob in the future.

Before wrapping up this section, we want to discuss how Alice/Bob should be punished if (s)he is detected to be malicious during the five phases in the transaction. The general rule of thumb is that if a delegation member finds Alice/Bob is malicious, it should punish Alice/Bob more severely than the case Alice/Bob is prosecuted against, e.g., deducting the double amount of the money specified in the contract, which in turn results in less available money and worse reputation. Note that a honest delegation member will only punish malicious nodes. Although there may exist dishonest delegation members, the wrong punishment intentionally performed by them cannot impact the real history of Alice/Bob as long as the majority of the delegation set are honest.

6. System properties and defense against various attacks

6.1. System properties

We study the properties of MARCH, which solves or alleviates the problems in the previous approaches.

First, according to the money transfer procedures in Section 5.4, transactions among members in the same colluding group cannot increase the total amount of available money of the group. We have the following property, which indicates that the malicious nodes cannot benefit by cooperation.

Property 1. *Regardless of its size, a colluding group cannot increase its members' money or reputation by cooperation without decreasing other members' money and/or reputation.*

Second, unlike some other schemes [5,19,23,21], MARCH does not maintain the history of any consumer's complaints, and does not punish frequent complainers. Thus, we have the following property.

Property 2. *If a consumer is deceived, it is not restricted by the system in any way from seeking prosecution against the malicious providers.*

Third, the overpaid money is not returned to the complaining consumer, which eliminates any reason for the consumer to lie if the consumer is not malicious. If the consumer is malicious and intends to defame the providers, it has to pay the price for the transactions before committing any harm, which serves as an automatic punishment. Consequently, its ability of defaming is limited by the money it has, which cannot be increased artificially by collusion, according to Property 1.

In addition, by Property 2, a deceived consumer can seek revenge with no restriction, which means a malicious provider cannot benefit from its action. We have the following properties.

Property 3. *A malicious provider cannot benefit by deceiving the consumers, and a malicious consumer will be automatically punished for defaming the providers.*

Property 4. *The maximum amount of loss for an innocent provider or consumer in a transaction with a malicious node is limited by the price specified in the contract.*

Property 3 removes financial incentives to cheat. A provider can make money only by serving others; a consumer will not be refunded for cheating. Property 4 makes sure that an innocent node will not be subject to negative discrimination attacks [5], in which nodes with excellent reputation can severely damage other nodes.

In summary, the malicious nodes cannot increase their power (in terms of available money) by cooperation, and they can only attack others at the cost of their own interests, i.e., money and/or reputation. Consequently, the total damage caused by the malicious nodes is strictly limited. They will eventually be rejected from the system due to poor reputation or be enforced to serve others for better reputation in order to stay in the system.

6.2. Defending against various attacks

In the following, we consider four different types of attacks launched by a colluding group [5].

Unfairly high ratings: The members of a colluding group cooperate to artificially inflate each other's reputation by false reports, so that they can attack innocent nodes more effectively. In MARCH, a colluding group can inflate the reputation of some members only by moving the available money from other members to them. According to Property 1, the total money in the group cannot be inflated through cooperation. Therefore, although some members' reputation can be made better, other members' reputation will become worse, making them ineffective in attacks.

Unfairly low ratings: Providers collude with consumers to "bad-mouth" other providers that they want to drive out of the market. Because MARCH requires all consumers to pay money for their transactions before they can defame the providers, the malicious consumers lose their money (and reputation) for "bad-mouthing", which in turn makes it harder for them to stay in the system.

Negative discrimination: A provider only discriminates a few specific consumers by offering services with much lowered quality than what the contract specifies. It hopes to earn some "extra" money without damaging its reputation since it serves most consumers honestly. In MARCH, a provider cannot make such "extra" money because of the prosecution mechanism and Properties 2 and 3.

Positive discrimination: A provider gives an exceptionally good service to a few consumers with high reputation and an average service to the remaining consumers. The strategy will work in an incentive scheme where a consumer's ability of affecting a provider's reputation is highly related to the consumer's own reputation, and vice versa. MARCH does not have this problem. The provider's reputation changes after a transaction is determined by how much money it receives for the service, not by the reputation of the consumer.

7. Discussion

In this section, we discuss other important issues on implementing MARCH.

7.1. Rewarding delegation members

The system should offer incentive for the delegation members to perform their tasks. A simple approach is for the provider and the consumer of a transaction to reward their delegation members with a certain amount of money, which should be less than the price of the transaction. More specifically, after the transaction, the provider A signs an incentive payment certificate and sends the certificate to every delegation member $D_A(i)$, which reduces TM_A by a certain amount and then forwards the certificate to its delegation members, where the certificate is authenticated and the money is deposited. The consumer pays its delegation members in a similar way. If a delegation member in D_A refuses to serve, node A can increase k to bring new members into D_A.

7.2. Money refilling

Because the overpaid money will be frozen forever, the total amount of available money in the whole system may decrease over the time. As a result, the system may enter into deflation and lack sufficient money for the providers and the consumers to engage in transactions. This problem can be addressed by *money refilling*. The delegation members of a node A will replenish the total money TM_A of the node at a slow, steady rate. In this way, a minimal amount of service is provided to all consumers, even the free-riders, at all time, which we believe is reasonable as this is the common practice of today's peer-to-peer systems such as BitTorrent. For additional service, a consumer has to contribute to the P2P network by also serving as a provider. At first sight, money refilling seems to allow a botnet to take advantage of the scheme by funneling immense amounts of money into a given peer and extract resources from the system. On the one hand, the overall resource that the botnet can extract is still restricted by it's own members' total money, and the botnet cannot exaggerate its power by cooperation to defect the system by Property 1. On the other hand, eventually a few nodes in the botnet have to harvest the very small money from each bot member in a relatively short time. Note that all transactions are performed through delegations, and the delegations of those few nodes will observe the dramatic increment of available money and reputation through a large number of transactions in a very short period. Such anomaly can be exposed to other nodes and taken into account during the contract negotiation phase to avoid potential attacks.

7.3. System dynamics and overhead

In a P2P network, nodes may join/leave the network at any time. When a node X leaves the network, its DHT table will be taken over by the closest neighbor X'. In MARCH, suppose X is a delegation member of A. After X leaves the network, X' will become a new member in A's delegation. In order to deal with abrupt departure, X' should cache the information kept at X, or it can learn the information from other delegation members after X leaves.

MARCH is designed generally for all DHT networks. The specifics of a particular DHT network may present opportunity to improve the approach for selecting the delegation members in Section 4.1. Take Chord [20] as an example. We can select a subset of the $\log n$ neighbors of node A as the delegation D_A. In this way, the maintenance of the delegation is free as Chord already maintains the neighbor set.

The communication overhead of a transaction (excluding the actual service) consists of $O(k)$ control messages, which are sent from the provider (consumer) via k pairs of delegation members to the consumer (provider) throughput direct TCP connections. This overhead is quite small comparing to the typical services such as downloading video files of many gigabytes or sharing storage for months. More importantly, the overhead does not increase with the network size, which makes MARCH a scalable solution, comparing with other schemes [12,9] whose overhead increases with the network size.

7.4. Key compromise and revocation

MARCH can rely on a trusted third party to handle key revocation/compromise issues. We stress that the trusted third party is only responsible for producing key certificates. It is not involved in the MARCH operations, which are fully decentralized. When the trusted third party issues a certificate to a node, e.g., Alice, it includes an expiration time determined by Alice. The key expiration and compromise are handled by Alice herself. When Alice finds that her key is expiring or potentially compromised, she actively requests the trusted third party for rekeying (or additionally, identification reassignment to change her delegation). Specifically, Alice initiates the process by sending the request to the trusted third party for a new key certificate in a secure channel protected by a separate key. After verification, the trusted third party requests all history information from Alice's delegation, and calculates TM_A and OM_A based on the majority rule if there are discrepancies. All the information will be included in Alice's new certificate, so that the new delegation of Alice can reconstruct Alice's history information. If an attacker compromises Alice's key, but stays silent and does not misbehave in any way, Alice may never be able to detect such a compromise. However, after rekeying, Alice will use a new key, and even a new delegation.

7.5. Light-weighed MARCH

We have discussed how to implement the trustworthy delegation to support the proposed incentive scheme assuming that there is no trustful nodes except for the trusted third party and the participants involved in a transaction do not trust each other. In reality, the restriction can be relaxed and the process can be further simplified. For example, if Alice and Bob are mutually trusted, most phases in a transaction can be skipped. In this case, Alice and Bob exchange the contract signatures directly, and the delegation set will only be involved in the money transfer phase. Over the long term, we can expect that nodes will have more and more trusted friends, and thus the transactions among them will take much less overhead.

We also want to point out that MARCH does not have any overhead on free service provided by altruistic members. In other words, whether MARCH should be applied to a transaction depends on the participants involved.

8. Simulation

In our simulations, the dishonest nodes fall into three categories with equal probability.

Category one: These nodes never offer services to others after receiving money, and always defame the providers after receiving services.

Category two: When these nodes find that they may be rejected from the system, they behave honestly. Otherwise, they behave in the same way as the nodes in category one.

Category three: When these nodes find that they may be rejected from the system, they behave honestly. Otherwise, they cheat their transaction partners with a probability taken from [0.5, 1] uniformly at random.

If not explicitly specified otherwise, the system parameters are set as follows. The number of nodes is 100,000 and k is 5. The average number of dishonest nodes is 1000. Initially, the total money for a node is 500, and the overpaid money is 0. The service price G estimated by the consumers is 10. The threshold T is 0.9. To satisfy the threshold requirement, the maximum selling price for a provider is denoted as max (max is the maximum value of L that keeps R above the threshold, calculated based on Eq. (3). If max is negative, then the node can no longer be a provider. If a dishonest node in Category two or three finds that its max value may become negative after additional malicious acts, it will behave honestly). The actual selling price is a random number taken uniformly from (0, max]. If a node can neither be a provider (due to poor reputation), nor be a consumer (due to little money), it is said to be rejected from the system.

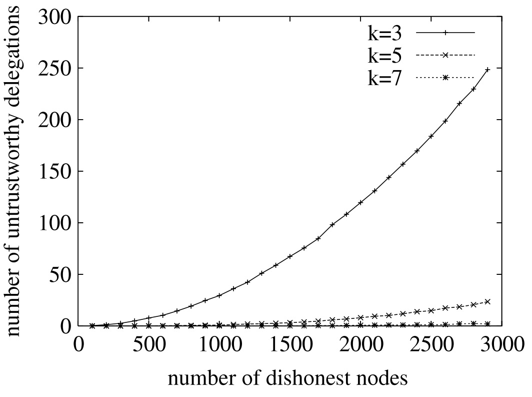

Fig. 3. Trustworthiness of delegation.

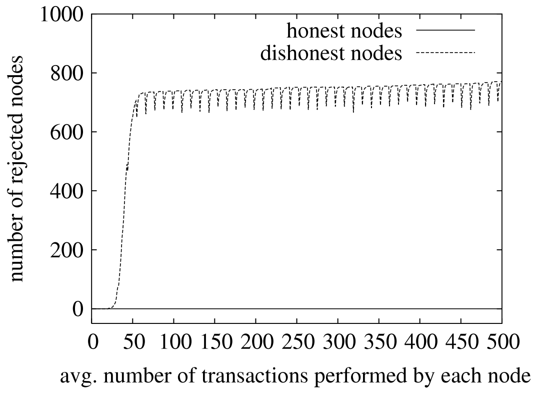

Fig. 5. Most of malicious nodes are rejected within the first 50 transactions.

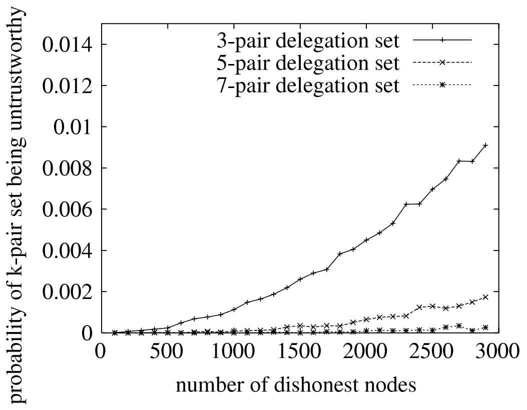

Fig. 4. Trustworthiness of *k*-pair delegation set.

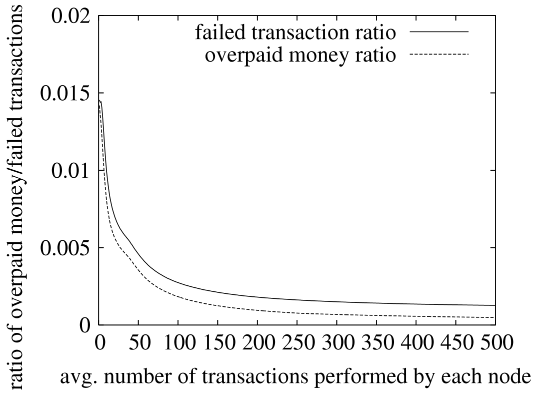

Fig. 6. The failed transaction ratio and the overpaid money ratio drop quickly to small percentages within the first 100 transactions.

If one participant in a transaction tries to deceive the other one, the transaction is called a *failed transaction*. We define "failed transaction ratio" as the number of failed transactions divided by the total number of transactions, and "overpaid money ratio" as the total amount of overpaid money divided by the total amount of money paid in the transactions. These metrics are used to assess the overall damage caused by dishonest nodes.

8.1. *Effectiveness of authority*

In the first set of simulations, we study the trustworthiness of the delegations and the *k*-pair delegation sets. Fig. 3 shows the number of untrustworthy delegations with respect to the number of dishonest nodes for $k = 3, 5,$ and 7. Recall that a delegation is untrustworthy if at least half of its members are dishonest. Out of 100,000 delegations, only a few number of them are untrustworthy. For $k = 5$, the number of nodes with untrustworthy delegations is just 23 even when there are 3000 dishonest nodes. Fig. 4 shows the probability for an arbitrary *k*-pair delegation set to be untrustworthy (Section 4.2). The 5-pair delegation set is trustworthy with a probability larger than 99.8% even when there are 3000 dishonest nodes. Note that when a delegation is untrustworthy, the dishonest members may not belong to the same colluding group. Without cooperation, the damage they can cause will be smaller.

8.2. *Effectiveness of MARCH*

The second set of simulations study the effectiveness of our incentive scheme. Fig. 5 presents how the number of rejected nodes changes with the average number of transactions performed per node, which can be used as the *logical time* as the simulation progresses. Recall that the default number of dishonest nodes is 1000. The figure shows that most dishonest nodes are rejected from the system within 50 transactions per node. Because of money refilling, some rejected nodes will recover after enough money is refilled, but they will be rejected again after performing malicious transactions. No honest nodes are rejected from the system during the simulation.

Fig. 6 shows that the failed transaction ratio drops quickly from 1.4% to 0.3% within the first 100 transactions per node, and the overpaid money ratio drops from 1.4% to 0.2% in the same period. As the time progresses, these ratios become even more insignificant. Note that the overpaid money ratio is smaller than the failed transaction ratio. This is because the dishonest providers have to lower their prices in order to compete with honest providers, which in turn lowers their ability to cause significant damage. Ironically, if a dishonest node with poor reputation wants to stay in the system, not only does it have to behave honestly to gain reputation, but also it has to do so with lower price in order to get consumers, which "repairs" the damage it does to the system previously.

Next, we study how the number of dishonest nodes affects the system performance. Fig. 7 shows the overpaid money ratio after 250 transactions per node. We find that the ratio increases linearly with the number of dishonest nodes. However, even when there are 3000 dishonest nodes, the overpaid money ratio remains very small, just 0.15%. Fig. 8 shows that the more the number of dishonest nodes, the more they are rejected.

Last, we study the impact of the threshold on the system performance. The threshold is used by a consumer to select the potential providers (Section 5.2). Fig. 9 shows that the overpaid

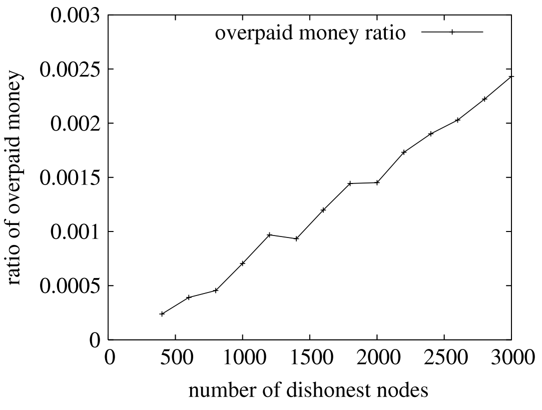

Fig. 7. The overpaid money ratio (measured after 250 transactions) increases linearly with the number of dishonest nodes.

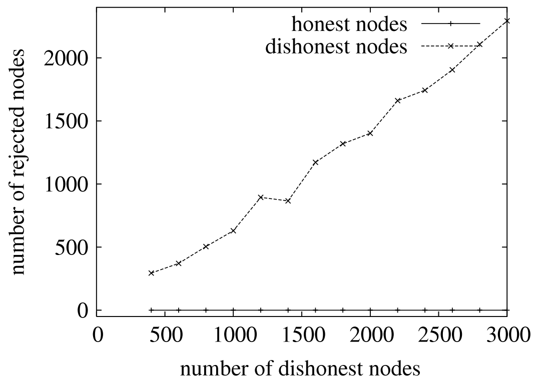

Fig. 8. The number of rejected dishonest nodes (measured after 250 transactions) increases linearly to the number of dishonest nodes.

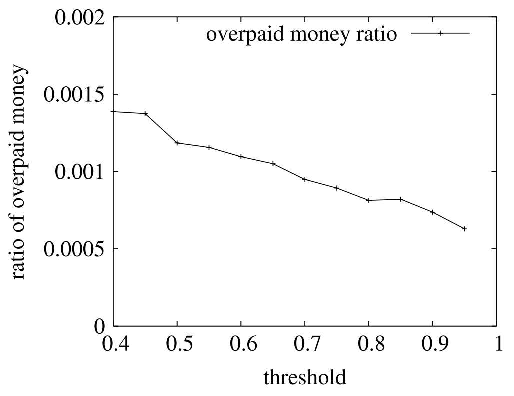

Fig. 9. The overpaid money ratio with respect to the threshold.

money ratio decreases linearly with the threshold value, which means the system performs better with a larger threshold. Fig. 10 shows that the number of rejected dishonest nodes is largely insensitive to the threshold value. However, when the threshold is too low, some honest nodes may be rejected by the system because a smaller threshold allows the dishonest nodes to do more damage on the honest nodes, which may even cause some honest nodes to be rejected from the system due to defamed reputation. The numbers in the above two figures are measured after 250 transactions per node.

9. Conclusion

We propose a distributed incentive scheme (called MARCH) for P2P networks. The scheme uses a distributed authority infrastructure with delegations, instead of a centralized server, to maintain the money/reputation information of the nodes. We use a five-phase transaction framework to incorporate both virtual money and reputation into our scheme, which solves a number of problems that the previous schemes have. We also present a key sharing protocol and a contract verification protocol to produce the contract proofs that are authorized by the delegations of the provider and the consumer of a transaction. We analyze the system properties and use simulations to evaluate the system performance. The results demonstrate that MARCH has the potential to solve the free-riders problem in today's P2P networks.

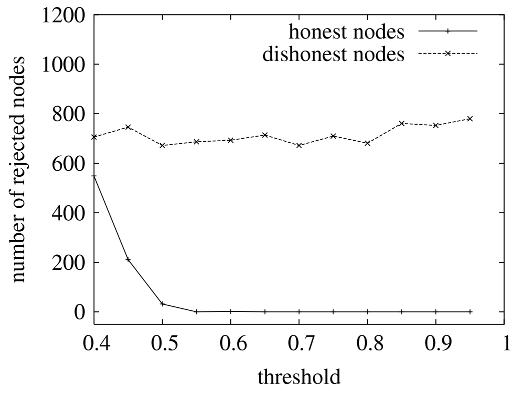

Fig. 10. The number of rejected nodes with respect to the threshold.

References

[1] T. Bocek, M. Shann, D. Hausheer, B. Stiller, Game theoretical analysis of incentives for large-scale, fully decentralized collaboration networks, in: Proc. of IEEE IPDPS, April 2008.

[2] L. Buttyn, Removing the financial incentive to cheat in micropayment schemes, IEE Electronics Letters 36 (2) (2002) 132–133.

[3] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, D. Wallach, Secure routing for structured peer-to-peer overlay networks, in: Proc. of OSDI'02, 2002.

[4] A. Cheng, E. Friedman, Sybilproof reputation mechanisms, in: Proc of the ACM SIGCOMM workshop on Economics of peer-to-peer systems, P2PECON '05, 2005.

[5] C. Dellarocas, Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior, in: Proc. of EC'00: the 2nd ACM conference on Electronic commerce, 2000.

[6] Y.G. Desmedt, Y. Frankel, Threshold cryptosystems, in: Proc. of CRYPTO'89, 1989, pp. 307–315.

[7] P. Dewan, P. Dasgupta, Securing reputation data in peer-to-peer networks, in: Proc. of Parallel and Distributed Computing and Systems, 2004.

[8] J.R. Douceur, The sybil attack, in: Proc. of IPTPS'01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, 2002, pp. 251–260.

[9] M. Feldman, K. Lai, I. Stoica, J. Chuang, Robust incentive techniques for peer-to-peer networks, in: ACM Electronic Commerce, 2004.

[10] M. Jakobsson, J. Hubaux, L. Buttyan, A micropayment scheme encouraging collaboration in multi-hop cellular networks, in: Proc. of Financial Crypto'03, 2003.

[11] R. Landa, D. Griffin, R. Clegg, E. Mykoniati, M. Rio, A sybilproof indirect reciprocity mechanism for peer-to-peer networks, in: Proc. of INFOCOM'09, 2009.

[12] S. Lee, R. Sherwood, B. Bhattacharjee, Cooperative peer groups in nice, in: Proc. of INFOCOM'03, Apr 2003.

[13] J.C.L.T.B. Ma, Sam C.M. Lee, D.K. Yau, A game theoretic approach to provide incentive and service differentiation in p2p networks, in: Proc. of ACM SIGMETRICS/PERFORMANCE, June 2004.

[14] S. Marti, H. Garcia-Molina, Identity crisis: anonymity vs. reputation in p2p systems, in: Third IEEE International Conference on Peer-to-Peer Computing, 2003.

[15] M. Meulpolder, J. Pouwelse, D. Epema, H. Sips, Limitations on the effectiveness of decentralized incentive mechanisms, in: Proc. of IEEE ICC, June 2011.

[16] R.S. Paul Resnick, Sybilproof transitive trust protocols, in: ACM Conference on Electronic Commerce, 2009.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content addressable network, in: Proc. of ACM SIGCOMM'01, August 2001.

[18] A. Rowstron, P. Druschel, Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: Proc. of Middleware'01, November 2001.

[19] M. Srivatsa, L. Xiong, L. Liu, Trustguard: countering vulnerabilities in reputation management for decentralized networks, in: Proc. of WWW'05, May 2005.
[20] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proc. of ACM SIGCOMM'01, August 2001, pp. 149–160.
[21] M. Venkatraman, B. Yu, M.P. Singh, Trust and reputation management in a small-world network, in: Proc. of ICMAS'00, 2000.
[22] V. Vishnumurthy, S. Chandrakumar, E.G. Sirer, Karma: a secure economic framework for p2p resource sharing, in: Proc. of the Workshop on the Economics of Peer-to-Peer Systems, June 2003.
[23] Y. Wang, J. Vassileva, Bayesian network trust model in peer-to-peer networks, in: Proc. of AP2PC'03, 2003.
[24] C. Wang, H. Wang, Y. Lin, S. Chen, A lightweight currency-based p2p vod incentive mechanism, in: Proc. of IEEE International Joint Conference on Computational Science and Optimization, CSO, 2010.
[25] Z. Zhang, S. Chen, Y. Ling, R. Chow, Capacity-aware multicast algorithms on heterogeneous overlay networks, IEEE Transactions on Parallel and Distributed Systems 17 (2) (2006) 135–147. Special issue on algorithm design and scheduling techniques (realistic platform models).
[26] Y. Zhang, W. Lou, Y. Fang, Sip: a secure incentive protocol against selfishness in mobile ad hoc networks, in: Proc. of WCNC'04, March 2004.
[27] B.Q. Zhao, J.C.S. Lui, D.-M. Chiu, A mathematical framework for analyzing adaptive incentive protocols in p2p networks, IEEE/ACM Transactions on Networking 20 (2) (2012) 367–380.
[28] H. Zhao, X. Yang, X. Li, Wim: A wage-based incentive mechanism for reinforcing truthful feedbacks in reputation systems, in: Proc. of IEEE Globecom, December 2010.

Zhan Zhang is software engineer in Juniper Networks. He received his M.S. degree in computer science from Fudan University of China in 2003, and Ph.D degree in Computer & Information Science & Engineering from University of Florida in 2007. His research interests include overlay networks, wireless sensor networks and network security.

Shigang Chen is an associate professor in the Department of Computer and Information Science and Engineering at University of Florida. He received his B.S. degree in computer science from University of Science and Technology of China in 1993. He received M.S. and Ph.D. degrees in computer science from University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for three years before joining University of Florida in 2002. His research interests include network security and wireless networks. He received the IEEE Communications Society Best Tutorial Paper Award in 1999 and NSF CAREER Award in 2007. He was a guest editor for ACM/Baltzer Journal of Wireless Networks (WINET) and IEEE Transactions on Vehicle Technologies. He served as a TPC co-chair for the Computer and Network Security Symposium of IEEE IWCCC 2006, a vice TPC chair for IEEE MASS 2005, a vice general chair for QShine 2005, a TPC co-chair for QShine 2004, and a TPC member for many conferences including IEEE ICNP, IEEE INFOCOM, IEEE ICC, IEEE Globecom, etc.

Zhen Mo is a current Ph.D. student in the Department of Computer and Information Science Engineering at University of Florida. He received his B.E degree in Information Security Engineering from Shanghai Jiao Tong University in 2007. Then he received his M.E degree in Theory and New Technology of Electrical Engineering from Shanghai Jiao Tong University in 2010. His research interests include network security and cloud computing security.

MyungKeun Yoon is an assistant professor in the Department of Computer Engineering at Kookmin University, Korea. He received the B.S. and M.S. degrees in Computer Science from Yonsei University, Korea, in 1996 and 1998, respectively. He received the Ph.D. degree in computer engineering from the University of Florida in 2008. He worked for the Korea Financial Telecommunications and Clearings Institute from 1998 to 2010. His research interests include computer & network security, network algorithm, and mobile networks.