Zhang Q, Wei W, Yu T. On the modeling of honest players in reputation systems. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 24(5): 808–819 Sept. 2009

On the Modeling of Honest Players in Reputation Systems

Qing Zhang[1] (张 晴), Wei Wei[2] (卫 伟), and Ting Yu[2] (于 挺), *Member*, *ACM*

[1] *Teradata, EL Segundo, CA 90503, U.S.A.*

[2] *Department of Computer Science, North Carolina State University, Raleigh, NC 27695, U.S.A.*

E-mail: qzhangqing@gmail.com; {wwei5, tyu}@ncsu.edu

Revised May 26, 2009.

Abstract Reputation mechanisms are a key technique to trust assessment in large-scale decentralized systems. The effectiveness of reputation-based trust management fundamentally relies on the assumption that an entity's future behavior may be predicted based on its past behavior. Though many reputation-based trust schemes have been proposed, they can often be easily manipulated and exploited, since an attacker may adapt its behavior, and make the above assumption invalid. In other words, existing trust schemes are in general only effective when applied to honest players who usually act with certain consistency instead of adversaries who can behave arbitrarily. In this paper, we investigate the modeling of honest entities in decentralized systems. We build a statistical model for the transaction histories of honest players. This statistical model serves as a profiling tool to identify suspicious entities. It is combined with existing trust schemes to ensure that they are applied to entities whose transaction records are consistent with the statistical model. This approach limits the manipulation capability of adversaries, and thus can significantly improve the quality of reputation-based trust assessment.

Keywords reputation, trust, user behavior modeling, collusion-resilient behavior testing

1 Introduction

Recently, we have witnessed the emergence and popularity of large-scale decentralized systems which allow entities to interact and transact in an ad hoc manner. Examples of such decentralized systems include online-auction communities and P2P resource sharing systems. These systems, on the one hand, offer great advantages in terms of service variety, flexibility and availability. On the other hand, they impose significant security challenges. In particular, entities in large-scale decentralized systems are often strangers. They are from different domains and may not have any pre-existing knowledge of each other. This leaves malicious parties the opportunities to cheat during transactions without being identified and punished. Therefore, before a transaction is conducted, it is paramount for two parties to assess each other's trustworthiness and evaluate the potential risks involved during the transaction.

Inspired by social interactions between human beings, reputation mechanisms have been introduced as one of the major techniques for the establishment and management of trust in the above mentioned decentralized systems. In a reputation system, once a transaction is finished, involved parties issue feedbacks that evaluate each other's behavior during the transaction. Before a new transaction starts, we first assess a party's trustworthiness based on the feedbacks on its previous transactions. This process can be viewed as the application of a so-called *trust function*, which takes as input the feedbacks of a party's previous transactions, and outputs a trust value. This trust value indicates the party's trustworthiness. A trust function may be further augmented by considering other factors such as the cost of an ongoing transaction, and the trustworthiness of feedback issuers.

Reputation-based trust management is built on the assumption that an entity's future behavior can be predicted from its past behavior. If a party provides good services in its past transactions (and thus obtains good feedbacks), then it is believed that it is likely to do so in future transactions. In other words, we expect that an entity will behave consistently so that its reputation can be meaningfully used to assess its trustworthiness. Thus, the effectiveness of reputation mechanisms depends on how well this assumption holds in a decentralized system.

Many trust functions have been proposed in the literature[1−9]. But often they can be easily manipulated or exploited by adversaries[10]. We argue that the essential reason for this insufficiency resides in the fact that adversaries do not follow the above mentioned assumption. They do not act consistently. Instead, they may intentionally adapt their behavior to take

Regular Paper
This work is partially supported by the National Science Foundation of USA under Grant Nos. IIS-0430166 and CNS-0747247.

advantage of a system's trust function. Existing trust functions are suitable to assess the trustworthiness of entities who behave consistently, but are vulnerable to adversaries who may change their behavior arbitrarily.

In this paper, instead of designing yet another trust function and applying it indistinguishably to all parties in a system, we advocate a two-phase approach to trust assessment. This approach integrates the modeling of "honest players" with trust functions. By "honest players", we mean those entities who behave consistently in a system. In the first phase of our approach, we examine the transaction history of an entity. Only when it follows the model of honest players, will we apply trust functions to further determine its trustworthiness in the second phase. Those who do not pass the first phase may be either discarded as untrustworthy (as they appear to manipulate the reputation system) or prompted to users for further examination.

In detail, we make the following contributions:

• We propose a statistical model of the behavior of honest players. Specifically, we consider the number of good transactions (those offering satisfactory services and receiving positive feedbacks accordingly) of an honest player as a random variable x. We show that if an entity's behavior is consistent and not affected by other factors, then x follows a binomial distribution $B(n, p)$, where n is the number of transactions a party conducted during a period of time, and p is the percentage of good transactions among these n transactions.

• We identify two typical attacks which exploits existing trust functions. Attacker manipulates the trust system by building up its reputation through a number of transactions before launching attacks. Given the transaction history of a party, we design an algorithm to determine with high confidence whether it follows the behavior model of honest players.

• Attackers often rely on collusion and issue false feedback information to boost their reputations. We further extend the above statistical model and make it resilient to collusion.

• We show through experiments the quality improvement of trust assessment offered by the proposed two-phase approach. Intuitively, because we compare a party's transaction history with the behavior model of honest players, an adversary cannot dramatically change its behavior without being detected. Instead, a successful attack needs to ensure in the first place that the resulting transaction pattern is consistent with the honest player model. This will significantly increase the cost of attacks.

We organize the rest of the paper as follows. In Section 2, we present an abstract model of reputation systems, and introduce some of the basic concepts and notations used in the later sections. In Section 3, we discuss the intrinsic properties of honest players and formally model their behavior. We then present algorithms to determine whether an entity is an honest player by comparing its transaction history with the proposed statistical model. Section 4 discusses a behavior model that is resistant to collusion. Their effectiveness is shown experimentally in Section 5. We report related work in Section 6, and conclude this paper in Section 7.

2 Reputation Systems

We assume that entities in a decentralized system interact with each other through transactions. Transactions are not limited to monetary interactions. They also include activities such as retrieving information, and downloading files. We further assume a transaction is uni-directional, i.e., given a transaction, there is a clear distinction between a service provider (the server) and a service consumer (the client). For simplicity, in this paper we only consider trust assessment of service providers.

A feedback is a statement issued by the client about the quality of a server in a single transaction. In general, a feedback may be multi-dimensional, reflecting the client's evaluation on a variety of aspects of a service, e.g., price, product quality and time of delivery. For simplicity, we assume in this paper that a feedback is one-dimensional and taken from the domain {*positive*, *negative*}. We say a transaction is good if it gets a positive feedback. Otherwise, it is a bad transaction. For the purpose of our discussion, we assume all the transaction feedbacks are available for trust assessment (e.g., through a central server as in online auction communities, or through special data organization schemes in P2P systems[11]). In practice, our scheme can be equally applied to systems where only portions of feedbacks can be retrieved. We denote a feedback as a tuple (t, s, c, r), where t is the time that a transaction happens, s the server, c the client, and r the rating from the client.

Let $\mathcal{F}$ denote the set of all possible feedbacks, and V be the set of all entities in a system. A trust function is thus a mapping $2^{\mathcal{F}} \times V \rightarrow T$, where $T = [0, 1]$ is the domain for trust values. A server's trust value is essentially a prediction of its future behavior. Without loss of generality, we can interpret a server's trust value as the predicted probability that the next transaction with the server will be satisfactory. Each client defines its own trust threshold. Only when a server's trust value is above the threshold, will the client proceed to a transaction with the server.

3 Modeling of Honest Players

When we use reputation to choose good service providers, we implicitly assume that we can safely infer a party's future behavior from its trustworthiness in the past transactions. This assumption is in general true for honest players. By honest players, we mean those service providers who try their best to provide good services to others. This intention does not change when dealing with different clients or conducting transactions at different times.

Note that a honest server does not mean that it can be fully trusted. Instead, it may still get negative feedbacks from time to time, due to factors that cannot be controlled by itself. For example, a party Alice in an online auction site may have delayed deliveries from time to time. But this is not because Alice wants to do so intentionally. Instead, it is caused by the poor services of Alice's local postal office, whose service quality is out of the control of Alice. As another example, users of an online music store may occasionally experience difficulties during downloading, due to the store's shaky file servers.

Because of these uncontrollable factors, the outcome of an honest player's transactions can be viewed as a random variable that follows a certain distribution D. This view justifies the rationale of reputation mechanisms. A server's past transactions are essentially a sample of D. And the goal of a reputation system is to derive a server's distribution through the sample. This is also consistent with our interpretation of trust values that a trust function outputs.

Many trust functions have been proposed in the literature. They have been proven to be effective when assessing the trustworthiness of honest players. On the other hand, it is also common that, given a particular trust function, dedicated attacks can be designed to manipulate and exploit that trust function. There are also some generic attacks against trust functions, which we briefly discuss as follows.

Hibernating Attack. An attacker first carries out some good transactions to build his reputation up to a trust value T_1. We call T_1 the *cover reputation* of this attacker. When his trust value meets the trust threshold of some specific users he want to cheat, he can then consecutively launch attacks towards his target users without being detected.

Periodic Attacks. Every time the attacker successfully achieved a cover reputation T_1, he will launch attacks until his trust value drops to T_2. Then he will provide some good services again to re-build his reputation. It can continue doing so without being detected.

From these attacks, we see that since an attacker can adjust its behavior arbitrarily and intentionally, its past transaction feedbacks do not serve as an effective means to predicting the quality of its future transactions.

The above observation suggests that it would not be sufficient to apply trust functions *indistinguishably* to servers. Instead, since trust functions are designed to predict a server's future behavior, it is important to first study the behavior pattern of a server and determine whether it is indeed an honest player. Therefore, we propose a two-phase approach to trust assessment, which combines trust functions with honest player screening. Specifically, in the first phase, we check whether the transaction history of a server follows the typical behavior pattern of honest players. Only when the first phase is passed, will we apply existing trust functions to determine whether the server is a good service provider. Fig.1 shows the general framework for the proposed two-phase approach.

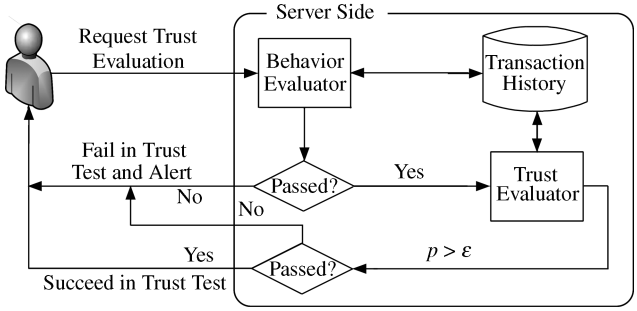


Fig.1. General framework of the proposed two-phase approach.

3.1 Statistical Model of Honest Players

In this subsection we propose a statistical model to capture the behavior pattern of honest players. Recall that the trustworthiness of a server is to approximate the probability that we obtain a satisfactory service from the server. Further, this probability is caused by factors that cannot be controlled by the server. For simplicity, we assume this probability is static, i.e., it does not change from transactions to transactions. Our techniques can be easily extended to handle dynamic cases.

Given a sequence of independent transactions, $trans_1, \ldots, trans_n$, conducted by a server, let X_i denote the event that $trans_i$ is a good transaction for $i = 1, \ldots, n$. Then $X_1, \ldots, X_n$ is a sequence of independent identical distributed (iid) events. The probability of each event to happen is given by

$$P(X_i) = \begin{cases} p, & \text{if } X_i = 1, \\ 1 - p, & \text{if } X_i = 0, \end{cases}$$

where p is the trustworthiness of the server.

Clearly, given a sequence of n transactions, the number of good transactions among them will follow

a binomial distribution $B(n, p)$. Thus, it seems we only need to perform a binomial test to determine whether a server is honest. One subtlety here is that in practice the parameter p is not known. Further, the order of transactions does matter for our purpose. For example, suppose both Alice and Bob have finished 100 transactions, among which 90 receive positive feedbacks. While the bad transactions of Alice spread among the 100 transactions, those of Bob are in fact the last 10 transactions he conducted. If we treat these 100 transactions from Alice and Bob respectively as one sample and perform a binomial test, they will have the same test result, though clearly Bob seems more suspicious in this example. This problem also shares similarity to pseudo random sequence testing[12]. But again, we do not know the probability p which is required by existing pseudo random sequence testing algorithms.

Before we present our approach to server behavior testing, we would like to have a brief discussion of possible extensions to the above statistical models.

First, as stated above, for simplicity, we assume that a honest player's behavior is consistent and static, i.e., it follows the same behavior model from transaction to transaction. In practice, the factors that affect one's service quality may vary depending on the specific application environments. For example, the network condition for file sharing systems providing file service can be workload dependent and may vary during different time periods. We can adapt our model based on such application-specific knowledge.

Second, in many applications feedback ratings are not binary, or are multi-dimensional. Our model can be easily extended to handle such feedbacks. Specifically, we only need to replace binomial distributions in our framework with multinomial distributions for multi-value feedbacks, or build a statistical model for each dimension for multi-dimensional feedbacks. The statistical model can also be temporal. We may have different models for weekdays and weekends, or for the time 9am to 5pm and for other time intervals. Again, these are all application-dependent and require domain-specific knowledge, which we will not elaborate further. To show the mechanism of our framework concisely, we will assume the binomial distribution model for honest players in the rest of this paper.

Note that in reputation systems it is also possible for attackers to launch cheat-and-run attacks. That is, an attacker conducts one bad transaction after several honest transactions, or even upon joining the system, then leaves the system and never returns. It is difficult to prevent such attacks by purely relying on reputation mechanisms. Typically approaches are to increase the cost of joining a system in the first place (e.g., requiring certified IDs or membership fees) so that short affiliations with a system are not cost-effective. We assume such techniques have been deployed in a reputation system, and thus focus on the attack such as periodic attacks and hibernating attacks which requires long affiliations with a system.

3.2 Server Behavior Testing

Given a sequence Q of n transactions conducted by a server S, we break Q into $k = \lfloor n/m \rfloor$ consecutive blocks, each with m transactions. We call each block a transaction window. Let G_i denote the number of good transactions in transaction window W_i, $i = 1, \ldots, k$. If the server is an honest player with trust value p, we will have the following lemma.

Lemma 3.1. *Given an honest server with trust value p, for any preselected small values ϵ and δ, there always exists an N such that if the total number of transactions $n > N$, then the probability*

$$P\Big(\frac{\sum G_i}{n} - p \geqslant \epsilon\Big) < \delta.$$

Proof. The number of good transactions in the whole sequence is $\sum G_i$. And the quality of a transaction conducted by an honest player with trust value p is actually the outcome of an Bernoulli experiment with success probability p. Thus the inequality holds naturally following Bernoulli's law of large numbers. □

Lemma 3.1 suggests that for an honest player with trust value p, when the transaction history is big enough, we can use $\hat{p} = \frac{\sum G_i}{n}$ to approximate p. And $\{|G_1|, \ldots, |G_k|\}$ forms a set of samples following the distribution $B(m, \hat{p})$. Thus, to test whether a server is honest, we can simply check whether the number of good transactions in each window follows $B(m, \hat{p})$. There are many ways to check if a set of samples follows a specific distribution. In this paper, we will use L^1 norm of the distribution distance d between the actual distribution of G_i and the binomial distribution $B(m, \hat{p})$. If d is less than a predetermined threshold ε, then we can conclude that the server is honest.

The confidence of our conclusion depends on the selection of distribution distance threshold ε. One way to achieve high confidence is to derive the distribution of L^1 norm distance and select ε that corresponds to 95% confidence interval. More specifically, given a set of samples randomly generated following $B(n, \hat{p})$, we define a random variable *dist* that represents the L^1 norm distance between the set of samples and $B(n, \hat{p})$. Once we derive the distribution of *dist*, we can select ε to be the threshold that gives 95% confidence interval.

Though theoretically simple, it is rather complex to derive the distribution of *dist*. In this paper, we take

an empirical approach instead. We randomly generate a reasonably large number of sets, each of which contains m transactions whose feedbacks are generated following $B(m, \hat{p})$. We then measure the L^1 norm distances of these sets to $B(m, \hat{p})$. ε is selected such that 95% of the distances of the generated sample sets are smaller than ε.

Fig.2 shows the pseudocode of the behavior testing algorithm.

```
Retrieve the transaction history H of a service provider;
Break H sequentially into k transaction windows, each of which has m transactions;
for i = 1 : k do
    Count the number of good transactions G_i in each window;
end for
Use L^1 norm distribution distance to check whether the distribution of all the G_i follows the binomial distribution B(m, (sum G_i)/|H|);
if L^1 norm distribution distance is bigger than ε then
    Alert ("Destination peer is suspicious");
    Abort;
else
    Call a user specified trust function
    Return the computed trust value to user
end if
```

Fig.2. Two-phase behavior testing algorithm.

3.3 Multi-Testing of Server Behavior

If a server has a long transaction history, then a small number of additional transactions will not significantly change the statistics of the transaction history. An adversary thus may still take advantage of this property to launch hibernating attacks. Note that no matter what trust assessment schemes are used, such attacks cannot be prevented. For example, as an extreme case, an attacker can deliver good services all the time, and then suddenly starts cheating on clients. The first bad transaction in this attack can never be prevented. Thus, the goal of a trust management system is instead to limit the number of bad transactions that may evade trust assessment in a short period of time.

Suppose an attacker suddenly starts conducting bad transactions. The significance of these transactions depends on the length of its transaction history. Thus, if we reduce the number of transactions considered when testing a server's behavior, such abnormal increases in bad transactions will be revealed. On the other hand, if we only consider the most recent l transactions, it will open doors to periodic attacks, since bad transactions are totally discarded once they are outside of the most recent l transactions.

The above observation suggests that we need to consider both the long-term and the short-term behaviors of a server so that we can have a balanced assessment of its trust. Thus, we propose a multi-testing approach to checking a server's compliance with the behavior of honest players. Suppose a server has conducted a total of l transactions. We first check whether the behavior of the server is honest when considering all l transactions. This corresponds to a long term behavior checking. We then perform the same test by only considering the most recent $l-k$ transactions, where k is a constant that controls the progress of the testing. We continue doing so until the number of considered transactions is too small to be statistically significant. For an honest player, its behavior during any subsequence of the transaction history should follow binomial distributions. Therefore, the failure of any test would indicate a potentially suspicious server.

4 Collusion-Resilient Behavior Testing

In our discussion so far, it is assumed that the attacker tries to achieve his goal alone without other's help. Therefore, we do not consider the authenticity of a feedback. In particular, we assume that a server can only earn positive feedback through providing good services. However, in reputation systems, it is not unusual for a group of attackers to collude and boost each other's reputation through fake feedbacks. Without checking the authenticity of feedbacks, a group of attackers can easily evade behavior testing. For example, whenever an attacker needs to provide a good service in order to be conformed with the honest player model, he can simply ask a colluder to submit a fake positive feedback. Note that even in systems with means to prevent arbitrary feedbacks (e.g., a seller has to pay a fee in each transaction in eBay), the cost to obtain a good feedback from a colluder is still much cheaper than actually providing a good service (e.g., no actually shipping of goods is needed for transactions between colluders).

The above observation suggests that we have to consider the pattern of feedback issuers as well when modeling honest player behavior. One seemingly attractive approach is to identify as colluders those clients who issue many positive feedbacks to a server. However, this may easily bring false positives. For example, www.shop.com may be Alice's favorite online shopping website. She often shops there and keeps having good experiences. The fact that Alice issues many positive feedback does not mean she colludes with www.shop.com and boost its reputation intentionally and falsely.

In this paper, instead of trying to identify specific

colluders, we propose using the overall client patterns to model an honest player. Intuitively, if an honest player consistently provides good services, its reputation will gradually build up and attract more clients. Therefore, the set of clients who leave good feedbacks will expand as time goes by. We call this set of clients the server's *supporter base*. On the other hand, if an attacker relies on its colluders to maintain the good reputation while cheating on other clients, its supporter base would be relatively stable after its reputation builds up. Another key observation is that, for an honest server, its services will always be of high quality no matter the service consumer is a frequent client or an occasional one. In other words, given any subset of all its clients, their feedbacks are likely to have very similar distributions. For an attacker, the distribution of feedbacks from colluders tends to be different from that from other clients.

Based on the above analysis, we propose the following technique for collusion-resilient behavior testing. Recall that a feedback is a tuple (t, s, c, r), where t is the time that a transaction happens, s the server, c the client, and r the rating issued by c for this transaction. Given a sequence Q of feedbacks for a server s, we break them into groups by feedback issuers, and then re-order the sequence such that groups with more feedbacks appear before those groups with fewer ones. Feedbacks inside each group are ordered according to the time of transactions.

Given this new feedback sequence Q', we conduct the distribution-based behavior testing as described in Subsection 3.2. The intuition behind this technique is that for an honest player the distribution of feedbacks from frequent clients should be similar to that of occasional clients. In order to pass this behavior testing, an attacker has to increase its supporter base. In other words, he was forced to provide good services to other clients beyond his colluders.

Similarly, to deal with attackers who build reputations through long histories, we can also perform multi-testing of server behavior. Specially, suppose Q has l transactions. After behavior testing over the whole sequence, we choose the latest $l-k$ transactions and perform a behavior testing as described above.

As noted early, in practice a server may not always provide uniform services to all the users, even if they are honest. For example, an online movie server in the US may provide good services to customers in North America, but not to those in Africa, due to network capabilities. This does not mean that any US customers who leaves positive feedback is in collusion with the server. To include such cases into our framework, we may extend our scheme and apply statistical modeling and testing to transactions in different categories. In the above example, we may group transactions into two categories: North America and Africa. And our testing can be performed on each category, or only on those in which we are interested. For example, if a user is in North Carolina, knowing the server's service quality to customers in North America would suffice. On the other hand, if some factors that affect service qualities are unknown when we design behavior testing schemes, then false alerts may be raised, which will help us identify such factors. Thus our scheme not only gives an extra level of protection, but also makes it possible to adaptively discover important factors about a system.

5 Experiments

As mentioned above, a reputation system cannot guarantee that a client never receives bad services from an adversary. The purpose of trust management is to restrict the ability of an attacker, making it hard to launch attacks without being detected. Suppose an attacker wants to conduct M malicious transactions. If these transactions can be conducted arbitrarily, especially, continuously, and meanwhile its trust value is always maintained above clients' trust threshold, then we consider that the attacker succeeds in these attacks. On the other hand, if a scheme forces an attacker to conduct a lot of good transactions to cover those bad ones, such that its transaction history is statistically indistinguishable from the behavior of honest players, then we say that the scheme is resilient against attacks. The number of good transactions an attacker needed in this case can be treated as the cost for the attacker to finish M attacks. As a practical measurement, we will use the total number of good transactions needed to launch M attacks as the metrics to measure the strength of a scheme. The bigger M is, the more cost the attacker will have to pay, which also means that he needs to adjust his behavior more to be like an honest user.

5.1 Server Behavior Evaluation

We design a set of experiments to show the ability of the proposed two-phase approach, and compare it with traditional approaches which apply a single trust function directly. In the experiment, we will first assume the attacker has pre-established a high reputation in the system, by behaving as an honest player. Let H be the transaction history of the attacker before he launches attacks. We call these transactions *the preparation phase* of the attack.

The first trust function we compare with is to simply compute the trust value as the ratio of the number of good transactions over the total number of transactions. We call this trust function the average trust function. Many existing studies are based on trust functions in this form, and further augment it by considering other

factors, such as the credibility of the source of feedbacks, and transaction context[3,7]. As argued in [13], the average function is often the most cost-effective in complex systems.

The average trust function takes all transaction history for trust computation. Thus it only considers the long-term behavior of a server. In order to capture the dynamic behavior of servers more quickly, many techniques have been proposed to discount old transactions when deriving one's reputation[14−16]. The second type of trust function we compare with is the one proposed in [15]. Let f_t be the feedback of the last transaction of a server, and R_{t-1} be the trust value before considering the last transaction. Then in their approach, the latest trust value will be computed as $R_t = \lambda f_t + (1-\lambda)R_{t-1}$. We call this function the weighted trust function.

In our experiments, we suppose that an attacker has prepared H transactions as an honest user with trustworthiness of 95%. We investigate how many more good transactions are needed if the attacker wants to launch 20 successful attacks. We call those transactions after the preparation phase *the attack phase*. We first apply the average trust function and the weighted function directly as done in previous works. We then combine them with the two behavior testing schemes, and see how many more transactions they will force attackers to conduct in order to achieve their goal. The trust threshold of clients is set to 0.9.

We assume that attackers are strategic and aware of the trust functions as well as the behavior testing algorithms. Specifically, it adopts the following procedure to determine whether to provide good or poor services in the next transaction. It first assumes that it will conduct a bad transaction next, and considers the resulting transaction history H'. If H' is consistent with the behavior model of honest players, and the trust value computed from H' is no less than 0.9, then the attacker will cheat in the next transaction. Otherwise, it will provide good services.

The experimental result with the average trust function is shown in Fig.3. The x axis represents the number of transactions that an attacker has conducted during the preparation phase, and the y axis represents the number of good transactions the attacker needs to conduct before it finishes 20 bad transactions in the attack phase. "Average" denotes that the system only uses the average trust function. "Scheme1 + Average" and "Scheme2 + Average" denote the situations when we combine the single-behavior testing and the multi-behavior testing with the average trust function, respectively.

We see that, if we only apply the average trust function, as long as the ratio of the number of good transactions over the total number of transactions exceeds the trust threshold 0.9, the attacker can always keep conducting bad transactions, until its trust value hits 0.9. Then it has to conduct some good transactions. In general, after every 9 good transactions, the attacker can launch an attack. When the number of transactions in the initial transaction history is over 400, the attacker can always launch 20 attacks consecutively and still satisfy the trust threshold (90%). This is a typical hibernating attack.

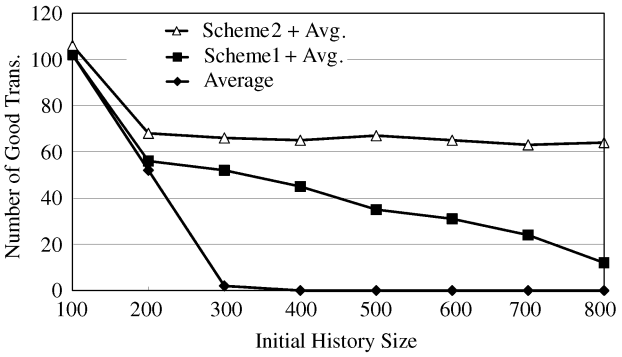


Fig.3. Cost of attackers when varying initial histories: average function.

We then combine our behavior testing algorithms with the average trust function. We choose transaction window to be of size 10, and the binomial distribution $B(10, \hat{p})$ is the expected behavior, where $\hat{p}$ is dynamically computed from the transaction history. From the figure we can see that when the history contains 100 transactions, all schemes impose the same cost to attackers, since the number of good transactions needed in this situation is mostly determined by the trust threshold. As the initial history size grows, the behavior testing schemes begin to take effect, which imposes higher cost than only using the average trust function. We also observe that when the size of the initial transaction history is relatively small, attacker needs to conduct more good transactions in order to pass the distribution test. But when the attacker prepares a longer transaction history, it will need fewer good transactions to pass the single behavior test. This is because single behavior testing only considers the distribution over the whole transaction history. When the size of the transaction history is large, there will be more transaction windows. Those windows containing bad transactions only occupy a smaller portion of the overall transaction history. Therefore it will not affect the distribution much. Thus, when an attacker has a long preparation phase, single behavior testing is prone to hibernating attacks.

Next we look at "Scheme2 + Average". The results

of "Scheme2 + Average" always outperform those of "Scheme1 + Average". In particular, even as the attacker includes more transactions in the initial transaction history, the number of good transactions needed by the attacker to achieve its attacking goal remains constant. This is because multi-testing checks both the long term and the short term behaviors of a server. Increasing the size of the initial transaction history will not buy much benefits to attackers. In other words, multi-testing is more resilient against attacks.

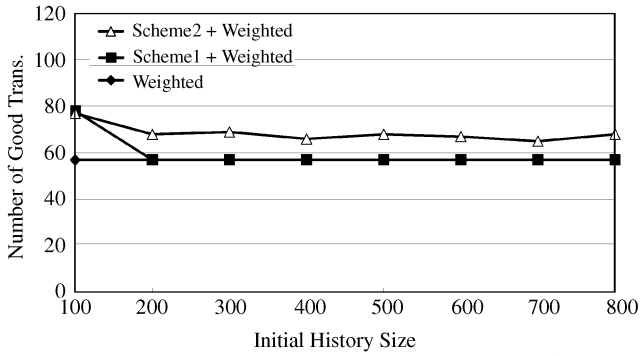


Fig.4. Cost of attackers when varying initial histories: weighted function.

Fig.4 shows the result when behavior testing algorithms are combined with the weighted trust function. In this experiment, we set $\lambda = 0.5$. We have observed similar trends as in Fig.3. With only the weighted trust function, the attacker essentially launches a periodic attack. Since the most recent transaction is assigned a much larger weight than that of others, the attacker can never conduct two consecutive bad transactions. Instead, after each bad transaction, the attacker needs to conduct 2∼3 good transactions to ensure its trust value to be over 0.9.

For approaches combining behavior testing with trust functions, when the initial history is small, periodic attacks introduce significant statistical difference. So both "Scheme1 + Weighted" and "Scheme2 + Weighted" perform well. When the attacker introduces more transactions during the preparation phase, "Scheme1 + Weighted" fails to constraint the attacker, due to similar reasons as discussed above. Meanwhile, the performance of "Scheme2 + Weighted" is not affected by the size of the initial transaction history. It imposes significant cost for the attacker to fulfill its goal.

5.2 Evaluation of Collusion-Resilient Behavior Testing

We conduct experiments to evaluate the effectiveness of the proposed collusion-resilient behavior testing. Like in Subsection 5.1, the goal of the attacker is to conduct 20 bad transactions and maintain his reputation over 0.9. We assume among a total of 100 potential clients, 5 of which are colluders of the attacker. During the preparation phase, the attacker only interacts with his colluders, and builds up a reputation of 0.95. During the attack phase, for each transaction, the attacker has three choices: cheating on a client, providing a good service to a client, or getting help from a colluder (i.e., ask the colluder to give a good feedback). The attacker will strategically determine his next action, by consulting both the trust function used in a system and the above collusion-resilient behavior testing algorithm.

Different from the experiments in Subsection 5.1, here we need to consider the list of clients who come to the server for service, as feedback issuers are important to behavior testing. We adopt a probabilistic approach to generate the client list at each step. Specifically, if a client c has never gotten service from the server s before, then the probability for c to request service from s is $a_1 \cdot p$, where a_1 is a constant and p is the current reputation of s. Similarly, we have parameters a_2 (and a_3) for those clients who recently got a good (or a bad) service from s. In the experiment, we set $a_1 = 0.5$, $a_2 = 0.9$ and $a_3 = 0.2$ respectively.

Figs. 5 and 6 show the effect of single- and multi-behavior testing to combat collusion when the simple

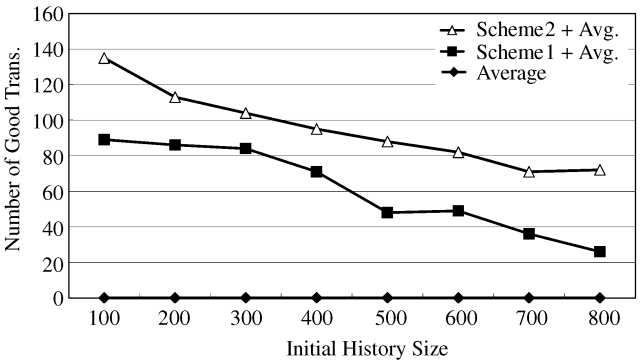


Fig.5. Cost of attackers with collusion: average function.

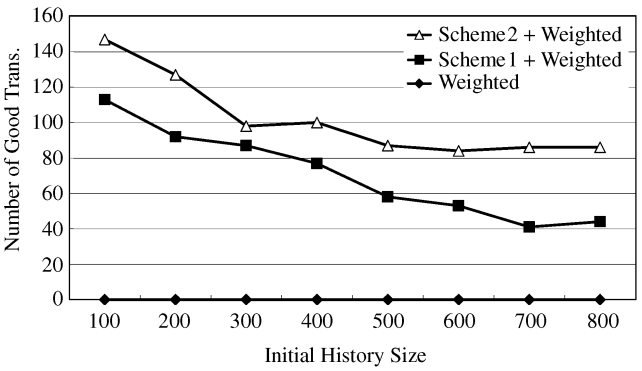


Fig.6. Cost of attackers with collusion: weighted function.

average trust function and the weighted trust function are used respectively. They show the similar observations as in Subsection 5.1. Note that the "y" axis shows the number of good transactions that the attacker has to provide to clients other than his colluders, which reflects the true cost for the attacker to achieve his goal. Compared with Figs. 3 and 4, we see that, when there is no behavior testing, the attacker can achieve his attacking goal without providing any good services to the clients, due to the help of colluders. On the other hand, our collusion-resilient behavior testing significantly constrains the behavior of an attacker, forcing him to behave more like an honest player instead of gaming the reputation system arbitrarily. These two figures yet again show the importance of multi-behavior testing. While a longer preparation phase helps the attacker lower his cost at the attack phase, multi-behavior testing imposes an almost constant cost on the attacker independent of the number of transactions in the preparation phase. This is particularly important with the existence of collusion, because with the help of colluders, the cost of the attacker in the preparation phase is extremely low.

5.3 Detection Rate

Our scheme is based on statistical differences of the transaction patterns. It may not achieve 100% detection rate. We design experiments to examine the detection rate of the proposed two-phase behavior testing approach. Suppose an attacker tries to keep his reputation value no less than 0.9 while launching periodic attacks according to a certain size of attack windows $N = 10, 20, \ldots, 80, \ldots$. That is, attackers will launch $N \times 0.1$ attacks within every N transactions. Fig.7 shows the detection rate with respect to the attack window size.

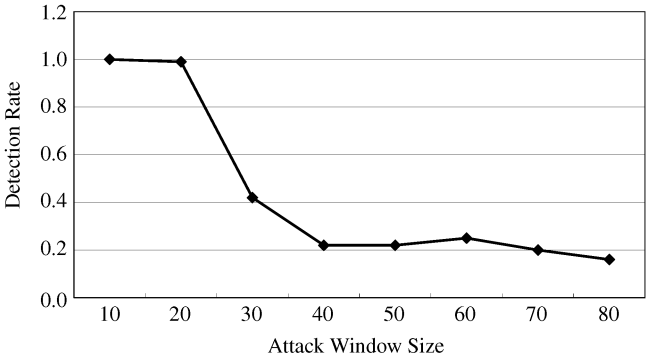


Fig.7. Detection rate vs. attack window size.

From the result, we observe that the detection rate decreases as the size of attack windows increases. When the size of attack window is 10, an attacker's behavior is quite different from random binomial behavior. Hence, the proposed algorithm detects this attack pattern. As the size of attack window increases, the attack behavior looks closer and closer to the behavior of a normal service provider, and the detection rate also drops. This result implies that, the closer the behavior of service provider is to the honest player behavior model, the less likely the service provider will be identified as a malicious one. This is one desirable property: if the attacker is forced to behave similarly to an honest player, then it can be regarded as an honest player.

5.4 Distribution Distance

As described in Subsection 3.2, the distribution distance determines the confidence of behavior testing. Thus we also design experiments to show how the distribution distance changes with the increase of initial history size. We study the distribution distance under different initial history sizes with 95% confidence, as explained in Subsection 3.2. Fig.8 shows that the distribution distance converges very quickly as the initial history size increases.

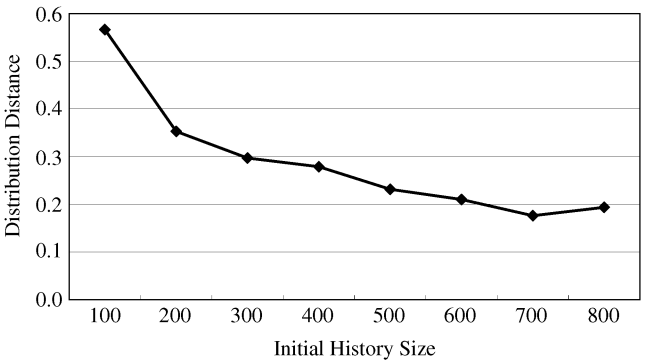


Fig.8. Distribution distance vs. initial history size.

5.5 Performance

Considering the large set of transaction information available in open systems, we expect that the proposed scheme should not degrade the performance of the trust system severely. Next we evaluate the performance overhead for single- and multiple-behavior testing. Let n denote the size of the initial history. Obviously, the time complexity of single-behavior testing is $O(n)$, since we only need to check the whole transaction history once. For multi-behavior testing, the basic scheme proposed in Subsection 3.3 would have a complexity of $O(n^2)$. The analysis is straightforward: the behavior testing will run $\frac{n}{k}$ times, where k is defined in Subsection 3.3. And each run of this multi-behavior testing will process $n-k, n-2k, \ldots$ transactions.

For large data sets, $O(n^2)$ is not desirable. In our

implementation, we optimize the basic multi-behavior testing algorithm by reusing some intermediate statistics. Specifically, we first compute statistics of the most recent $n - sk$ window, where $s = \lfloor(\frac{l}{k})\rfloor$. Then when we compute the $n - (s-1)k$ window, we can reuse the results for the $l - sk$ window, and so on for $n-(s-2)k, n-(s-3)k, \ldots$. The complexity is reduced to $O(n)$ with this optimization.

Fig.9 shows the running time of each of our scheme. The experiments are executed on a Dell desktop with 2.66 GHz CPU and 4 GB memory. For showing the running time more clearly, the sizes of the initial history are set to be from 100 000 to 800 000. Even in this case, behavior testing can be finished very quickly. For normal transaction histories with less than 10 000 transactions, the time for behavior testing is negligible.

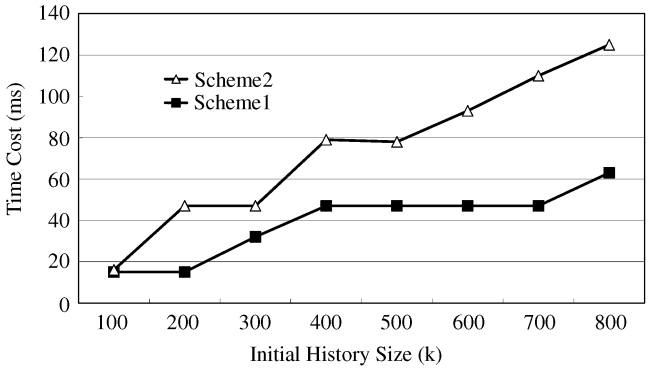


Fig.9. Time cost vs. initial history size.

6 Related Work

Reputation-based trust management has raised a lot of interests among researchers in recent years, and many trust functions have been proposed. Previous work mainly focuses on the design of trust functions based on various feedback collection mechanisms. For example, in [7], Xiong and Liu proposed the PeerTrust model, in which trust is evaluated as the average of satisfied transactions over all the transactions data, weighted by different transaction contexts. In [3], the number of satisfied transactions between each pair is collected to infer a ranking over all peers in the system. There is also some work that deals with non-binary values. In [9], for example, Yu *et al.* proposed a trust combination algorithm for feedbacks that can take values from {positive, neutral, negative}. Some recent work such as [17] studies how to make the reputation aggregation in decentralized systems in an efficient way. These trust functions do not take the time of feedbacks into consideration.

Some other work does consider entities' dynamic behavior in the design of trust functions. Due to the observation that once an attacker establishes its high reputation in a system, it can abuse it to conduct malicious activities, decay factors are commonly used so that most recent feedbacks have a higher weight than those issued long time ago. By doing so, one's reputation reflects more of its most recent behavior. Examples include [14, 16, 18–19]. The general idea is to assign time-based weights w_i to each feedback f_i, such that $\sum w_i = 1$.

In recent years, researchers begin to study how to integrate richer information into reputation computation. For applications such as Wikipedia, the amount of a user's editing being preserved reflects his trustworthiness. Adler *et al.* proposed a new reputation scheme by looking at the content of the service provider's contribution directly[20] instead of taking user's referral scores as inputs. Inspired by h-index, Zhao *et al.* proposed H-Trust which combines the evidence score and the number of referrals providing that score in trust computation[21]. In [22], cost of finding information source was taken into consideration in the selection process of referral. The multi-dimensional reputation computation is discussed in [23]. In [24], Fullam *et al.* studied the case when both local information and referrals are available, and the algorithms of how to combine these information are carefully designed. Despite all the efforts to identify the factors that may affect the trust rating, [13] analyzed many complex trust computation mechanisms and claimed that when the target system has a lot of dynamics and malicious peers, employing a complex mechanism may be not worth the cost comparing a simple trust mechanism such as the average function.

Hypothesis testing is an important technique in statistics. It checks whether a set of samples follows an expected distribution[25]. Most hypothesis testing techniques assume the parameters of the expected distribution are known, which is different from the problem in this paper. Hypothesis testing for distributions with unknown parameters mainly focuses on normal distributions. It is possible to approximate a binomial distribution with a normal distribution. However, the accuracy by testing the approximate normal distribution is questionable in the context of this problem.

7 Conclusion

A single trust function is often not sufficient to provide high quality trust assessment in reputation systems. An adversary is often able to adapt its behavior arbitrarily and manipulate a specific trust function. In this paper, instead of guessing what an adversary can do and trying to prevent specific attacks, we propose a statistical model to capture the behavior of

honest players. We develop algorithms to test whether the transaction history of a server is compatible with the behavior model of honest players. We propose a two-phase approach to trust assessment that combines behavior testing with trust functions. This approach forces an adversary to behave consistently with the model of honest players, and thus significantly limits its manipulation capability.

Our approach is applicable when a server has a reasonably long transaction history. When applying to service providers who have only joined a system for short period of time, the probability to classify an honest player as suspicious may be significant. We do not consider it an insufficiency of our approach. Service providers with short histories are widely considered high-risk groups, and a client has to be cautious when transacting with them. In fact, there is no good solution to tell an attacker from a honest player when only providing a short transaction history. We may have to rely on other mechanisms to help new servers build reputation. For example, if we want to conduct a low-risk transaction, we may relax behavior testing so that we can choose service from new servers.

There are many interesting directions worth further exploration. In particular, we are interested in developing a more realistic model that captures the behavior of both honest servers and clients. This model should consider not only feedbacks but also other factors of the system that affects user behaviors, e.g., time and dates, location, and transaction types.

References

[1] Aberer K, Despotovic Z. Managing trust in a peer-2-peer information system. In *Proc. the Ninth International Conference on Information and Knowledge Management* (*CIKM2001*), Atlanta, USA, November 5–10, 2001, pp.310–317.

[2] Golbeck J, Hendler J. Accuracy of metrics for inferring trust and reputation in semantic Web-based social networks. In *Proc. the 14th International Conference on Knowledge Engineering and Knowledge Management* (*EKAW2004*), Northamptonshire, UK, October 5–8, 2004, pp.116–131.

[3] Kamvar S, Schlosser M, Garcia-Molina H. EigenRep: Reputation management in P2P networks. In *Proc. the Twelfth International World Wide Web Conference*, Budapest, Hungary, May 20–24, 2003, pp.159–166.

[4] Lee S, Sherwood R, Bhattacharjee B. Cooperative Peer Groups in NICE. In *Proc. INFOCOM*, San Francisco, USA, March 30–April 3, 2003, pp.523–544.

[5] Mui L, Mohtashemi M, Halberstadt A. A computational model of trust and reputation. In *Proc. 35th Hawaii International Conference on System Science*, Hawaii, HO, USA, Jan. 7–10, 2002, pp.2431–2439.

[6] Richardson M, Agrawal R, Domingos P. Trust management for the semantic Web. In *Proc. the Second International Semantic Web Conference*, Sanibel Island, FL, USA, October 20–23, 2003, pp.351–368.

[7] Li Xiong, Ling Liu. Building trust in decentralized peer-to-peer electronic communities. In *Proc. the 5th International Conference on Electronic Commerce Research* (*ICECR 2002*), Montreal, Canada, October 23–27, 2002, pp.1–15.

[8] Wang Y, Vassileva J. Bayesian network-based trust model. In *Proc. IEEE/WIC International Conference on Web Intelligence* (*WI2003*), Halifax, Canada, October 13–16, 2003, pp.372–378.

[9] Yu B, Singh M P. An evidential model of distributed reputation management. In *Proc. the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems* (*AAMAS2002*), Bologna, Italy, July 15–19, 2002, pp.294–301.

[10] Srivatsa M, Xiong L, Liu L. TrustGuard: Countering vulnerabilities in reputation management for decentralized overlay networks. In *Proc. the 14th International Conference on World Wide Web* (*WWW 2005*), Chiba, Japan, May 10–14, 2005, pp.422–431.

[11] Aberer K. P-Grid: A self-organizing access structure for P2P information systems. In *Proc. the 9th International Conf. Cooperative Information Systems* (*CoopIS2001*), Trento, Italy, September 5–7, 2001, pp.179–194.

[12] Elaine B Barker. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication (SP 800-22), 2000.

[13] Liang Z, Shi W. Analysis of recommendations on trust inference in open environment. *Performance Evaluation*, 2008, 65(2): 99–128.

[14] Ray I, Chakraborty S. A vector model of trust for developing trustworthy systems. In *Proc. the 9th European Symposium on Research in Computer Security* (*ESORICS 2004*), Sophia Antipolis, France, September 13–15, 2004, pp.260–275.

[15] Fan M, Tan Y, Whinston A B. Evaluation and design of online cooperative feedback mechanisms for reputation management. *IEEE Transactions on Knowledge and Data Engineering*, 2005, 17(2): 244–254.

[16] Ismail R, Josang A. The beta reputation system. In *Proc. the 15th Bled Conference on Electronic Commerce*, Bled, Slovenia, June 17–19, 2002, pp.708–721.

[17] Zhou R, Hwang Kai. Gossip-based reputation aggregation in unstructured P2P networks. In *Proc. IEEE International on Parallel and Distributed Processing Symposium* (*IPDPS2007*), Long Beach, USA, March 26–30, 2007, pp.1–10.

[18] Huynh T D, Jennings N R, Shadbolt N. Developing an integrated trust and reputation model for open multi-agent systems. In *Proc. Autonomous Agents and Multi Agent Systems* (*AAMAS2004*), *Workshop on Trust in Agent Societies*, New York City, USA, July 19–23, 2004, pp.65–74.

[19] Selcuk A A, Uzun E, Pariente M R. A reputation-based trust management system for P2P networks. In *Proc. the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid* (*CCGrid 2004*), Chicago, Illinois, USA, April 19–22, 2004, pp.251–258.

[20] B Thomas Adler, Luca de Alfaro. A content-driven reputation system for the Wikipedia. In *Proc. the 16th International World Wide Web Conference* (*WWW 2007*), Alberta, Canada, May 8–12, 2007, pp.261–270.

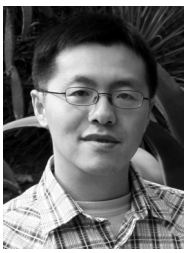
[21] Zhao H, Li X. H-Trust: A robust and lightweight group reputation system for peer-to-peer desktop grid. In *Proc. the 28th International Conference on Distributed Computing Systems Workshops*, Beijing, China, June 17–20, 2008, pp.235–240.

[22] Reches S, Hendrix P, Grosz B J, Kraus S. Efficiently determining the appropriate mix of personal interaction and reputation information in partner choice. In *Proc. the Seventh International Conference on Autonomous Agents and Multi-Agent Systems* (*AAMAS 2008*), Estoril, Portugal, May 12–13, 2008, pp.583–590.

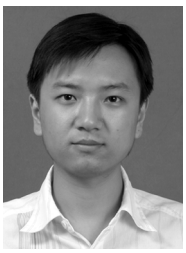
[23] Reece S, Rogers A, Roberts S, Jennings N. Rumours and reputation: Evaluating multi-dimensional trust within a decentralised reputation system. In *Proc. the Sixth International Conference on Autonomous Agents and Multi-Agent Systems* (*AAMAS2007*), Hawaii, USA, May 14–18, 2007, pp.1–8.

[24] K Fullam, K Suzanne Barber. Dynamically learning sources of trust information: Experience vs. reputation. In *Proc. the Sixth International Conference on Autonomous Agents and Multi-Agent Systems* (*AAMAS2007*), Hawaii, USA, May 14–18, 2007, pp.164–171.

[25] Agresti A, Franklin C A. Statistics: The Art and Science of Learning from Data. Prentice Hall, 2006.



Qing Zhang obtained his B.E. degree in automatic control from University of Science and Technology of China in 2001. After that he went to North Carolina State University and obtained his Ph.D. degree comajor in computer science and operations research. His Ph.D. research spans several areas of information security and privacy, including security of reputation systems, sensor networks, and microdata privacy protection. Currently he is developing database kernel in Teradata, that is the world's largest data warehouse company.



Wei Wei received his B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology and Shanghai Jiaotong University, respectively. He is pursuing his Ph.D. degree of computer science at North Carolina State University. His major research interests include distributed computing and network security.

Ting Yu is an associate professor in the Department of Computer Science, North Carolina State University. He obtained his B.S. degree from Peking University in 1997, M.S. degree, from the University of Minnesota, Twin Cities in 1998, and Ph.D. degree, from the University of Illinois at Urbana-Champaign in 2003, all in computer science. His research interests are in trust management, data privacy and security policies. He is a member of ACM.