

# On the Modeling of Honest Players in Reputation Systems

Qing Zhang

Department of Computer Science  
North Carolina State University

Ting Yu

Department of Computer Science  
North Carolina State University

## Abstract

*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 exploited, as an attacker may adapt its behavior and make the above assumption invalid. In this paper, we build a statistic model for honest entities in decentralized systems, which serves as a profiling tool to identify suspicious entities. It can be combined with existing trust schemes to ensure that they are applied to entities whose transaction records are consistent with the statistic model. This approach limits the manipulation capability of adversaries, and thus can greatly improve the quality of reputation-based trust assessment.*

## 1 Introduction

Reputation mechanisms are one of the major techniques for establishing trust in the decentralized systems. In reputation systems, 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 its feedbacks. This process can be viewed as the application of a *trust function*, which takes as input a party's feedbacks, and outputs a trust value that 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 (e.g., [5, 8, 9, 11–13]). But they often can be easily manipulated by adversaries. 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 advantage of a 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. First, we propose a statistical model of the behavior of honest players. We consider the number of good transactions of an honest player as a random variable  $x$ . We show that if an entity's behavior is consistent, 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.

Second, 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.

Third, 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.

## 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, downloading files, etc. 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. 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 timelessness 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<sup>1</sup>. In practice, our scheme can be equally applied to systems where only portions of feedbacks can be retrieved.

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 conducting a transaction with the server.

## 3 The 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 through the qualities of its past transactions. This assumption is in general true for honest players. By honest players, we mean those service providers who tries 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 trusted. Instead, it may still get negative feedbacks from

<sup>1</sup>This may be done through a central server as in online auction communities, or through special data organization schemes in P2P systems [1]

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 proven to be effective when assessing the trustworthiness of honest players. However, it is also common that, given a particular trust function, dedicated attacks can be designed to manipulate and exploit it. 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 launch attacks towards his target users consecutively 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. 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.

### 3.1 A Statistical Model 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.

Given a sequence of transactions,  $trans_1, \dots, trans_n$ , conducted by a server, let  $X_i$  denote the event that  $trans_i$  is a good transaction for  $i = 1, \dots, n$ . Then  $X_1, \dots, X_n$  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 [3]. But again, we do not know the probability  $p$  which is required by existing pseudo random sequence testing algorithms.

### 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  $W_i$ ,  $i = 1, \dots, k$ . If the server is an honest player with trust value  $p$ , we will have

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

$$P\left(\frac{\sum G_i}{n} - p \geq \epsilon\right) < \delta$$

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  $\{|G_1|, \dots, |G_k|\}$  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  $\epsilon$ , then we can conclude that the server is honest.

The confidence of our conclusion depends on the selection of distribution distance threshold  $\epsilon$ . One way to achieve high confidence is to derive the distribution of  $L^1$  norm distance and select  $\epsilon$  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  $\epsilon$  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  $B(m, \hat{p})$ .  $\epsilon$  is selected such that 95% of the distances of the generated sample sets are smaller than  $\epsilon$ .

#### 3.2.1 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 of bad transactions will be revealed. However, if we only consider the most recent  $t$  transactions, it will open doors to periodic attacks, since bad transactions are totally discarded once they

are outside of the most recent  $t$  transactions.

The above observation suggests that we need to consider both the long-term and the short-term behavior of a server so that we can have a balanced assessment of its trust. Thus, we propose a multi-testing approach to check a server’s compliance with the behavior of honest players. Suppose a server has conducted a total of  $t$  transactions. We first check whether the behavior of the server is honest when considering all  $t$  transactions. This corresponds to a long term behavior checking. We then perform the same test by only considering the most recent  $t/2$  transactions. 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 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 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.

In this section, we will design a set of experiments to show the ability of our 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 it the average trust function. Many existing works are based on functions in this form, and further augment it by considering other factors, such as the credibility of the source of feedbacks, transaction context, etc [8, 12].

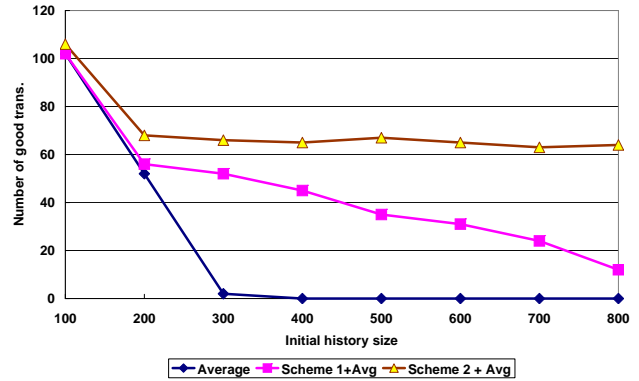


Figure 1. The cost of attackers with varying initial histories: average function

The average trust function takes all transaction history for trust computation, and only considers the long-term behavior of a server. To capture the dynamic behavior of servers more quickly, many works propose to discount old transactions when deriving one’s reputation [4, 7, 10]. The second type of trust function we compare with is the one proposed in [4]. 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 the simulation, we suppose that an attacker has prepared  $H$  transactions as an honest user with trustworthiness 95%. We investigate how many more good transactions is 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 are set to be 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.

Figure 1 shows the result with the average trust function. The  $x$  axis represents the number of transactions 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. “Scheme 1+Average” and “Scheme 2+Average” denote respectively the situations that combine single behavior testing and multi-behavior testing with the average trust function.

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.

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. 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 case 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 “Scheme 2+Average”. The results of “Scheme 2+Average” always outperform that of “Scheme 1+Average”. In particular, even as the attacker includes more transaction 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 behavior 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.

Figure 2 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

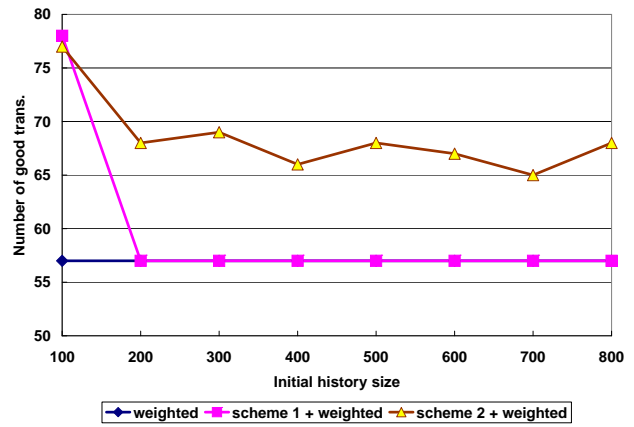


Figure 2. The cost of attackers with varying initial histories: weighted function

trends as in figure 1. With only the weighted trust function, the attacker essential 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 “Scheme 1+Weighted” and “Scheme 2+Weighted” perform well. When the attacker introduces more transactions during the preparation phase, “Scheme 1+Weighted” fails to constraint the attacker, due to similar reasons as discussed above. Meanwhile, the performance of “Scheme 2+Weighted” does not affected by the size of the initial transaction history. It imposes significant costs for the attacker to fulfill its goal.

## 5 Related work

Many trust functions have been proposed recently. Previous work mainly focuses on the design of trust functions based on various feedback collection mechanisms. For example, 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 [12]. In [8], the number of satisfied transactions between each pair is collected to infer a ranking over all peers in the system. There are also some works that deals with non-binary values. For example, Yu et al. proposed a trust combination algorithm for feedbacks that can take values from {positive, neutral, and negative} [14].

Many other trust functions also consider entities’ dynamic behavior. 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. Example works include [6, 7, 10]. The general idea is to assign time-based weights  $w_i$  to each feedback  $f_i$ , such that  $\sum w_i = 1$ .

Hypothesis testing is an important technique in statistical analysis. It checks whether a set of samples follows an expected distribution [2]. 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.

## 6 Conclusion

A single trust function is often not sufficient to provide high-quality trust assessment. 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.

There are many interesting directions worth further exploration. In particular, we are interested in applying our approach to detect collusions among adversaries. This requires us to extend the behavior modeling to consider the structure of transaction histories of a group of entities.

## References

- [1] K. Aberer. P-Grid: A self-organizing access structure for p2p information systems. In *Cooperative Information Systems, 9th International Conference (CoopIS)*, 2001.
- [2] Alan Agresti and Christine A. Franklin. *Statistics: The Art and Science of Learning from Data*. Prentice Hall, 2006.
- [3] Elaine B. Barker. A Statistical Test Suite for Random And Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication (SP) 800-22, 2000.
- [4] Ming Fan, Yong Tan, and Andrew B. Whinston. Evaluation and design of online cooperative feedback mechanisms for reputation management. *IEEE Transactions on Knowledge and Data Engineering*, 17(2), 2005.
- [5] Jennifer Golbeck and James Hendler. Accuracy of metrics for inferring trust and reputation in semantic web-based social networks. In *International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2004.
- [6] T. D. Huynh, N. R. Jennings, and N. Shadbolt. Developing an integrated trust and reputation model for open multi-agent systems. In *Autonomous Agents and Multi Agent Systems (AAMAS 04), Workshop on Trust in Agent Societies*, 2004.
- [7] R. Ismail and A. Josang. The beta reputation system. In *15th Bled Conference on Electronic Commerce*, 2002.
- [8] S. Kamvar, M. Schlosser, and H. Garcia-Molina. EigenRep: Reputation Management in P2P Networks. In *Twelfth International World Wide Web Conference*, 2003.
- [9] Seungjoon Lee, Rob Sherwood, and Bobby Bhat-tacharjee. Cooperative Peer Groups in NICE. In *INFOCOM*, 2003.
- [10] Indrajit Ray and Sudip Chakraborty. A Vector Model of Trust for Developing Trustworthy Systems. In *9th European Symposium on Research in Computer Security (ESORICS'04)*, 2004.
- [11] Matt Richardson, Rakesh Agrawal, and Pedro Domingos. Trust Management for the Semantic Web. In *Second International Semantic Web Conference*, 2003.
- [12] Li Xiong and Ling Liu. Building Trust in Decentralized Peer-to-Peer Electronic Communities. In *The 5th International Conference on Electronic Commerce Research (ICECR)*, 2002.
- [13] Wang Y. and Vassileva J. Bayesian Network-Based Trust Model. In *Proc. of IEEE/WIC International Conference on Web Intelligence (WI)*, 2003.
- [14] Bin Yu and Munindar P. Singh. An Evidential Model of Distributed Reputation Management. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2002.