Available online at www.sciencedirect.com
ScienceDirect
Performance Evaluation 65 (2008) 99–128
PERFORMANCE EVALUATION
An International Journal
www.elsevier.com/locate/peva

Analysis of ratings on trust inference in open environments

Zhengqiang Liang*, Weisong Shi

Wayne State University, United States

Received 8 March 2006; received in revised form 7 December 2006
Available online 29 April 2007

Abstract

Ratings (also known as recommendations) provide an efficient and effective way to build trust relationship in the human society, by making use of the information from others rather than exclusively relying on one's own direct observations. However, it is uncertain that whether the rating can play the same positive effect in the open computing environment because of differences between the computing world and human society. We envisage that there are two kinds of uncertainties: *the uncertainty resulting from rating aggregation algorithms* and *the uncertainty resulting from other algorithm-independent design factors*, which are coined as *algorithm uncertainty* and *factor uncertainty* in this paper. The algorithm uncertainty is related to such a problem: are the complex aggregating algorithms necessary? The factor uncertainty refers to how the performance of ratings is affected by all kinds of factors, including trust model design related factors and trust model design independent factors. In this paper, we take an initial step to answer these two uncertainties. First, we study the effect of all factors based on a simple averaging rating algorithm in terms of several proposed performance metrics. Then we compare different rating aggregation algorithms in the same context and platform, focusing on several relevant metrics. The simulation results show that ratings are not always as helpful as what we expected, especially when the system is facing malicious raters and highly dynamic peer behaviors. In certain circumstances, the simple average aggregation algorithm performs better than the complex ones, especially when there are considerable number of bad raters in the system. Considering the system dynamics, the cost of the algorithm design, and the system overhead, we argue that it is not worth putting too much energy on the design of complex rating aggregation schemes for trust inference in open computing environments.

Keywords: Rating; Trust inference; Open environment; Algorithm uncertainty; Factor uncertainty; Performance evaluation; Analysis

1. Introduction

Ratings (also known as recommendations) from our family members, friends and the people with high reputation are treated as reliable information sources, which are helpful and time-saving for human beings to explore the quality of other people or services. Hinted by the human social network, researchers are of great interest to employ ratings in the computing society [16]. However, whether the ratings in the computing world has the same positive effect is still an open problem, because of the following fundamental differences between the computing world and human society:

* Corresponding address: Wayne State University, Computer Science, 5143 Cass Ave, 312 State Hall, 48202 Detroit, MI, United States. Tel.: +1 313 577 4786; fax: +1 313 577 6868.
E-mail addresses: sean@wayne.edu (Z. Liang), weisong@wayne.edu (W. Shi).

0166-5316/$ - see front matter
doi:10.1016/j.peva.2007.04.001

(1) *Dynamics*. Dynamics in the computing world is more than in human society, which is shown in two aspects. First, is the *quality dynamics*. In the computing world autonomous peers carry two roles at the same time, the service provider (we will call it *SP* simply in the rest of the paper) and the rater. On the one hand, the quality change of a *SP* can make the honest raters give out wrong ratings, thus making the system punish the honest rater unfairly. On the other hand, the quality change as a rater can mislead the system in the process of discovering good *SP*s. Peers change their qualities as *SP* and rater more frequently than the corresponding case in our human society. The second dynamics is the *lifetime dynamics*. Autonomy in open computing environments allows peers to switch to be online or offline at anytime, which bring more dynamics for the lifetime than human society; (2) *Limitation of storage.* Though the storage cost reduces significantly nowadays, considering the applicability, scalability and optimization, we have to limit the storage of every peer in the real implementation so that the memory fading process of peer is normally faster than that of human, which restricts the scope of the system one peer knows; and (3) *Diameter of interaction*. Since it is quite stable for human sociability, the diameter of interaction of human is much smaller than that of the computer environment. In the computing world, peers might know every peer with the help of networks, while human beings normally keep a much smaller circle of people to interact with, e.g. people within the same village. These differences bring the uncertainty whether the rating can play the same positive effect in the open computer environment as in human society.

We envision that there are two kinds of uncertainties: *the uncertainty resulting from rating aggregation algorithms* and *the uncertainty resulting from other algorithm independent design factors*, which are coined as *algorithm uncertainty* and *factor uncertainty* in this paper. The core of research on ratings is the design of the rating aggregating algorithm. The algorithm uncertainty is related to such a problem: is the complex aggregating algorithms worth more than the simple algorithm? The factor uncertainty comes from how the ratings affected by algorithm-independent design factors, e.g. environment-related factors. Making these two kinds of uncertainties clear is very important for the design of trust and reputation systems in open environments, e.g. peer-to-peer systems, inside which peers (also known as agents in AI community)[1] interact with each other while making decisions independently to share resources. To the best of our knowledge, few researchers have been aware of these two uncertainties and have studied the actual worthiness (effect) of ratings in the open computer environment. They just take for granted that the rating is always useful, and focus on developing the complex aggregating algorithms to try to filter out the bad rating becomes the major research direction. In this paper we take the first step to investigate the effect of these uncertainties. Specifically, we intend to answer the following questions: *Are ratings always helpful? How to configure factors related to the rating so as to improve the performance of the system? How is the rating affected by other factors, such as the quality of raters, the scale of the system, the dissemination mode, and so on? Are the complex aggregation algorithms better than the simple ones?*

To answer the above questions, we abstract a basic decentralized trust inference model for open environments. In this model, each peer assigns a trustworthiness value for a set of peers of interest. The trustworthiness value is derived from the self-experience information (interaction-derived or first-hand information) and ratings (second-hand information). Given this abstracted model, we first systematically evaluate the effects of algorithm-independent factors based on a simple average rating algorithm. The algorithm-independent factors can be categorized as *trust-model related* factors and *environment-related* factors. We then compare the performance of all state-of-the-art rating aggregating algorithms with simulation. The simulation platform is developed with Netlogo, a popular multiple-agent simulation tool in the AI community [25]. Its GUI allows us to tune and adjust all factors in a flexible way. To evaluate the effect of ratings, we propose ten novel performance metrics, including both direct and indirect metrics. The simulation results show that in the open environment, ratings behave different from human society, and not as good as expected; the capacity of the storage to hold self-experience information dominates the effect of the rating; setting a low weight to the rating in a dynamic environment is helpful to get better performance; the complex algorithms do not always outperform the simple counterparts.

The major contributions of this paper include five-fold: (1) We abstract a decentralized trust inference model, based on both self-experience information and rating information; (2) We develop a general simulation platform to investigate trust inference models by allowing fine-tuning of multiple factors and algorithms. We expect this platform will be an ideal vehicle for other performance studies; (3) We define several novel performance metrics to measure the

[1] In the rest of the paper, we will use *peer* and *agent* interchangeably.

performance of the rating algorithms; (4) We have conducted a comprehensive analysis of the effect of ratings on trust inference in open environments in terms of the proposed performance metrics. We argue that because of the existence of dynamics and the limitation of the storage, we can not count too much on ratings; and (5) We take an initial step to compare all state-of-the-art aggregating algorithms within one common platform, and get several interesting results.

The rest of the paper is organized as follows. Related work and discussions are described in the following section. Section 3 abstracts a simple trust inference model. Section 4 explores the performance evaluation space of the trust inference model. Based on the performance metrics introduced in Section 5, we present the comprehensive simulation and analysis in Section 6. The extensive comparison of several state-of-the-art aggregation algorithms is depicted in Section 7. Finally, we conclude in Section 8.

2. Relate work and discussions

Our work is inspired by a large amount of previous work on reputation-based systems [3,17,26] and trust inference in P2P systems [1,10–15]. To the best of our knowledge, we are the first to systematically analyse the effect of ratings on trust inference in open environments. Next, we will list some related work on rating aggregating algorithms, trust models and reputation systems.

Trust inference or reputation-based systems has been a hot topic and studied in the literature [13,2,18,19,23,27–30, 20,32,31]. Basically, many researchers are advocating the use of ratings and prefer to complex rating aggregation algorithms to try to filter out the bad ratings [2,18,29,30,20,21,6,9,24]. The key to the success of ratings is the rating aggregation algorithm, i.e. how to integrate others' ratings into the trustworthiness derivation. Wang et al. [23,22] suggest that, averaging should be applied only for stranger raters, but for acquaintances, their ratings should be weighted. We call this approach *half weighted* method in the following sections. Yu et al. [19,28–30] give another thought on this issue. They believe that only ratings from witnesses, who have interacted with the ratee (We call the peer which is recommended by raters as ratee), are useful. In their weighted majority algorithm, only the ratings from witnesses are aggregated, and the weight of witnesses is decreased if the rating is different from his own recognition. Different from Yu et al.'s approach, Srivatsa et al. [21] argue that, the weight of ratings should be based on the similarity of the experience between the rater and the peer itself. We denote this approach as *personalized similarity measure (PSM)*. Finally, Jøsang et al. propose to aggregate the ratings and to update the weight of raters through deriving the expectation of the *Beta* distribution [2,6,9,24]. All these algorithms are complex algorithms considering the complexity of the algorithm design and the workload in the system running. Though noting the potential advantages of ratings, Resnick et al. [17] challenge the feasibility of the distribution of feedbacks, from the point of the expensive cost for the feedback distribution. Holding the same view, Liang and Shi [12,13] suggest to treat the ratings from different raters equally considering the dynamics of P2P systems. They argue that the simple averaging rating is good enough considering the simplicity of the algorithm design, and the low cost in the system running. The above approaches reflect the state-of-the-art of rating aggregating algorithms. There are other approaches having been proposed in different application contexts. Kamvar et al. [10] present EigenTrust, a distributed and secure method to compute global trust values based on "Power Iteration". In EigenTrust, peers extend their social circles by the ratings from their acquaintances, which is similar to the pull model analysed in this paper. In [10] several threat models are described and analysed. EigenTrust addresses these weakness by assuming there are pretrusted nodes in the system, which is not applicable in distributed open environments. Zhou and Hwang propose PowerTrust [32] and GossipTrust [31], which leverage the power-law feedback characteristics and the power of gossip to disseminate feedbacks and reputation data, and finally derive a global trustworthiness value for each peer. These approaches [10,32, 31] share the same goal of finding a global trustworthiness value for each peer, using different distributed approaches. In this paper, we focus more on those approaches allowing each peer to have different trustworthiness values, which we felt fit open environments better.

Several recent proposals try to filter out the bad ratings through introducing another evaluation model dedicated to evaluate the credibility of the ratings [2,18,23,27–30], where they try to add the weight to the rating based on the evaluation of raters. At the same time for the evaluation, peers adjust the weight for different raters based on the correctness of ratings in the history. These approaches are close to the social network, so they are applicable to the relatively stable environment. In open environments such as P2P systems, however, these approaches will lose their applicability because they are not sensitive enough to catch the dynamics of the system.

It is worth noting that the comparison of different rating aggregation algorithms is one of the two objectives of this paper, which distinguishes our work with previous work which focuses on propose one specific aggregation algorithm. In previous work, people who proposed the aggregating model only focus on their own model. Whitby et al. [24] conduct a similar analysis on the unfair ratings based on Bayesian reputation systems. However, [24] just focuses directly on the fair and unfair ratings, and does not involve other factors, such as the dynamics of raters and *SP*s. Also, their evaluation of the effect in their work is just according to the average rating errors. Recently, Srivatsa et al. [21,20] gives another similar analysis in TrustGuard; however, their approach is pursuing from the angle of *SP*s, while our approach is from the angle of raters. Also, we consider more factors than their work. Thus, our work compliments to their work very well. Despotovic and Aberer [4] comprehensively compare the performance of two groups of P2P reputation management approaches – probabilistic estimation and social networks – against various classes of collusive peer behaviour and analyse their properties with respect to the implementation costs they incur and trust semantics they offer to the decision makers. Their work is similar to our work of the evaluation with different rating algorithms. But we consider more behaviour patterns of service provider and rater.

3. The trust inference model

To analyse the effect of ratings on trust inference, we first abstract a generic decentralized trust model in open environments. We define trust as *the subjective probability by which a peer A expects that another peer B performs a given action as good as expected*. The trust is quantified by the trustworthiness value. Next, some basic assumptions are listed.

3.1. Assumptions

To reduce the complexity of modelling, we make the following two assumptions: (a) We assume the quality for a peer to provide services is independent of its quality to provide ratings (denoted as the *trust-independent assumption*). That is, the weight of its ratings about others is not related to its own trustworthiness for the service providing. To validate this assumption, we compare two different approaches in Section 4; (b) We assume each peer has a unique and stable ID, thus reputation and trustworthiness will make sense. This can be implemented by using either the Public Key Infrastructure (PKI) or a centralized server; and (c) Each peer has limited storage space because of the concern of implementation overhead.

3.2. Basic trust inference model

Traditionally there are two major classes of trust models. The first class is the *Central* model (CM), which has a central trust point. Each entity in the *Central* model has the same opinion as what the central trust point thinks. Reputation-based systems [5], e.g. eBay [7], are the typical examples of the CM model. The certificate authority (CA)- based trust model, which has been widely deployed in e-commerce [1,17,26], is another typical example. The second class is the *Transitive* model (TM) [8] which has a transitive trust chain. In TM, the rating from the rater is highly emphasized for the trustworthiness. Actually, the CM model can be seen as a special case of the TM model with just one-step transitive relationship and the rating being totally trusted. Perils including collusion and wrong ratings are big threats for the TM model. However, making use of the rating is also the merit for the TM model when the rating is good, which is helpful to discover the quality of other peers even without any mutual interaction. It deserves further study to find the suitable tradeoff.

No matter what kind of trust models, two pieces of basic information, self-experience information (interaction-derived or first-hand information) and ratings (or second-hand information), are necessary to calculate the trustworthiness. In order to make our evaluation general enough, we employ a simple decentralized trust model in which the trustworthiness T is derived from two pieces of information mentioned above, as shown in Eq. (1),

$$T = W \times I + (1 - W) \times R \tag{1}$$

where I is the value of the self-experience information, R is the value of rating after aggregation, and W is the weight of I (correspondingly, $1 - W$ is the weight of R). For a neighbour k, we calculate its value I with the following

equation:

$$I = \begin{cases} 1 & S \geqslant \theta, \\ \dfrac{S}{\theta} & 0 < S < \theta, \\ 0 & S \leqslant 0. \end{cases} \tag{2}$$

S is the accumulative credits for the direct transaction. Peer will give the *SP* a certain credit after each transaction based on the quality of the received service. We will give more details about the service quality in the following sections. The θ is a threshold. Regarding to the value of R, different aggregating algorithms have different ways to set this value, which is detailed in Section 4.5. All values of T, R and I are taken from 0 to 1. The model is an independent and personalized model, that is, each neighbour has its unique trustworthiness from a peer's perspective, and two different peers are not necessary to share the same view of trust.

4. Evaluation space: Algorithms and factors

To evaluate the effect of rating in a comprehensive way, we have conducted two related evaluations in the following sessions, including comparison of different aggregating algorithms and evaluation on the effects of algorithm-independent factors, as shown in Fig. 1. Many factors could cast a reflection on the performance of ratings, as abstracted and illustrated in the right frame of Fig. 1. We classify them into two broad categories: *model design related factors* (they are simply called the *design* factors) and *model design independent factors* (they are simply called the *system* factors). The *design* factors include five factors which can be adjusted during the trust model design, while the *system* factors include three factors which are used to simulated the system environments. Our analyses in Section 6 aim to figure out in a certain prefixed environment simulated with the *system* factors, what is the performance of the trust model with a certain set of the *design* factors, and how to select the correct *design* factors. In the figure, the top part of each rectangular box lists the name of factor, while the lower part of each box represents the possible values of this factor. In addition to these two categories, some other important design concerns are also discussed. Since it is too complex to conduct all experiments on these factors with respect to all five algorithms, we instead decide to use the simplest but most generic one, the average aggregating algorithm, as the underlying rating aggregation algorithm when we analyze the effect of these algorithm-independent factors. There are five state-of-the-art algorithms to be compared, which represent all solutions in the context of distributed trust inference, as listed in the left column of the figure. In the following section, we first clarify several key notions in the system, then briefly explain algorithm-independent factors, and five rating aggregation algorithms.

4.1. Notions

From the angle of service providing and receiving, a peer can be a ***SP*** or a **service receiver**. From the angle of rating, a peer can be a **rater** which provides the rating, or a **ratee** which is the object to be rated (or recommended). After a peer joins the system, it will know some peers and put them into its *neighbor list*. We call the peers in the neighbour list **neighbours**, which are the candidates to ask for cooperations later. The neighbour with the highest trustworthiness value will be chosen when a peer needs help from its neighbours, and we call this neighbour the **cooperator**. When neighbours are known as bad peers, they will be purged from the neighbour list and recorded in the *blacklist*. It may have different approaches to deal with these blacklisted peers. In this paper, since we are targeting to an open systems, we assuming that there are enough peers to be the cooperator candidates, so we don't allow the peers in the blacklist to be chosen as a new neighbour again. For each neighbour, the peer maintains a corresponding *rating queue* to hold the ratings.

4.2. Trust model design related factors

Trust model design related factors include *the rating dissemination mode (M), the size of neighbour list* (S_N), *the size of the rating queue* (S_Q), *and the weight of ratings* (W). In order to validate the trust-independent assumption, we also introduce the factor *independence assumption (A)* in this type of factors.

Independence Assumption *(A)*. In Section 3, we make the trust-independent assumption, which is the fundamental assumption of this paper. However, one may ask that whether the quality of a peer as a *SP* is consistent with its

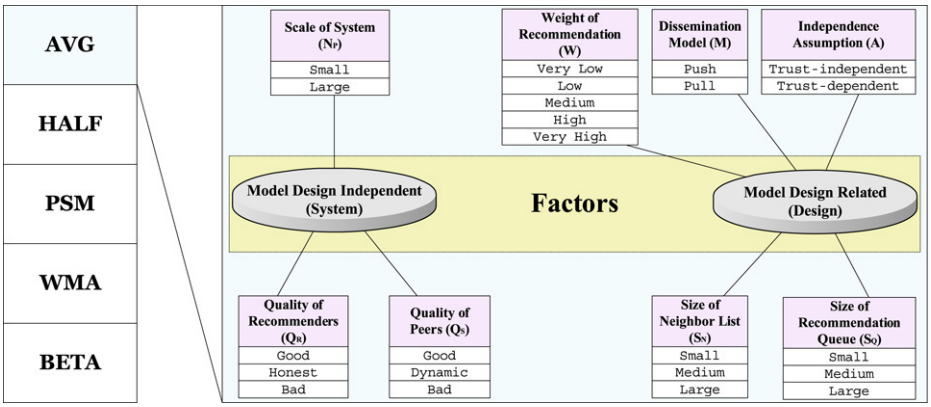


Fig. 1. Evaluation space consists of two parts corresponding to two uncertainties. Among the five algorithms listed in the left column of the figure, we choose the simplest one, the average aggregating algorithm, to evaluate how ratings affected by different algorithm-independent factors listed in the right part.

quality as a rater. To the best of our knowledge, there is no conclusion about this. To answer this question, we consider two school of thoughts here: one is the assumption that the quality for a peer to be a *SP* is independent of the quality for a peer to be a rater, which is denoted as *trust-independent*, the other is on the opposite side by coupling these two together, which is denoted as *trust-dependent*. Corresponding to case *trust-independent*, we use an average approach (**A1**) to derive the final rating value, while corresponding to case *trust-dependent*, we use a simple aggregation algorithm (denoted as **A2**), which simply relates the rating with the trustworthiness of the peer as a *SP* T_r. More formally, we can define $R = \frac{\langle \mathbb{W}, \mathbb{N} \rangle}{|\mathbb{N}|}$, where $\mathbb{W}$ is the weight set, $\mathbb{N}$ is the rating set, and $\langle \mathbb{W}, \mathbb{N} \rangle$ is the Euclidean inner product of set $\mathbb{W}$ and set $\mathbb{N}$. **A1** and **A2** is formalized as **A1**: $\forall x \in \mathbb{W}, x = 1$, **A2**: $\forall x \in \mathbb{W}, x = T_r$. In **A1**, all elements of $\mathbb{W}$ are set to one, while in **A2**, the trustworthiness value for the rater as a *SP*, T_r is the elements of $\mathbb{W}$. Note that in this part we are interested in studying the appropriateness of the trust-independent assumption. The analysis of different rating aggregation algorithms, which are proposed in the context of the trust-independent assumption, will be discussed in Section 7.

Ratings dissemination modes *(M)*. Two dissemination modes of ratings are studied in this paper: *push* and *pull*. In the *push* mode, once a neighbour's trustworthiness value changes over a threshold, its update of the trustworthiness will be propagated to another five neighbours (or less when the size of the neighbour list is less than five) with the highest trustworthiness value. So the push mode is an active mode. In the *pull* mode, a peer chooses five neighbours with the highest trustworthiness to ask for ratings when it needs to recalculate the trustworthiness value of its neighbour k. Since the asymmetry of the neighbourhood, it is possible those chosen neighbours don't have the information of k. In order to increase the hit ratio, every rater sends out five ratings on its five neighbours with highest trustworthiness value, not just one. Different from the *push* mode, all five neighbours included in this reply message will be updated at the requesting peer no matter their trustworthiness values are changed or not since last round. If we want to optimize this blinding update in the pull mode, peers have to monitor all their neighbours and record the neighbours whose trustworthiness value is changed and when the value is changed. This will consume significant storage, and make the system very complex. Both *push* and *pull* mode propagate the bad reputation of the peers recorded in the blacklist.

Size of neighbour list (S_N). The neighbours are recorded in the neighbour list. When peers need help from others, they choose the neighbour with the highest trustworthiness value. Normally, the number of neighbours (the size of the neighbour list) is fixed, but the neighbours can be altered as time goes on. When one neighbour is recognized as a bad peer (the trustworthiness value is below the threshold), it can be purged from the neighbour list, and a new neighbour is randomly chosen from the stranger peers. In a bad community in which most or all peers are bad peers, it is possible that the neighbour list shrinks to a small size. The size of the neighbour list represents the capacity of the peer for holding the self-knowledge information. It is expected that with the large size of the list, the system will gain better performance, which is verified by simulation results, but that will cost more storage and maintenance overhead.

Size of rating queue (S_Q). Similar to the size of the neighbour list, there is a limit on the capacity for every peer to hold ratings from others. FIFO structure is used here to record ratings, which is twofold: (1) every rater has just one

rating for each peer. The new rating given by the same rater will replace its old value and be lifted to the beginning of the queue. (2) If the queue reaches to its maximum size, the oldest value at the end of the queue is removed while the newest one is added to the beginning of the queue. Since in a dynamic environment ratings are somehow unreliable and have limited time-to-live, using more storage to hold such a kind of information is not justified, which is verified by the simulation result.

Weight of rating (W). Because the rating is one of the two components to derive the trustworthiness value, the weight of the rating is a very important factor for the evaluation. If more weight is put on the rating, it means the trust model focuses more on the knowledge of others. In our evaluation, five different weight settings for the rating are used: *Very Low*, *Low, Middle, High* and *Very High*.

4.3. Model design independent factors

Now we are in a position to discuss the model design independent factors, including *the qualities of peers* (Q_R *and* Q_S) and *system scale* (N_P).

Peer quality (Q_R **and** Q_S) **(Behavioural factors)**. Each peer carries two roles in the system, i.e. as a *SP* and as a rater. Correspondingly, there are two kinds of qualities every peer has: quality as a rater Q_R and quality as a *SP* Q_S. As a rater, the quality of a peer can be *Good*, *Bad*, or *Honest*. The good (bad) rater sends out the absolutely correct (wrong) ratings each time based on the ground truth. Honest raters provide ratings exactly based on their own views. The point here is, the honest raters always tell the best-effort truth (i.e. not the ground truth), that is, the current trustworthiness T of the recommended peers in the eyes of the rater. The correctness of the honest rating is related to the extent of correctly understanding the qualities of the recommended peers. Note that it is impossible to ask a peer to provide the ground truth in an open environment without the full global view of the system. Therefore, introducing the good and bad quality is purely for studying the extreme case of peer quality in Section 6.6. As a *SP*, the quality of a peer can be *Good, Bad*, or *Dynamic*. In order to make the system general enough, the approach to classify the *SP* in [13] is adopted. Four qualities of *SP*s are introduced in the simulation: *Good, Low-grade, No-response*, and *Byzantine*. *Good SP*s will always provide good service, which leads to the increase of accumulative credits. The last three qualities are related to the bad *SP*s, and their corresponding services will lead to the decrease of the trustworthiness value. From *Low-grade* to *Byzantine*, the services are turning worse, and the worst service *Byzantine* will bring the maximum trustworthiness value reduction. Good *SP*s always provide good services. Bad *SP*s provide one of the three bad services mentioned above randomly. For a dynamic *SP*, it provides the service with one of the four kinds of qualities uniformly, so that only 25% of its total services are good services.

System scale (N_P). In an open environment, as the scale of the system increases, the effect of ratings might degrade because of the possible increase of the workload and the complexity of communication. The system scale is measured by the number of peers in our analysis. In our simulation, two scales are compared: small scale (300 peers) and large scale (600 peers).

4.4. Other factors

In addition to the major factors mentioned above, there are several other factors needs to be clarified: *interval of dynamics, percentage of bad SPs, dynamic SPs, honest raters*, and *the number of simulation steps*.

Dynamic *SP*s change their qualities in a specific interval (I). Decreasing the value of I can make the *SP*s change their qualities faster, through which we can introduce another kind of dynamics into the system. P_B and P_D are the percentage of bad and dynamic *SP*s respectively in the system. If $P_D \neq 0$, there will be $N_P * P_D$ dynamic *SP*s, $N_P * (1 - P_D) * P_B$ bad *SP*s, and the rest are good *SP*s, that is, $N_P * (1 - P_D) * (1 - P_B)$ good *SP*s in the system. P_H is the percentage of honest raters. If $P_H \neq 0$, then there are $N_P * P_H$ honest raters, and the remaining $N_P * (1 - P_H)$ raters are all bad raters. Finally, S is the running steps of the simulation. Instead of using the physical running time, we use the notion of *step*, which is introduced in the NetLogo [25], to calculate the simulation time. Within each step, peer will finish all the activity including service request, service providing, trustworthiness value update and rating dissemination.

All the factors discussed above and their possible values used in the simulation are summarized in Fig. 2.

	Description	*Possible Values*
A	Assumption of Independence	A1, A2
I	Interval of dynamics	3, 12, 30
M	Mode of dissemination	Pull, Push
N_P	Number of peers	300, 600
P_B	% of bad *SP*s	20%, 70%
P_D	% of dynamic *SP*s	20%, 70%
P_H	% of honest raters	20%, 70%, 100%
Q_R	Quality as rater	Bad, Good, Honest
Q_S	Quality as *SP*	Bad, Dynamic, Good
S	Steps of simulation runs	3000, 3200, 4500
S_N	Size of neighbor list	8, 14, 20
S_Q	Size of rating queue	5, 8, 12
W	Weight of the rating	0.3, 0.5, 0.7, 0.9

Fig. 2. The factors and their possible values in the simulation.

4.5. Algorithms

In the following we briefly describe the five algorithms evaluated in this paper. These five algorithms are the typical aggregating algorithms in the distributed trust inference where every peer holds personalized views on their potential raters. The key role of these algorithms is getting rid of bad raters and obtaining accurate ratings. Note that all these algorithms are proposed in the context of trust-independent assumption, where the quality for a peer to be a *SP* is independent of the quality for a peer to be a rater. Some classical algorithms such as EigenTrust [10] are not included here because it is not able to be applied in such a context.

Average (Avg) [12,13]. In Avg, the rate aggregation R is defined as: $r_{ij} = \frac{\sum_{z=1}^{g} tr_{zj}}{g}$, where r_{ij} is the value of aggregated ratings towards peer j in peer i. tr_{zj} is z's rating towards j. g is the total number of ratings towards j. For the honest rater, tr_{zj} is the trustworthiness value of peer j in peer z, but for the bad rater, this value can be any value depending on what behaviour pattern the rater acts with. Since all the weights of the ratings are equal (to 1), there is no need to update the weight, thus there is no corresponding communication and storage cost for the weight update. This is the simplest algorithm among the five algorithms.

Half weighted (Half) [22,23]. In HALF, the rating aggregation is defined as: $r_{ij} = w_t * \frac{\sum_{l=1}^{k} t_{il} * tr_{lj}}{\sum_{l=1}^{k} tr_{il}} + w_s * \frac{\sum_{z=1}^{g} tr_{zj}}{g}$, where w_t is the weight about the rating from the acquaintance, w_s is the weight about the rating from the stranger, and t_{il} is the weight (trustworthiness) about the rating from rater l. r_{ij} and tr_{lj} are the same as *Avg*. For the weight update: $t_{il} = \alpha * t_{il}^{o} + (1-\alpha) * e_\alpha$, where t_{il} denotes the new trust value that the ith peer has towards rater l; t_{il}^{o} denotes the old trust value. α is the learning rate—a real number in the interval [0,1]. e_α is the new evidence value. In the original paper [22], e_α can be -1 or 1. Authors suggest that, if the rating and the actual experience has considerable difference, e_α will be set to -1, otherwise, 1. However, if the value of α is large, it will make the value of t_{il} fluctuate very fast. So in the simulation, we change this formula a little bit with: $t_{il} = \alpha * t_{il}^{o} + (1-\alpha) * |s^j - t_{lj}^{o}|$ where, s^j is i's self-opinion towards transaction j, and α is set to be 0.51.

Personalized Similarity Measure (PSM) [21]. In PSM, the rating aggregation is: $r_{ij} = \sum_{l=1}^{k} (tr_{lj} * \frac{S(il)}{\sum_{n=1}^{k} S(in)})$, where similarity $S(in)$ equals to $1 - \sqrt{\frac{\sum_{r \in IJS(i,n)} (\frac{\sum_{v \in I(i,r)} tr_{iv}}{|I(i,r)|} - \frac{\sum_{v \in I(n,r)} tr_{nv}}{|I(n,r)|})^2}{|IJS(i,n)|}}$. $S(in)$ is the personalized similarity between peer i and rater n, which is similar to the weight of the rater in *Half*. tr_{lj} has the same definition as that in *Avg*. Let $IJS(i,n)$ denote the set of common peers with whom both peer i and n have interacted, and I(i, r) denotes the collection of interactions between peer i and r. The similarity between peer i and n is computed based on the root mean square of the differences in their opinion over the nodes in IJS(i, n). Note that in our simulation, what we use is actually a slight variant of the original algorithm proposed in [21]. However, our basic ideas are the same, i.e. based on the personalized similarity towards the same interaction.

Weighted Majority Algorithm (WMA) [30]. In WMA, the rating aggregation is defined as: $r_{ij} = \sum_{l=1}^{k} (lr_{lj} * \frac{t_{il}}{\sum_{n=1}^{k} t_{in}})$, where lr_{lj} is the firsthand information of peer l about j. For the weight update: $t_{in} = \theta t_{in}^{o}$, where t_{in}^{o} is the old weight of rater n in peer i. θ is defined as: $\theta = 1 - (1-\beta)|lr_{nj} - s|$.

Beta model (Beta) [2,6,9,24]. In [9], Jøsang and Ismail propose the beta reputation system which use beta probability density functions to combine ratings and derive reputation towards the raters. In BETA, the rating aggregation is defined as: $r_{ij} = \frac{p+1}{p+n+2}$, where p is the number of positive ratings (>0.5) towards peer j, and n is the number of negative ratings (<0.5) towards peer j. Similar to *Avg*, in this algorithm we don't need to maintain the weight of raters. The raters in the upper and lower 10% will be excluded from the aggregation. This algorithm is also a simple algorithm without the weight update. Note that several variants have been proposed since they introduced the original idea; however, in our simulation, we just use a simple algorithm plus a simple filter to exclude the bad ratings.

5. Performance metrics

Before presenting our analysis results, we propose several important performance metrics against which to evaluate the effect of ratings, including *direct* and *indirect* metrics.

5.1. Direct metrics

Direct metrics can be observed or directly calculated from the results, including *convergent time* τ, *selection hit* ς, *number of service* η, *consensus recognition* ϱ, *workload* ω, *correction rate* ϕ, *storage cost* ζ, and *running time of simulation* Υ.

Convergent time. The convergence time is a useful metric to evaluate the evolution of the whole system, especially the bootstrap phase of the system. A lot of previous work focuses on the system behaviour when the system is stable. Here we argue that the convergence time is an important metric to measure how fast the system can reach a stable point, especially in a dynamic environment. To understand the convergence of the system thoroughly, four kinds of convergent times are defined.

Consensus convergent time (τ_c). At a certain time, different peers will form their own local views on other peers to form a snapshot of the system. A peer can be viewed as a good peer if its trustworthiness value is larger than a certain threshold T_{good}, as a bad peer if its trustworthiness value is lower than a certain threshold T_{bad}, or as a dynamic peer if its trustworthiness value is between T_{good} and T_{bad}. There are five possibilities for one peer in the view of others in the system scope: (1) *Good*. Only all the peers knowing the target peer (knowing means the target peer is within the neighbour list or the blacklist) reach the consensus that the target peer is a good peer, the target peer is treated as a good peer; (2) *Bad*. Similar to first case, but the target peer is known as a bad peer instead; (3) *Dynamic*. All peers knowing the target peer think the target peer is a dynamic peer; (4) *Unknown*. No peer knows the target peer; and (5) *Not consistent*. The views on the target peer are not consensus. τ_c is the time after which the consensus status for all five kinds of peers doesn't change any more. For some configurations, it is possible that there is no such a convergence point at all.

Neighbor list convergent time (τ_n). The neighbour list changes every moment to add new neighbours and drop the bad neighbours. After a certain point τ_n, the neighbour list may stay unchanged or changed very rarely, that is, reaching a stable stage.

Rating convergent time (τ_r). It is expected that after certain point the number/percentage of good ratings should stay in a fixed range. This point is defined as the convergent time of the rating.

Service convergent time (τ_s). Different from τ_r, τ_s defines the convergence from the angle of service. It is the time after which the number of good(/bad) services keeps stable in every moment.

It is worth noting that in a dynamic changing environment, it is very difficult to formally define the convergence time. However, we take the following schemes in our analysis. Given a series of collected data, we take a fix window and calculate the average variance as following equation $\Delta = \sum_{i=1}^{N} |\frac{x_i - \bar{x}}{\bar{x}}|$. If the difference of two continuous average variances is close enough (e.g. less than a predefined threshold), we take the left boundary of the first window as the convergent point.

Selection hit (ς). When the system reaches a point of τ_n, the neighbour list gets to a stable stage. The total number of good cooperators at this moment is called selection hit. When the cooperator of every peer is a good cooperator, the value of ς is equal to the total number of peers (N_P) in the system.

Number of services (η). This metric includes *the number of good service* η_g and *the number of bad service* η_b. η_g is the most direct metric to show the benefit of the system affected by the trust model. If we fixed other parameters related to the trust model, and just change the factors related to the rating, then η will be the indication of the performance

of the rating, and η_g reflect the benefit of the system gets from the rating. Thus, in the following section, when we mention the benefit to the system, we refer to the value of η_g. The larger of η_g with the same running time, the more effective of the model is. The total number of service η is defined as the sum of η_b and η_g: $\eta = \eta_b + \eta_g$.

Consensus recognition (ϱ). This metric reports the evolution of the system, including the number of peers with consensus status 'Good' (ϱ_g) and 'Bad' (ϱ_b).

Workload (ω). In every step, the total number of ratings accepted by all peers is called the step workload ω_S, which is the indication of the system instantaneous communication cost. We denote the system accumulative workload ω_A as the accumulative number of total ratings within a specific number of steps. A good design is expected to bring lower workload to the system.

Correction rate (ϕ). This metric is used to measure the capacity of the rating to catch the correct status of the system. Similar to the definition of workload, there are also two kinds of correction rates, step correction rate ϕ_S which is the ratio of the number of good ratings to the number of ratings ω_S in every step, and accumulative correction rate ϕ_A. which is the ratio of the accumulative number of good ratings to the accumulative number of ratings ω_A. ϕ can be treated as one metric to evaluate the effect of different parameter combinations. Higher value of ϕ is expected if the combination is optimal.

Storage cost (ζ). In some special environments, such as wireless network, there is a strict limitation of storage. Storage cost of the rating algorithm will be one of the considerations for the system developer. The storage cost consists of three parts: neighbour list, rating queue, and blacklist. When these data structures are enlarged, not only the storage but also the system workload will be increased because the number of information to be interchanged is increased. On the other hand, this metric can indirectly show the severity of the environment, because when the environment turn worse, more information needs to be stored to help pick up the good peers.

Running time of simulation (Υ). It is the total time used after a specific number of steps. The simpler of the aggregating algorithm and the less severity of the environment can make the running time shorter.

5.2. Indirect metrics

In addition to the direct metrics defined above, we are also interested in two indirect metrics: *benefit speedup* β and *efficient factor* ε.

Benefit speedup (β). Good services are provided by two kinds of *SP*s. Good *SP*s always provide good services, while the dynamic *SP*s provide good services with 25% of their total services. So we can get an expected value E on the percentage of good services once P_B and P_D are fixed: $E = 0.25 * P_D + (1 - P_D) * (1 - P_B)$. If the rating plays as a positive factor in the trust model, we would expect the system to get more good services η_g. In order to describe this kind of improvement, we introduce the benefit speedup β: $\beta = \frac{\eta_g/(\eta_g+\eta_b)}{E}$. i.e. the ratio of the simulation benefit to the expected benefit. We expect that the larger value of β, the more positive the ratings are.

Efficient factor (ε). In the system the convergence time is expected to be shortened if ratings are fully utilized. In this case, the more number of ratings, the shorter the convergent time. However, a large amount of ratings can bring heavy workload to the system. In order to show the efficiency of each unit of rating contributing to the system convergence, we define the efficient factor ε: $\varepsilon = \frac{\tau_n}{\omega_A}$. Here we are using the neighbour-list convergent time τ_n, because in the simulation, only this convergent time is available for all cases. ω_A is the total workload within time τ_n.

6. Performance evaluation I: Factor uncertainty

Now we are in a position to describe our simulation results and discussion. We build a simulation platform using NetLogo [25], a very popular multi-agent simulation tool in the AI community, which can easily model the parallel and independent agents, to simulate interactions among peers. With NetLogo we have developed a friendly GUI-based user interface to control the simulation, and we can easily tune different parameters to form different configurations. In the following, some details about the simulation will be discussed first. Then we present the configuration of the baseline for the comparison in the rest of the section. Finally, a complete study and discussion are depicted based on the metrics we describe in Section 5.

EXECUTION SKELETON OF REQUESTER:

```
while TRUE do
  pick up one peer k with the highest T;
  send the service request to k;
  after transaction, adjust k's T, and update the history list;
  if k's T < Threshold then
    put k into the blacklist;
  end if
  rating update and dissemination (with push or pull model);
  update the weight of rater if the algorithm needs;
end while
```

EXECUTION SKELETON OF SERVICE PROVIDER:

```
while TRUE do
  get the service request from i;
  if i is in the blacklist then
    Reject i's request;
  else
    provide the service to i;
    if neighbor list is not full then
      add i to the neighbor list;
    end if
  end if
end while
```

Fig. 3. The skeleton of system execution from the perspective of service requesters and service providers.

6.1. Simulation details

Fig. 3 shows the brief skeleton of the system execution from the perspective of service requesters and *SP*s separately. In each step, each peer finishes either the job of service requester, or the job of *SP*, or both when the peer requests the service and receives the service request at the same time. In the simulation, when the simulation starts or the neighbour list becomes empty, a new peer is chosen as a new cooperator randomly. At this point there is no self-experience information or rating for the new chosen cooperator. The system will assign a low trustworthiness value $T_0 = 0.4$ to the new cooperator. Afterwards, the trustworthiness values change according to variations of I (ΔI) and R (ΔR) based on this value. In other words, in the simulation the Eq. (1) becomes $T = T_0 + W \times \Delta I + (1 - W) \times \Delta R$. When $T > 1$, it will be reset to 1. When $T < 0$, it will be reset to 0.

To make our simulation as close to the real P2P systems where peers often go offline, we also simulate the offline peers by assigning every peer a random lifetime (or Time-To-Live) within the step range [50, 100]. After reaching the lifetime, the peer will not respond to any service request, and won't be counted in the statistics either. After one more step, the peer comes alive again with a new life time randomly chosen from the range [50, 100]. In every step peers send out a service request with the possibility of 50%. In this part of evaluation, we compare all the algorithm-independent factors under the *Avg* algorithm. The reason to use *Avg* is because *Avg* is a simple and general algorithm which can be used in any trust model. Using this simple model to compare the results of different metrics is direct and fair. We also use part of the same metrics to compare different rating algorithms, and the evaluation results can be seen in Section 7.

6.2. Baseline

In order to illustrate the simulation results clearly, we develop a normal configuration with 300 peers as the baseline for comparison purpose. The configuration of the baseline is listed in the top portion of Fig. 4. In the baseline there are a small part of bad peers and bad raters, and $P_D = 20\%$, i.e. $300 * 0.2 = 60$ dynamic peers in this configuration. From the value of $P_B = 0.2$, it can be known that $(300 - 60) * 0.2 = 48$ peers are bad *SP*s, and $(300 - 60) * 0.8 = 192$ peers are *SP*s. All raters in the baseline are honest. The weight of the rating is set as 0.3, which means the rating is less important than the self-experience information. The dissemination mode is *push*.

The bottom portion of Fig. 4 shows all the simulation results in Section 6 in terms of the metrics defined in Section 5. The reason to put these results together are for better comparison and analysis. To distinguish with the *temporal* results discussed below, we call this type of results *final* results. Note that the unit of all τ_c, τ_n, τ_r, and τ_s is "50 steps", and the unit of ω, η_b, η_g is "10^3". For the values of ϱ_g and ϱ_b, they are represented in the form of three-tuple: (# of Good *SP*s, # of Bad *SP*s, # of Dynamic *SP*s). Here is an example on how to understand this three-tuple. In this figure $\varrho_b = (0, 19, 20)$, which means there are totally $0 + 19 + 20 = 39$ *SP*s are known as bad *SP*s. Within these *SP*s actually there are 0 good *SP*s, 19 bad *SP*s, and 20 dynamic *SP*s. For the simplicity of comparison, the values of β and ε have been normalized. In some sections when some metrics don't have the corresponding values, or the values are meaningless, we mark a "N/A" in the corresponding cell in the table. In the leftmost column, the "Sec No." is the section number corresponding to the section where the results in the right at the same row to be described. C_{New} stands

Configurations of base line

	A	I	M	N_P	P_B	P_D	P_H	Q_R	Q_S	S	S_N	S_Q	W
Conf	A1	12	Push	300	20%	20%	100%	Honest	Dynamic	4500	8	5	0.3

Results from the view of 12 metrics

Sec No.	C_{New}	*4 Conv Time*				*Hit*	*BadSer*	*GoodSer*	*2 Conse Recg*		*WkLd*	*BenSdup*	*Storage*
		τ_c	τ_n	τ_r	τ_s	ς	η_b	η_g	ϱ_b	ϱ_g	ω	β	ε
6.2	**Base Line**												
	Baseline	N/A	10	12	N/A	300	174.10	419.10	(0, 19, 20)	(151, 0, 0)	8447.58	1.02	1
6.3	**Weight of Rating**												
	W = 0.5	55	4	22	10	300	324.38	351.65	(59,48,53)	(17,0,0)	5292.32	0.75	0.99
	W = 0.7	16	17	13	18	295	349.44	178.23	(59,36,35))	(47,0,0)	51238.03	0.49	0.50
	W = 0.9	10	19	10	5	245	466.98	201.58	(15,4,8)	(64,0,0)	2091.25	0.44	2.30
6.4	**System Scale**												
	N_P=600	N/A	11	15	N/A	598	485.63	850.51	(0, 39, 37)	(279, 0, 0)	14976.07	0.92	0.48
6.5	**Storage Capacity**												
	S_N=14	7	5	4	1	300	104.26	564.93	(0,6,2)	(120,0,0)	895.76	1.22	3.75
	S_Q=8	27	19	16	20	300	204.98	463.11	(0,25,13)	(136,0,0)	5866.53	1.01	0.94
	S_N = 20 N_P=600	4	1	2	1	600	44.06	1291.96	(0,3,0)	(185,0,0)	1426.82	1.40	1.66
6.6	**Quality of Raters**												
	P_H=20%	42	41	4	2	140	479.10	188.73	(0,1,1)	(76,0,0)	1792.70	0.41	3.22
	P_H=70%	23	21	8	9	193	351.09	316.82	(0,13,7)	(110,0,0)	2828.96	0.69	1.76
6.7	**Quality of *SP*s (S=3000)**												
	P_B=20%	N/A	10	12	N/A	300	143.89	301.35	(0, 19, 20)	(151, 0, 0)	4442.42	0.98	1
	P_B=50%	N/A	51	12	N/A	285	150.43	294.62	(0,64,34)	(102,0,0)	14822.77	1.47	0.53
6.8	**Dissemination Mode (S=3200)**												
	M = Push	N/A	10	12	N/A	300	152.14	330.04	(0, 19, 20)	(151, 0, 0)	5045.84	0.99	1
	M = Pull	20	16	12	5	300	155.97	311.93	(1,10,9)	(138,0,0)	86767.35	0.97	0.04

Fig. 4. Configurations of baseline, and all results of factor uncertainty evaluation in the whole Section 6.

for the new configuration. In C_{New} column each cell shows factor(s) which is changed compared with the baseline. The factors not shown will keep the same value as the baseline settings.

Fig. 5 shows the simulation results during the whole simulation. We call them *temporal* results. The x-axis of all subfigures stands for the number of simulation steps. Fig. 5(a) measures the correction rate of the rating, where y-axis is the number of the rating with unit 10^3. There are totally three plots in Fig. 5(a), which stands for the "Total number of ratings", "Number of good ratings", and "Number of bad ratings" respectively. We can see that, the number of bad ratings is very small, and decreases as time passes. Almost all the ratings are good ratings, so the plot representing the good rating is almost coincident with the one representing the total ratings. Another important fact in this figure is, the plot doesn't get stable within 4500 steps, and the good rating is still increasing. However, if transforming the number to the percentage, we will have a different view, as reported in Fig. 5(b), where y-axis shows the percentage. The plot 'Good Rating' stands for the good rating from the angle of percentage. From this angle, we can find that, the good rating gets convergent very fast, and τ_r is obtained from this plot. Another plot of Fig. 5(b) 'Good Service' stands for the percentage of the good services the system gets at every moment. It can be observed that this plot is not convergent within 4500 steps, and the trend is increasing. τ_s is derived from this plot. Here we introduce two convergent values $\overline{V}_R$ and $\overline{V}_S$, which are defined as the average percentage of good ratings (R) and good services (S) after τ_r. In the baseline $\overline{V}_R$ can reach a very high value 0.980, while τ_s doesn't exist. We calculate $\overline{V}_S$ within final 300 steps. For baseline $\overline{V}_S = 0.75$, and it is expected that if the running time gets longer $\overline{V}_S$ will be larger, which shows that the model is relatively effective.

Fig. 5(c) is used to observe the variation of the neighbour list. In this figure, the y-axis is the number of cooperators with different types of qualities. There are three plots, 'Good', 'Dynamic', and 'Bad', in this figure, which denotes the number of good cooperators, dynamic cooperators, and bad cooperators respectively. The values of τ_n and ς can be derived from the plot "Good". In Fig. 5(c) the plot 'Good' can reach 300 after τ_n, which means in the baseline every peer can find a good cooperator. Note that after τ_n, the plot still has some fluctuations, which is not because the variation of the neighbour list, but because some cooperators are offline when the system makes statistics. So in the following experiments, when the selection hit ς is mentioned, it refers to the maximum number of the 'Good' plot after convergent point τ_n. With a good model, when the neighbour list gets stable, ς should be the same with the P_N, which means all peers find good peers as cooperators.

Finally, Fig. 5(d) reports the consensus statistics of system snapshot. There are three units in the figure, and each unit consists of two bars. The left bar reflects the recognition consensus, which includes five parts, # of good *SP*s,

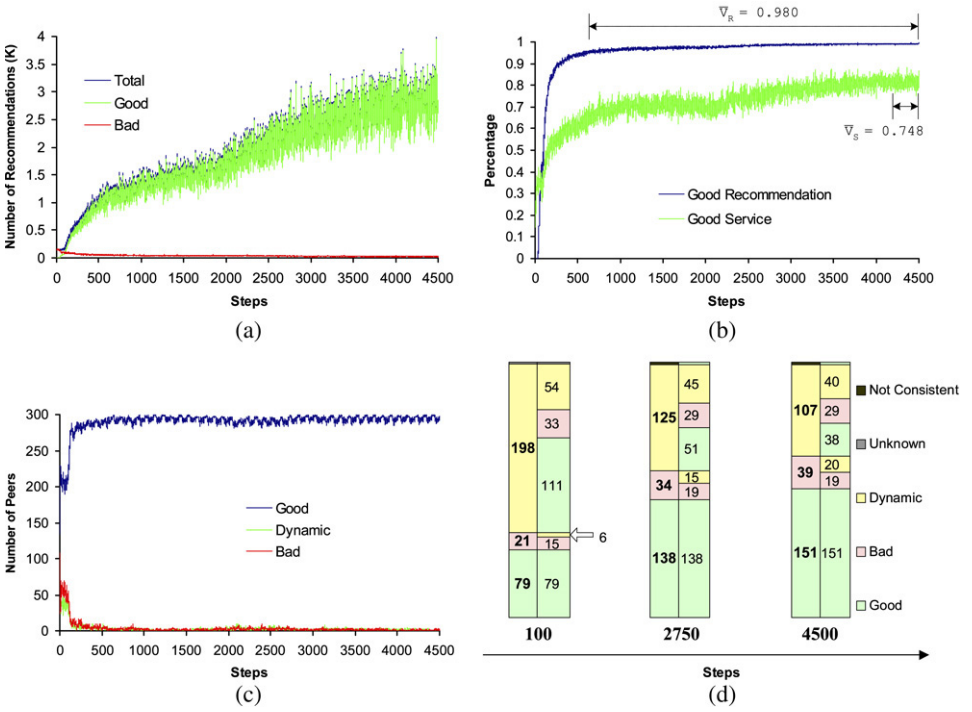


Fig. 5. Temporal results of the baseline: (a) correction rate, (b) percentage of good services and correct ratings, (c) statistics of the neighbour list, (d) consensus statistics of system snapshot.

# of bad *SP*s, # of dynamic *SP*s, # of unknown *SP*s, and # of not-consistent *SP*s, whose order is the same as the rightmost legends. These five parts are the current consensus status. For each part, the actual qualities of those *SP*s (i.e. the ground truth) are shown in the right bar within the same unit. For example, for the left most unit, the upper yellow part in the left part stands for at 100th simulation steps, there are 198 *SP*s are known as dynamic *SP*s, among which there are 54 *SP*s are actually real dynamic *SP*s, while 111 *SP*s are good *SP*s, and the remaining 33 *SP*s are bad *SP*s. In the figure, we can see that the numbers of *SP*s known as 'Not Consistent' and 'Unknown' are very few. The numbers of *SP*s with consensus status "Good", "Bad", and "Dynamic" occupy most portion. Within these three kinds of status, "Good" and "Bad" are actually the most important, because recognizing the good *SP*s and bad *SP*s is the major responsibility of the trust inference. Thus we will study the consensus progress base on ϱ_b and ϱ_g only. In Fig. 5(d), the three units are corresponding to three snapshots of the system. The first snapshot is at the 100th step, which stands for the initial stage of the system. The second snapshot is at the 2750th step, which represents the middle stage of the system, and the final one is at the 4500th step, which denotes the final stage of the system. From this figure, we can see that the number of good and bad *SP*s known by the whole system is increasing, and the *SP*s known as dynamic *SP*s are decreasing, which shows that the trust inference model do have an effect to decrease the disorder of the system, and achieving the goal to find out the good and bad *SP*s. From this figure, we also notice that the *SP*s known as good *SP*s are all actually good *SP*s, while for the known-as-bad *SP*s, some of the dynamic *SP*s are misclassified. However, this kind of misclassification is endurable.

It is worth noting that the above data also show the effectiveness of our simple trust inference model in the baseline. Next we will discuss according to the factors presented in Fig. 1.

6.3. Weight of rating

Changing the weight of rating is the direct way to change the role of ratings in the trust inference. However, it is not clear how the weight adjustment plays the effects. In Liang and Shi's paper [13], the authors provide some preliminary discussion on choosing the weight of the rating within their trust model. Basically [13] suggests that it is better to set a low weight to the rating, but it dose not dig inside. In this subsection, we delve into this issue in depth. Comparing with

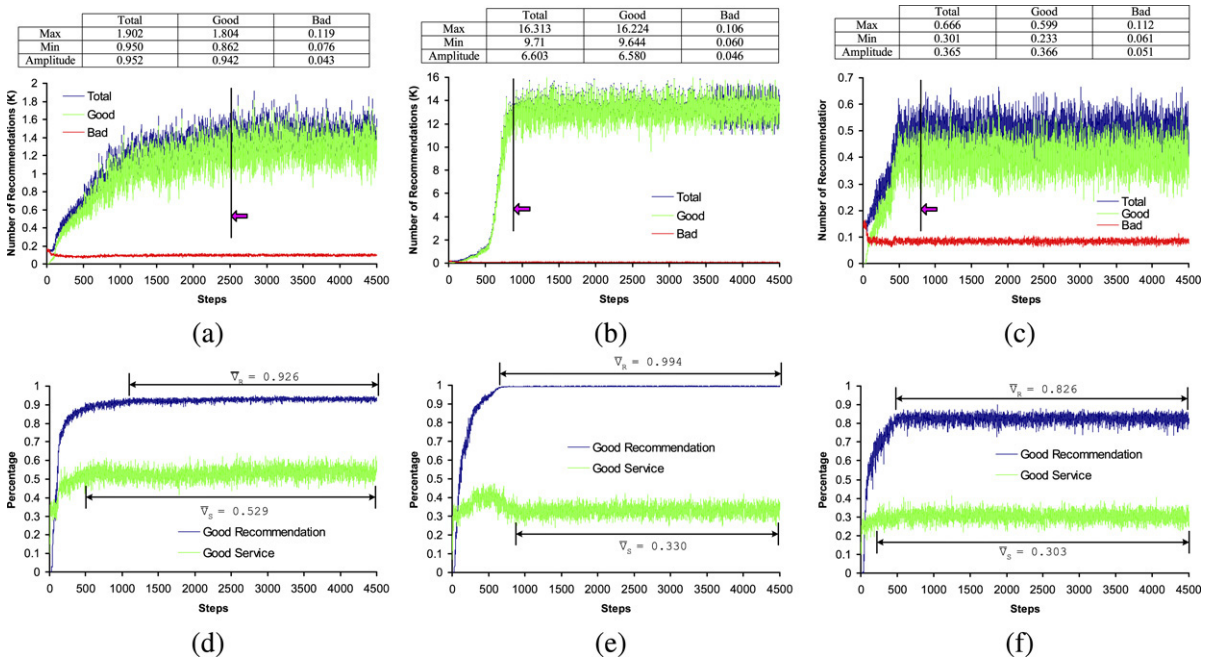


Fig. 6. Effects of different weights: correction rate with (a) $W = 0.5$, (b) $W = 0.7$, (c) $W = 0.9$, percentage of good services and correct ratings with (d) $W = 0.5$, (e) $W = 0.7$, (f) $W = 0.9$.

baseline, we change the weight to 0.5, 0.7, 0.9 separately to see the result from the prospective of correction rate and the percentage of good ratings and good services. The part of "**Weight of Rating**" corresponding to row "Section 6.3" in Fig. 4 shows the *final* results. Comparing these results with the results of baseline, we can derive some interesting observations: (1) Basically, the larger the value of W, the less the benefit speedup of the rating brought to the system; (2) Larger W is helpful to shorten the convergent time except τ_n, which shows a reverse variation tendency. The exception of τ_n tells us a fact that high W will interfere with the convergence on the recognition of neighbours, which is also implied by the decreasing of ς; (3) From the perspective of workload, with $W = 0.7$, the workload is ten times as others. Note that, with $W = 0.9$, the workload ω decreases a lot and the efficiency factor ε improves a lot. This is because the extremely high weight of rating plays the major role in the trustworthiness evaluation. However, this kind of efficiency is not good because it leads the system to a stable stage where the facts are distorted. So here we need to make a tradeoff between the efficiency and the correctness.

Fig. 6 depicts the comparison of two temporal results. In subfigure (a), (b), and (c) which present the result of the correction rate, there is a small table on top of each figure to show the fluctuation of the plot after τ_r (indicated by the vertical line in these figures). The column "Total", "Good", and "Bad" are corresponding to plot "Total", "Good", and "Bad". The row of "Max", "Min", and "Amplitude" lists the maximum value, the minimum value and the amplitude (equals to Max – Min) of each plot after τ_r. From Fig. 6(a), (b), and (c), we can again see that with higher weight on the rating, the system is faster to get convergent. From Fig. 6(d), (e), and (f), it can be seen that $\overline{V}_S$ is decreasing as W increases, which again shows that large W is harmful to bring more good services. From the above analysis, we get the following observation: *in a system with bad SPs, even when all raters are honest, it is better to set a low weight to the rating. The experiment results in our previous work* [13] *show that* $0.2 \sim 0.3$ *is a reasonable range for* W.

6.4. System scale

In order to study the scalability of the system, a larger scale system with 600 *SP*s is simulated.[2] Fig. 7 report the *temporal* results, and the corresponding *final* results are listed in the part of "**System Scale**" in the row "Section 6.4"

[2] The authors realize that 600 is not really large. However, the point here is to evaluate the trend of scalability and have some preliminary results on the scalability.

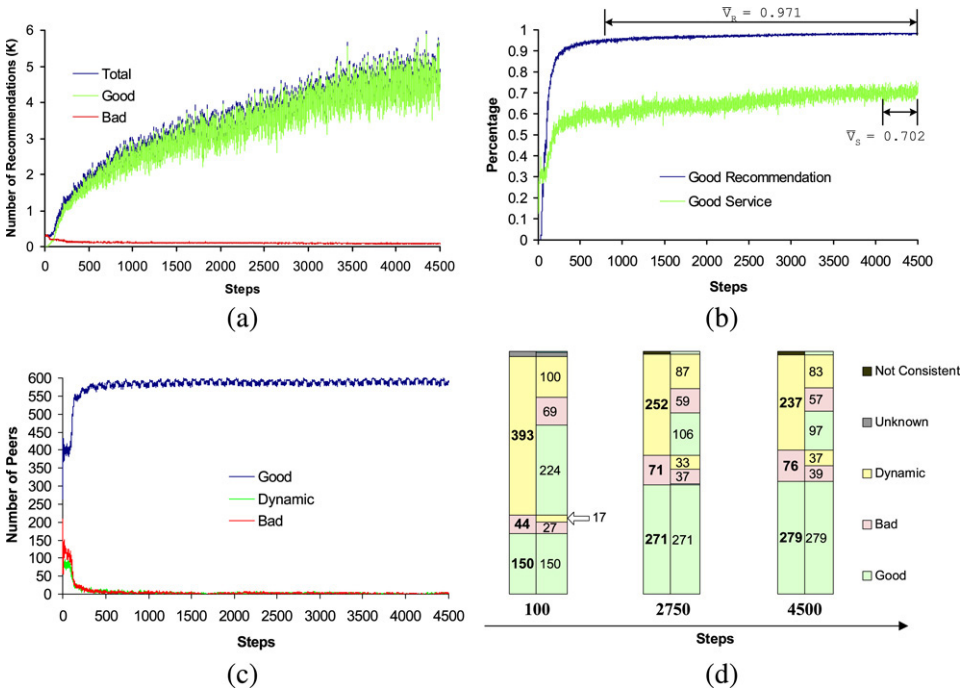


Fig. 7. Effects of large scale: (a) correction rate, (b) percentage of good services and correct ratings, (c) statistics of the neighbour list, (d) consensus statistics of system snapshot.

in Fig. 4. Though the scale is two times as the baseline, the workload is just about 50% higher. This is implied from both the total number of ratings in Fig. 7(a) and the value of workload ω. However, the performance of the system is almost the same: $\overline{V}_R$ is 0.97, 0.01 less than that of the baseline, and $\overline{V}_S$ is 0.70, 0.05 less than that of the baseline; when the neighbour list get stable, 598 (99.7%) *SP*s can get a good peer as the cooperator; 279 (72.7%) good *SP*s and 39 (40.6%) bad *SP*s are recognized, almost the same as the baseline (78.6% good *SP*s and 39.6% bad peers are recognized); the convergent times τ_C and τ_R, and benefit speedup β are all close to the baseline. Moreover, the efficient factor ε reaches to 15.81, almost two times as that of the baseline. All this data tells us that, *our simple trust model using the 'Push' dissemination approach is scalable*.

6.5. Storage capacity

Intuitively, the more storage space to hold the self-knowledge, the higher possibility to choose a good cooperator, which is helpful to improve the performance. However, in the real deployment, less space requirement is always desirable, especially for some systems like sensor networks, where the storage is very limited. So a tradeoff exists between the knowledge capacity and the space consumption. There are two kinds of capacity in the model: *Size of the neighbour list* S_N and *Size of the rating queue* S_Q. In order to study the effect of S_N and S_Q we conduct two experiments with the same configuration as the baseline, except enlarging S_N from 8 to 14, and S_Q from 5 to 8. Finally, in order to better evaluate the effect the S_N, we increase the scale of the system from 300 to 600 (but same percentage of different kinds of *SP*s as baseline, and enlarge S_N to a larger value 20. The *temporal* results are described in Fig. 8, while the *final* results are listed in the part of "**Storage Capacity**" corresponding to row "Section 6.5" in Fig. 4.

First, let us look at the case with $S_Q = 8$. From the *final* results in Fig. 4, we can see that except from the workload which decreases about 30%, the performance upgrade of other metrics are very limited; even worse, the performance of some metrics degrades, which shows that changing S_Q can not bring significant benefit to the system. Now let us turn to the case with $S_N = 14$. Except ϱ_g drops a little bit, the performance of other metrics are improved significantly. The reason of the drop of ϱ_g is, when S_N increases the possibility of the consensus conflict increases as well. For example, a good *SP* i enters the neighbour list of peer j but never gets chances to be selected as the

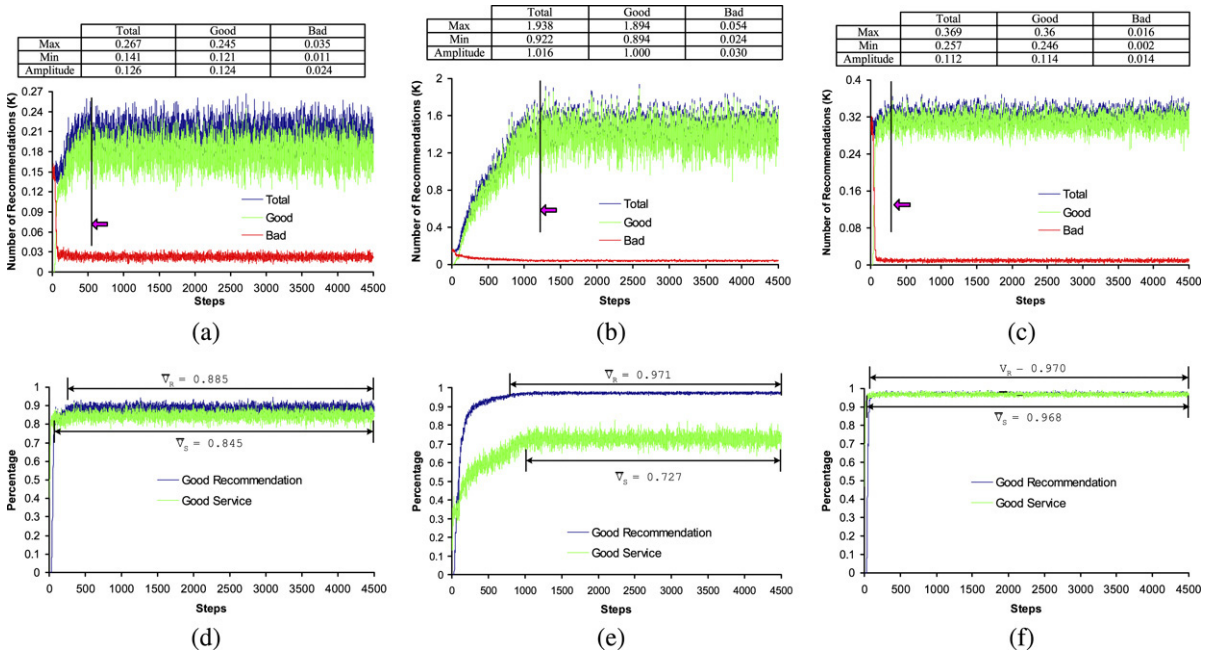


Fig. 8. Effects of different sizes of neighbour list and rating queue: correction rate with (a) $S_N = 14$, (b) $S_Q = 8$, (c) $S_N = 20, N_P = 600$, percentage of good services and correct ratings with (d) $S_N = 14$, (e) $S_Q = 8$, (f) $S_N = 20, N_P = 600$.

cooperator, because j's original neighbours are all good *SP*s and have accumulated high trustworthiness value. Since i can not be the cooperator, there is no chance for i to increase its trustworthiness value, thus though i always stays in the neighbour list, it is never selected. However, the drop of ϱ_g doesn't weaken the service benefit of system, on the contrary β increase from 1.02 in the baseline to 1.22. For the final case, though the scale is doubled, all the metrics are still get better performance than the baseline. From the convergent time and service benefit, we can see that it is even better than the case with $S_N = 14$ while the scale is just 300. If we compare this with the results of "Section 6.4" in Fig. 4, the simulation with the same scale 600 peers, we can see more significant improvements. In Fig. 8, we can see that increasing S_N is also helpful to make the percentage of good ratings and good services less fluctuated. From above all, we draw the following conclusion: *increasing the size of the neighbour list is more valuable than increasing of the size of the rating queue.* It again shows that the self-knowledge is more important than the knowledge from others (ratings) in open environments.

6.6. Quality of raters

In this subsection, we study how the quality of raters affects the system performance from two perspectives: *ratio of honest raters* and *extreme cases*.

Ratio of honest raters. In the baseline, all raters are honest raters. However, in a real system, such an ideal situation doesn't exist. So in this experiment we are going to study how the percentage of the honest raters affects the system. Two cases are studied: (1) only 20% of peers are honest raters, to simulate a relative unreliable system with large number of bad raters, and (2) 70% peers are honest raters, to simulate a relatively reliable system with small number of bad raters. The "temporal" results are illustrated in Fig. 9, and the "final" results are list in the part of "**Quality of Raters**" corresponding to row "Section 6.6" in Fig. 4. Comparing with the "final" results of the baseline, we can see that, when P_H drops from 100% in baseline to 70%, the selection hit (ς) drops to 193 (decrease 35.7%), and the benefit speedup ε drops to 0.69 (decrease 32.9%); while P_H drops more to 20%, these two values are 140 (decrease 53.3%) and 0.41 (decrease 60%) respectively. From Fig. 9, we can see other negative effects as well. Dishonest peers introduce significant noise and fluctuation. The noise and fluctuation in the case with $P_H = 70\%$ are already not acceptable, and the case with $P_H = 20\%$ is even worse. This is somehow consistent with what Whitby et al. [24] observed: with 30% of raters rating unfairly, the phenomenon of wrong judge of the *SP*'s reputations becomes severe,

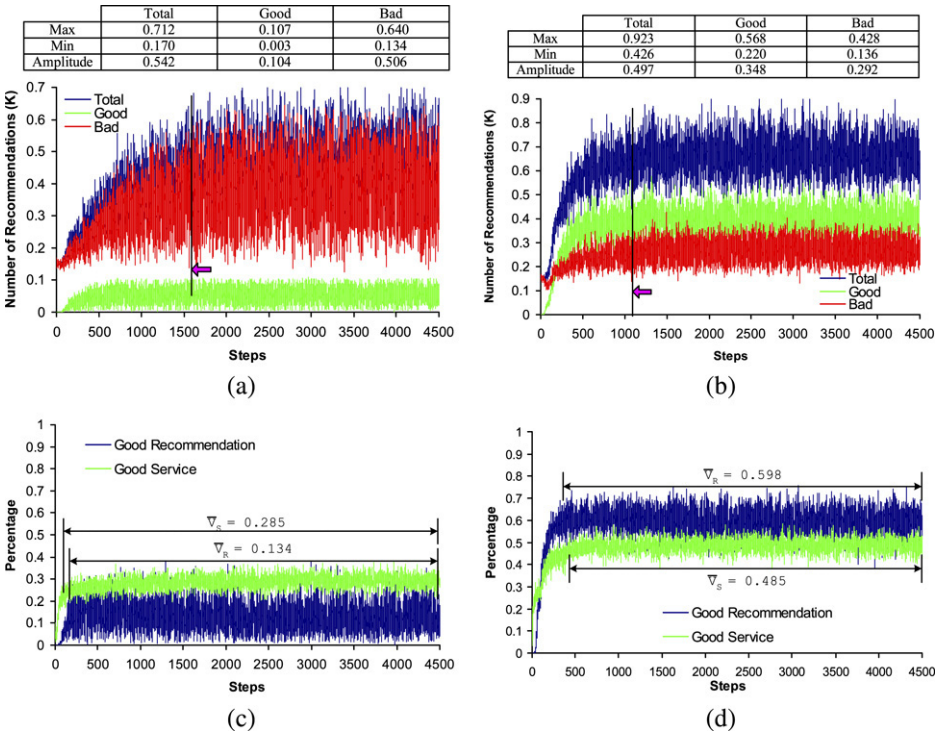


Fig. 9. Effects with different percentage of honest raters: Correction rate with (a) $P_H = 20\%$, (b) $P_H = 70\%$, percentage of good services and correct ratings with (c) $P_H = 20\%$, (d) $P_H = 70\%$.

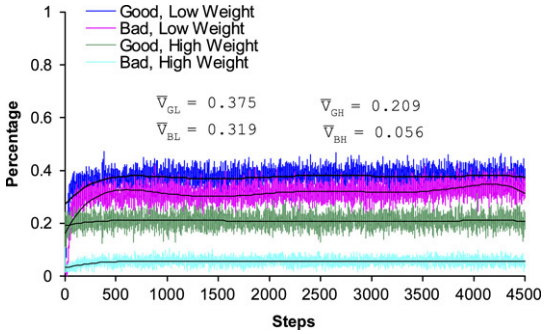


Fig. 10. Percentage of good services under the extreme case with different weights of ratings.

despite the bad rating filtering exists; with 40% of raters rating unfairly, iterating filtering finally breaks down. From this we can see that *the malicious rating is a big threat for the trust inference in open environments*. Even the ratings are good, it doesn't mean it will bring more reward for the system automatically, which can be seen in the following analysis. We believe that the better way to inhibit the negative effects of the bad rating is to lower the weight of the rating in the trust inference, rather than just focusing on filtering out the bad ratings, because focusing more on the self-experience information can improve the resistance to bad ratings.

Extreme cases. There are two intuitions most people will hold for the rating: (1) The performance of a system with all raters are bad should be worse than the system with all raters are good, no matter how to change the weight of the rating; (2) If all raters are good, it should set the weight of rating high to make full use of the rating. However, from the simulation results shown in Fig. 10, we find that none of these two intuitions hold. In Fig. 10, four plots with the order from top to bottom represent the percentage of good services the system gets with the option "good raters

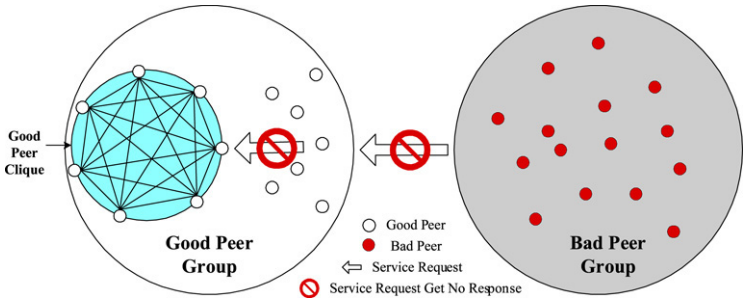


Fig. 11. An example of clique in the system.

with low weight ($W = 0.3$) of ratings", "bad raters with low weight ($W = 0.3$) of ratings", "good raters with high weight ($W = 0.7$) of ratings", and "bad raters with high weight ($W = 0.7$) of ratings" respectively. Note that here raters are not honest anymore, but either bad or good (see Section 4 for definition). A very amazing result from this figure is, even when all ratings are good (absolutely correct), $\overline{V}_S$ is just 37.5%, far less than 74.8% in the baseline. When the weight of ratings increases, the value drops to 20.9%. The possible culprits are the limited storage capacity of the peer and the fast spread of the ratings. The frequency of rating is expected to be higher than that of the direct interaction, so once the rating is assigned with a high weight, it tends to dominate the trustworthiness value update. Especially when ratings keep to boost a small portion of peers. Good ratings make the system get convergent very fast, which can be seen from the Fig. 10. Then some good peers form a close clique, inside which every member helps each other and boosts each other, but rejects any request from outside. Other good peers and bad peers, unfortunately, still choose peers in this clique as the cooperator, with the risk of their requests being rejected. However, once they drop the trustworthiness value of clique peers, ratings from the clique boost the trustworthiness values of clique members again. So the final results is only peers in the clique can get good services, as shown in Fig. 11, and other peers keep trying to request services from the clique even their requests are rejected again and again. When the weight of the rating is decreased, it can slow down the speed to form the clique, so does the enlarging speed of the clique. This is why even when all ratings are bad (absolutely wrong), when assigning the rating a low weight, it still outperforms the case with good ratings having a high weight. From this we can see that *even correct ratings sometimes play negative roles*.

6.7. Quality of SPs

In this section, we are going to study the results when the percentage of bad *SP*s and dynamic *SP*s are increased.

Percentage of bad SPs. The part of "**Quality of *SP*s**" corresponding to row "Section 6.7" in Fig. 4 shows the results when increasing P_B from 20% to 50% within 3000 steps (not 4500 steps as the baseline). We treat the result of "$P_B = 20\%$" in this section as 3000-step baseline. Comparing with the workload of 3000-step baseline, when P_B is 50%, the number of ratings increases significantly: from 4.4×10^6 to 14.8×10^6 (increase 236%), and the efficiency factor of the rating is only 53.4% of the 3000-step baseline. This data shows that, our model is sensitive to bad *SP*s, so that *more ratings are needed to help others to discover bad SPs*. The reason is, when the number of bad *SP*s increases, the chance for a peer to meet a bad cooperator also increases, so that the rating spreading is triggered more frequently. Though ε is decreased, the benefit speedup β increase from 0.98 to 1.47, which validates that our model can highly resist against the bad *SP*s. Note that here the simulation only runs to 3000 steps instead of 4500 steps, because the high workload generated makes the simulation become very slow after 3000 steps, which implies the necessity of mechanisms to find out the bad *SP*s in a system with large amount of bad *SP*s.

Percentage of dynamic SPs and dynamic interval. Dynamic *SP*s and length of dynamic interval are also the possible factors for the rating. Because the correctness of the rating is time related, when the rating arrives, maybe the dynamic *SP* which is rated has changed its quality. This section studies the effect of the dynamics within our model, in which ratings are just simply averaged and assigned with a low weight in the trust inference. In order to show more about the difference, the portion of dynamic *SP*s increase from 20% to 40%. When the frequency of *SP*s changing their quality increases, it is expected the correction rate of the rating decreases. However, from the results in Fig. 12, we find

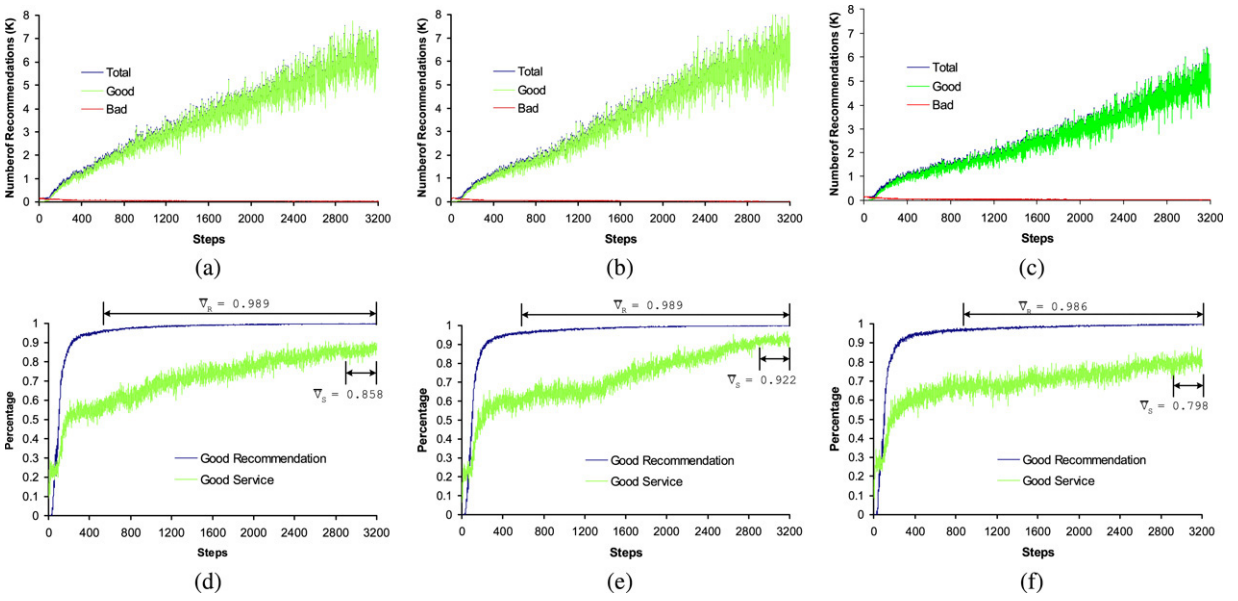


Fig. 12. Effects of different dynamic interval: correction rate with (a) $I = 3$, (b) $I = 12$, (c) $I = 30$, percentage of good services and correct ratings with (d) $I = 3$, (e) $I = 12$, (f) $I = 30$.

that *the negative effect brought by the shortening of the dynamic interval is not that obvious*. More surprisingly, the shorter the interval, the more percentage of the good ratings and services the system can get, with the cost of raising the workload of the system a little bit. We attribute this to the fact that the more frequently dynamic peers change their qualities, the higher possibility they will be caught for their bad services. In addition, our model relies more on the self-experience information, which offsets the negative effect of dynamics. To this end, we argue that our trust model is able to make use of the dynamics to catch the dynamic *SP*s while suppressing the negative effect. Many researchers [2,18,23,27–30] try to filter out bad ratings and make full use of strength of ratings. The basic idea of these work is introducing another model to associate the trustworthiness value (or weights) with ratings. We detain the comparison between these approaches and our simple average algorithm to the next section. Moreover, the fact that the change of the dynamic interval and percentage of dynamic *SP*s does not decrease the number of good services in our simple model implies that, *the simple trust inference model proposed in Section 3 is a good model in dynamic systems with high churn rates*.

6.8. Dissemination mode

So far all the simulations are based on the push model to disseminate ratings. The reason is, the push model is more efficient than the pull model, which is to be analysed in this subsection. From the results shown in the part of "**Dissemination Mode**" corresponding to row "Section 6.8" in Fig. 4 and the results in Fig. 13 we can see that the comprehensive performance of the pull model is close to that of the push model, except the overwhelming workload and the extremely low efficiency factor of the pull mode. Note that in the figure only 3200 steps are used. This is due to the large workload. Thus, we stop the simulation after we get all convergent times. The pull model incurs about 13-times of workload as that of the push model. The reason is, in the pull model, every time when a peer asks for the help from a new cooperator, the requested peer needs to pull the ratings about this cooperator and recalculate the cooperator's trustworthiness. While in the push model, only when one neighbour's trustworthiness value changes over a threshold, the ratings about this neighbour are pushed to others. So the frequency of ratings of the pull model is much more than that of the push model in a dynamic environment. From the table, we can see that, though the pull model can reach the convergent state earlier, the percentage of the good rating and the good service are worse than those of the push model. Therefore, we believe that *the push model is much better than the pull model for trust inference in the similar environments as we simulated*.

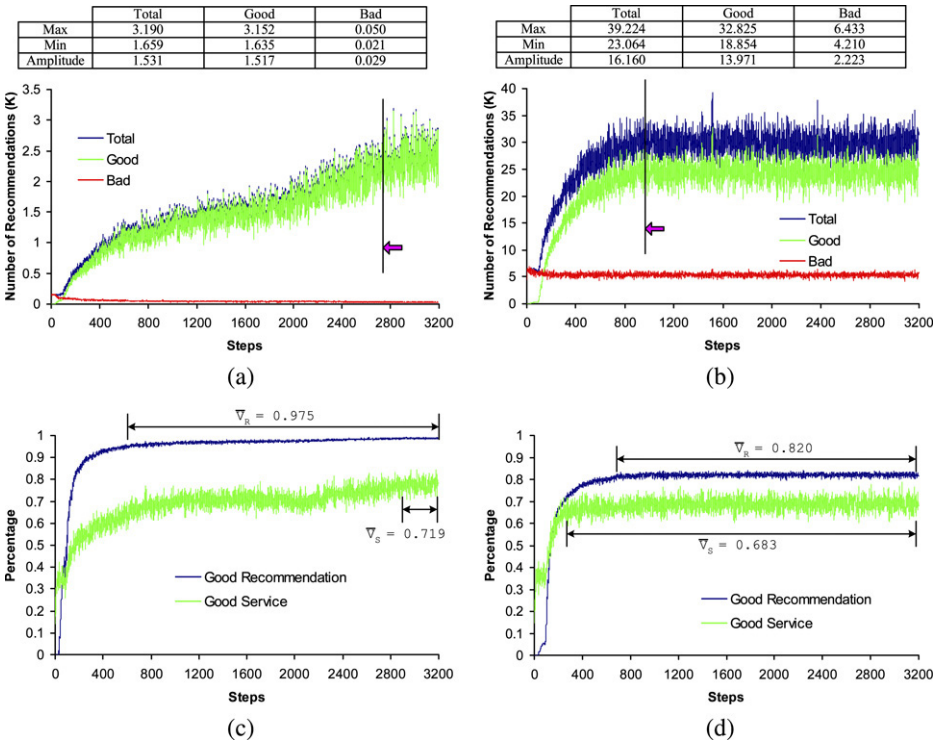


Fig. 13. Effects of pull model: correction rate with (a) $M = push$, (b) $M = pull$, percentage of good services and correct ratings with (c) $M = push$, (d) $M = pull$.

6.9. Validation of trustworthiness independence

Before we conclude this section, we want to answer the final question: *Does a good SP also have good rating attitude?* To our knowledge, there is no conclusion on this. For example, in human society, merchants tend to defame others to attract more customers, even those who are good *SP*s. In Section 3, we make the assumption that these two are independent. Here we want to illustrate what will happen if we relate the weight of the ratings with the trustworthiness value of the rater as a *SP* (i.e. the A2 algorithm in Section 4), as shown in the Fig. 14. Comparing with the baseline, we can see that except the convergence time is shortened and the workload of the system decreases a little bit, other metrics are even worse than the baseline. From Fig. 14(b), we can see that after τ_s, the average percentage of good service $\overline{V}_S$ is just 0.65, 12.7% less than that of the baseline ($\overline{V}_S = 0.75$). From Fig. 14(d), we can see that, the ability to recognize the good and bad *SP*s is decreased as the time goes on, and the number of unknown *SP*s increases. All these data show that the fast convergence resulting from relating ratings with the trustworthiness values of the rater as a *SP* just leads the system to a worse direction. So it is not a good idea to judge the ratings based on the rater's trustworthiness value, unless we can guarantee that the peer has the same quality as a *SP*s and as a rater. Unfortunately it is not always true. To this end, we argue that *the trust-independent assumption in Section 3 is appropriate for open environments*.

7. Performance evaluation II: Algorithm uncertainty

We have conducted a comprehensive evaluation on multiple algorithm-independent factors based on the average aggregation algorithm. Now we are in a position to compare all state-of-the-art rating aggregation algorithms. Note that all these algorithms are proposed with the trust-independent assumption, that is the quality for a peer to be a *SP* is independent of the quality for a peer to be a rater. The key role of these algorithms is getting rid of bad raters and obtaining accurate ratings.

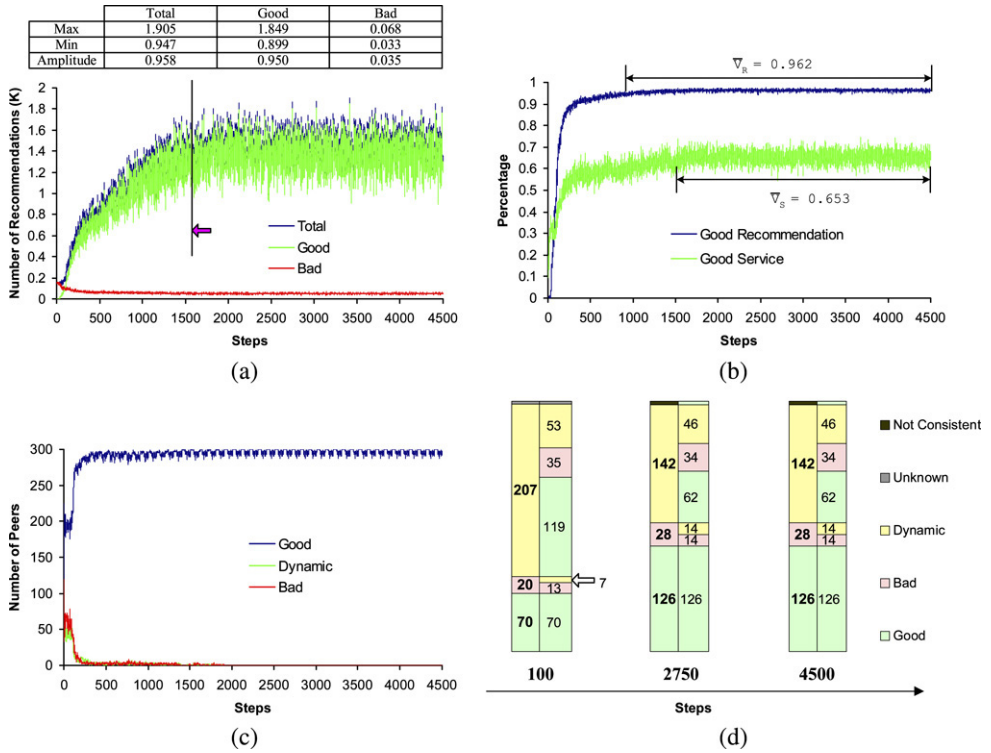


Fig. 14. The results of using the A2 algorithm: (a) correction rate, (b) percentage of good services and correct ratings, (c) statistics of the neighbour list, (d) breakdown of system discovery.

Fig. 15. A snapshot of the GUI interface of the simulation platform.

7.1. Extension

In the previous section, the simulation platform is built based on *Avg*. In order to integrate five algorithms listed in Section 4.5 into one common platform, several extensions need to be made. Fig. 16 shows the execution skeleton of service requesters and *SP*s for the extended simulation, and Fig. 15 illustrates a snapshot of the simulation platform. The major difference between the extension and the previous one is how to derive rating R. The update of R is

EXECUTION SKELETON OF REQUESTER:

```
while TRUE do
  pick up one peer k with the highest T;
  send the service request to k;
  after transaction, adjust k's T, and update the history list;
  if k's T < Threshold then
    put k into the blacklist;
  end if
  if k's service is bad then
    request all recommenders rating k to send the new ratings;
    every t steps, update the weight of raters;
  end if
end while
```

EXECUTION SKELETON OF SERVICE PROVIDER:

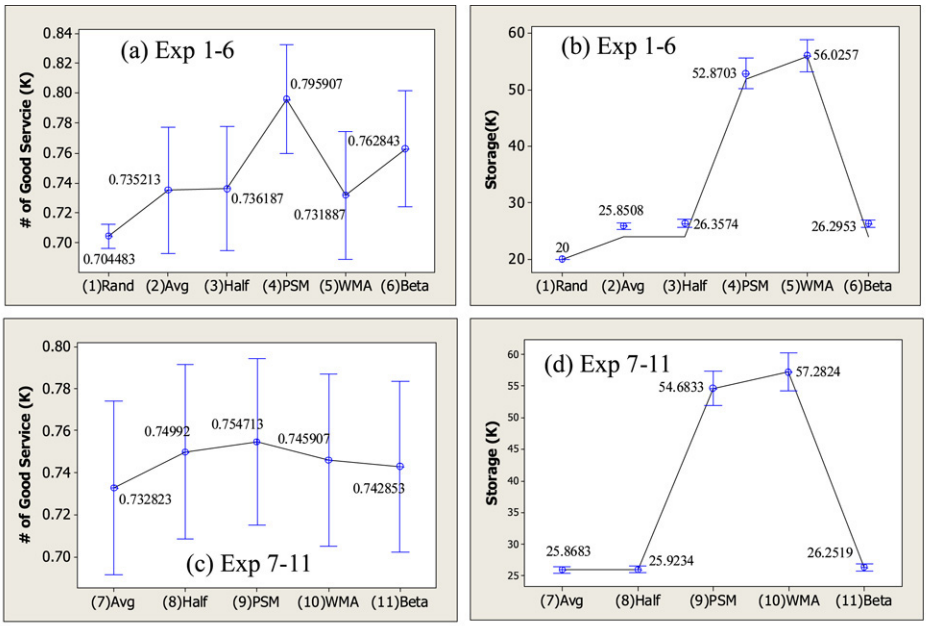
```
while TRUE do
  get the service request from i;
  if i is in the blacklist then
    Reject i's request;
  else
    provide the service to i;
    if neighbor list is not full then
      add i to the neighbor list;
    end if
  end if
end while
```

Fig. 16. The skeleton of system execution in evaluation II.

implemented by different aggregation algorithms now. For algorithms *Avg, Half*, and *Beta*, the possible raters can only be chosen from the neighbour list, i.e. only the peers with which the requester has interacted before can become a candidate. However, with the same limitation, the algorithms PSM and WMA will be restricted so that they can not show their special advantages. So for these two algorithms, we enlarge the rater candidate set to the two-hop neighbours, that is, neighbour's neighbour. Because of this, we can see that these two algorithms consume more storage in the analysis. In our evaluation, we try our best to keep our implementations in their original flavour. However, since these algorithms are proposed in different contexts, some small adjustments are needed when we integrate all of them into the same platform. It is worth nothing that, peers only request raters to update their ratings when peers get one bad service in the simulation. Besides getting one bad service (reactively), peers also update the weight of raters periodically. In the simulation, the program runs 2000 steps and then stops. The system has 300 peers.

We notice that peers can have more complex qualities than we have defined in Section 4.3. In this section, we redefine the type of qualities as both *SP*s and raters to extend our evaluation. Three types of behaviour patterns of *SP*s are studied: fixed (F), random (R), and oscillating (O). The type of "F" includes the fixed good and fixed bad. With the option F, *SP*s won't change their qualities once the simulation starts. With the option R, *SP*s will change their qualities randomly. While with the option O, *SP*s will change their qualities in a fixed oscillation span (20 steps in our simulation). For the quality as a rater, three kinds of bad raters are introduced and studied: "malicious", "exaggeration", and "collusive". Suppose s represents the real opinion of the rater. Malicious raters always give the complementary opinion, that is, sending out $1 - s$ to others. The exaggerating raters will exaggerate their ratings by a exaggerating factor α, which is 0.5 in our simulation. For this type of raters, the rating $s + \alpha * (s - 0.5)$ will be sent out. For the final category, collusive raters will send out 1 for the peers in the collusive group, and 0 for the peers outside the group. In our simulation, the size of the collusive group is chosen as 20% (60 peers) of the total number of peers in the system.

We select six performance metrics from the whole set proposed in Section 5, including the system workload ω, the accumulative correction rate ϕ_A, the total number of services η, the percentage of good service among all services η_g/η, the storage cost ζ, and the running time Υ, to conduct a complete comparison of the performance of different



Exp #	ω	ϕ_A	η	η_g / η	ζ	ϒ
1 (Rand)	0	0	278.914	0.758	6000.000	16.093
2 (Avg)	1.264	0.172	280.048	0.788	7755.226	22.362
3 (Half)	0.170	0.541	280.220	0.788	7907.216	19.158
4 (PSM)	49.048	0.743	280.604	0.851	15861.090	36.903
5 (WMA)	81.639	0.768	280.770	0.782	16807.712	41.38
6 (Beta)	0.947	0.946	279.558	0.819	7888.604	19.818
7 (Avg)	2.836	0.718	279.073	0.788	7760.482	22.132
8 (Half)	0.415	0.670	280.239	0.803	7777.020	20.069
9 (PSM)	72.619	0.779	279.540	0.810	16404.986	40.709
10 (WMA)	90.912	0.648	279.769	0.800	17184.726	40.247
11 (Beta)	0.707	0.098	280.224	0.795	7875.568	19.829

Fig. 17. Baseline and the test of change of the percentage of honest raters: (a) and (b) (corresponding to Exp.#1 to Exp.#6) are the interval plots of baseline about η_g/η and the storage cost. In the baseline, only 20% peers are malicious raters. (c) and (d) (corresponding to Exp.#7 to Exp.#11) are the interval plots on the scenario with 80% malicious raters based on the baseline.

algorithms. Each figure includes a figure part and a table part. The figures are the interval plots on the data set corresponding to each peer. Two set of data are plotted. One is the number of good services η_g (the subfigures on the left side), the other is the accumulative storage cost (the sub-figures on the right side, with unit of "10^3"). Each figure contains five to six vertical lines, which are corresponding to different algorithms. The random algorithm (denoted as *Rand*) is to simulate the case where no trust information is used to help to choose the *SP*s. So in *Rand*, peers randomly choose the *SP*s, and do not store the rating from raters. The value shown in the middle of the line is the corresponding mean. All plots are with 95% confident interval. In the table part, all the data corresponding to the metrics mentioned above are listed, and this data is from the view of the whole system. Note that since a large number of experiments with different parameter configurations have been conducted in our analysis, we use experiment numbers to distinguish them. The configuration of parameters corresponding to each experiment is explained in the caption of the figure. The experiment numbers are listed in the x-axis of every figure beside the names of algorithms, and the first column in the table. For example, `Exp.#(6)` in Fig. 17 means the *Beta* aggregation algorithm with the configuration that 20% peers are malicious raters.

7.2. Baseline and effect of malicious raters

Normally, we would expect most *SP*s and raters in the system are good, so we simulate an healthy environment as our baseline, where 80% *SP*s are good and 80% raters are honest. The remaining 20% *SP*s are fixed malicious *SP*s, and

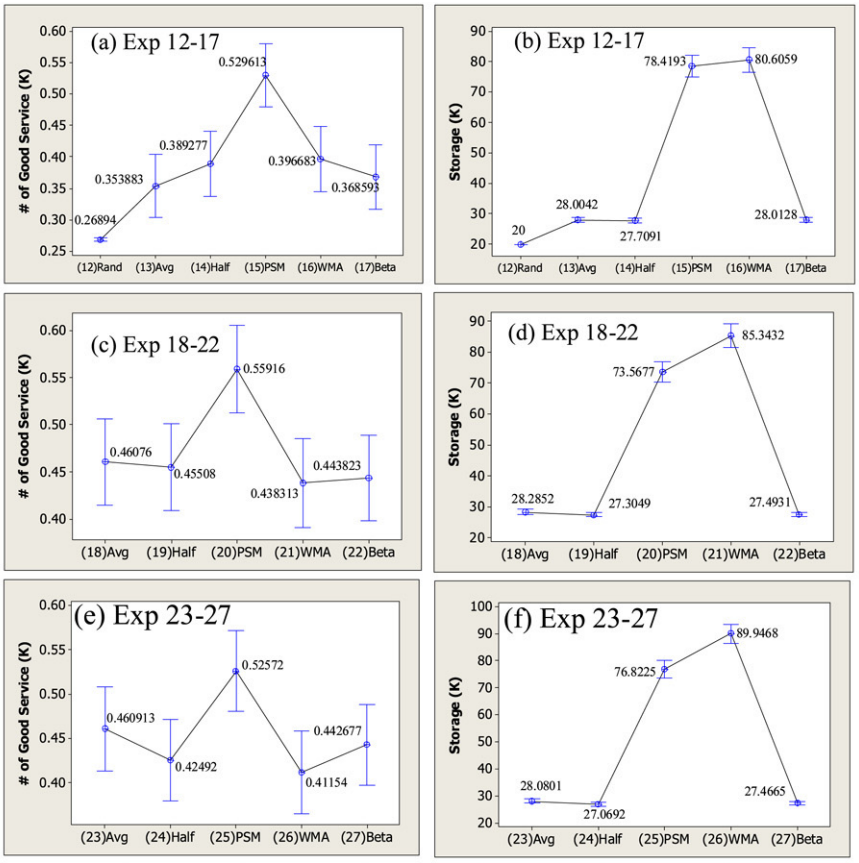
the remaining 20% raters are malicious. Since we want to study the effect of ratings in the trust inference, it is natural to set a high weight to the rating. In the baseline, the weight of rating W is set as 0.8, which is corresponding to figures (a) and (b), and Rows 1–6 of the table in Fig. 17. From Fig. 17(a) and column "η_g/η" in the table, we can see that, from the viewpoint of η_g/η, PSM outperforms other algorithms obviously. Though *Beta* is a little bit worse than PSM, from Fig. 17(b) and ζ, we can find that, the storage cost of *Beta* is just the half of that of PSM. The running time of *Beta* (19.818) is also far less than that of PSM (36.903). Despite WMA with the highest storage cost, the performance is not as good as PSM. It is even a little bit worse than *Avg* and *Half*. Because of the highest communication and storage cost, WMA is also the slowest algorithm. Comparing with *Rand*, all algorithms get considerable improvement, but at the cost of more storage and communication. Figs. (c) and (d), and the rows 7–11 of the table in Fig. 17 show the results when the percentage of malicious raters increases from 20% to 80%. Comparing these two scenarios, we can find that, η_g/η dose not decrease significantly. Among all algorithms, PSM drops most heavily, from 0.795 in subfigure (a) to 0.755 in subfigure (b) for the mean number of good services among all peers, and from 0.851 to 0.810 for η_g/η from the system view. Considering the significant change of the number of malicious raters, we can say that all algorithms still perform well. To this end, we argue that *when the environment is a healthy environment with a majority of good peers, all algorithms show quite a bit resistance against malicious raters*. From the view of the storage cost, only PSM and WMA increase a little bit, while other algorithms keep almost the same. The reason is, as explained in the Section 7.1, PSM and WMA will accept the raters from two-hops neighbours, thus more raters will be accepted and more ratings will be generated, which leads to more dependence on ratings for PSM and WMA. When the percentage of malicious raters increases, more ratings are needed to judge the quality of the rater. For the other three algorithms, the number of ratings is far fewer than those of the previous two algorithms, thus the change of the storage cost is not obvious.

7.3. Effect of bad SPs

To study the resistance against the bad *SP*s, 80% bad peers are contained in this group of simulations, so that we can see the obvious effect brought by bad *SP*s. From Fig. 18, we can see several similar results as the baseline: the costs of PSM and WMA are much higher than those of other three algorithms in terms of the storage cost and the running time, and PSM is still the best algorithm to get the largest η_g/η. However, comparing to the baseline, the significant increase of the percentage of bad *SP*s incurs the significant increase of the communication and storage for PSM and WMA, and sharply drop of η_g/η. When the quality of bad *SP*s are random or oscillating, η_g/η is basically higher than the case with fixed malicious quality. One amazing result is, in the case of *SP*s with random or oscillating quality, *Avg* is the second best algorithm to get the good services. Since the design of *Avg* is much simpler, and the cost of this algorithm is lower, it is worth considering *Avg* when using an open system where most peers are oscillating *SP*s. We can also observe that the performance of the other simple algorithm *Beta* is close to the performance of *Avg*. Their performance is better than the more complex algorithms, such as *Half* and WMA.

7.4. Effect of different types of bad raters

Next we compare the algorithms with respect to their resistance to the three kinds of bad raters, i.e. malicious raters, exaggerating raters and collusive raters. In this group of experiments, we take the same configuration as the baseline except that the percentage of bad raters increases from 20% to 80%, and 20% fixed malicious *SP*s are changed to 40% oscillating *SP*s. Comparing Figs. 17(a) and 19(a), we can see that when the percentage of malicious raters increases significantly, the number of good services obtained by the *Avg* algorithm drops the least among all algorithms: from 0.735 to 0.666 (0.069), which is only about 9.3%. It shows that *Avg* is the strongest to resist the malicious raters. PSM also plays well when facing the bad raters. In Fig. 19(c) when 80% raters are exaggerating, WMA shows a considerable weakness than other algorithms comparing to the previous cases, while in Fig. 19(e) where 80% raters are collusive, WMA performs closely to PSM and is better than other four algorithms; for *Half*, the situation is opposite to WMA. This implies that WMA highly resists collusive raters and weakly resists exaggerating raters, while *Half* can highly resist exaggerating raters, and weakly resist collusive ratings. Another interesting fact we can see from Fig. 19(a), (c) and (e) is, no matter facing what kind of bad raters, *Avg* performs similarly and is quite stable. This shows that *Avg* algorithm is robust against all the three kinds of bad raters. Fig. 19(b), (d), and (f), and column "ω" and column "ζ" of the table in Fig. 19 show the same result with all previous cases about the storage cost: WMA is with the highest communication and storage cost, and the cost of WMA and PSM are much higher than other cases from both views of the communication and storage.

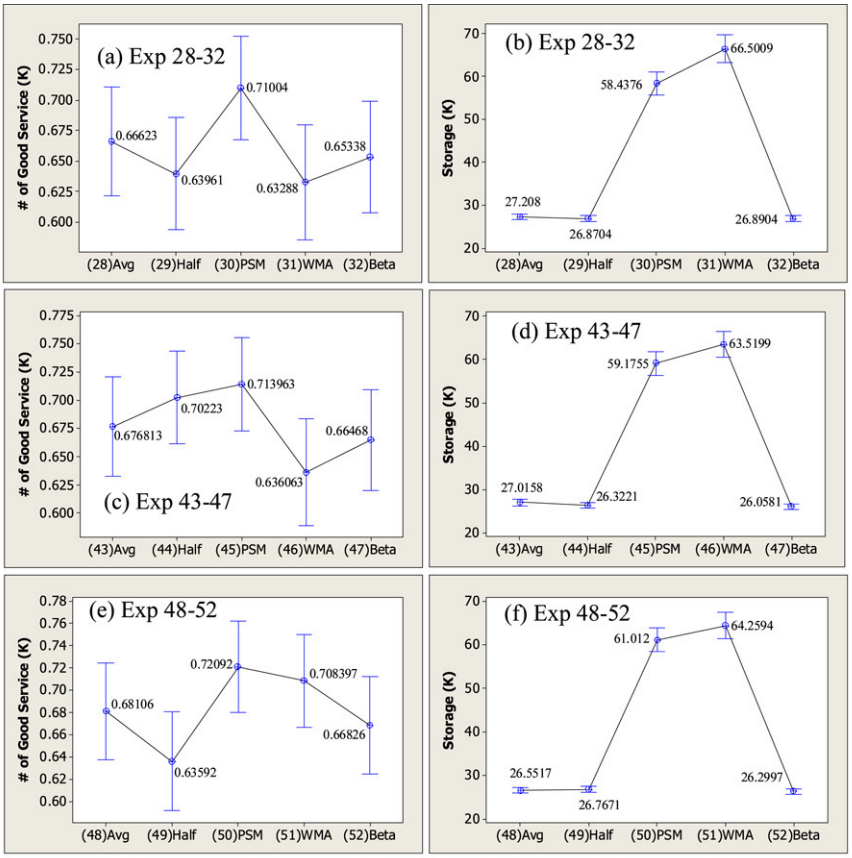


Exp #	ω	ϕ_A	η	η_g / η	ζ	Υ
12 (Rand)	0	0	280.130	0.288	6000.000	9.464
13 (Avg)	1.178	0.080	280.661	0.378	8401.274	22.302
14 (Half)	3.250	0.034	280.039	0.417	8312.734	24.465
15 (PSM)	243.649	0.326	280.088	0.567	23525.796	59.346
16 (WMA)	287.646	0.497	279.814	0.425	24181.782	77.421
17 (Beta)	3.103	0.035	279.772	0.395	8403.846	23.203
18 (Avg)	3.054	0.531	280.540	0.493	8485.548	33.078
19 (Half)	2.481	0.504	280.171	0.487	8191.470	23.034
20 (PSM)	148.413	0.644	279.263	0.601	22070.306	57.002
21 (WMA)	285.491	0.566	279.779	0.470	25602.964	73.976
22 (Beta)	1.111	0.512	279.468	0.476	8247.928	22.392
23 (Avg)	3.710	0.629	279.931	0.494	8424.042	23.294
24 (Half)	1.292	0.540	279.405	0.456	8120.748	22.643
25 (PSM)	191.658	0.660	279.383	0.565	23046.754	56.251
26 (WMA)	302.745	0.518	279.512	0.442	26984.046	78.112
27 (Beta)	1.149	0.521	278.889	0.476	8239.952	24.135

Fig. 18. Test of the quality change of *SP*s: (a) and (b) are the interval plots of the scenario where there are 70% fixed malicious *SP*s. (c) and (d) are the results about the scenario where 70% random *SP*s. (e) and (f) are the results about the scenario with 70% oscillating *SP*s.

7.5. *Effect of oscillating SPs*

One of the most challenging issues in open environments is to detect and handle dynamic behaviour, which attracts a lot of attention by researchers. In this group of experiments we intend to study the effect of dynamics. Comparing to the baseline, Fig. 19 increases the percentage of oscillating *SP*s from 30% to 70%. For better comparison we again include *Rand*. The result shows that, except PSM, all other four algorithms get lower η_g/η than *Rand*.



Exp #	ω	ϕ_A	η	η_g / η	ζ	Υ
28 (Avg)	3.128	0.728	279.965	0.714	8162.394	25.026
29 (Half)	2.960	0.707	279.530	0.686	8061.128	25.056
30 (PSM)	81.060	0.806	280.012	0.761	17531.268	51.835
31 (WMA)	168.392	0.668	280.109	0.678	19950.274	66.295
32 (Beta)	0.928	0.931	279.389	0.702	8067.122	22.131
43 (Avg)	2.273	0.956	281.109	0.722	8104.726	23.744
44 (Half)	1.928	0.963	279.799	0.753	7896.626	21.451
45 (PSM)	81.416	0.842	279.578	0.766	17752.656	41.640
46 (WMA)	146.900	0.736	280.156	0.681	19055.976	48.830
47 (Beta)	0.191	0.560	280.106	0.712	7817.424	19.668
48 (Avg)	0.356	0.503	279.624	0.731	7965.508	22.202
49 (Half)	6.577	0.502	279.457	0.683	8030.142	20.209
50 (PSM)	82.381	0.672	281.032	0.770	18303.598	40.398
51 (WMA)	110.891	0.619	280.622	0.757	19277.826	39.907
52 (Beta)	1.943	0.776	279.939	0.716	7889.914	23.183

Fig. 19. Test of different types of bad raters. In all these three scenarios, there are 40% oscillation *SP*s: (a) and (b) are the interval plots of the scenario where there are 80% malicious raters, (c) and (d) are the results about the scenario with 80% exaggerating raters, while (e) and (f) are the results about the scenario with 80% collusive raters.

Even for PSM, the best algorithm we think until now, it can just get very limited improvement. Considering the cost of communication and storage, we can say that all the compared algorithms play a negative effect when the quality of most *SP*s are oscillating. Thus we conclude that *the system dynamics is a big threat and challenge for the trust inference in open environments, and more intelligent and agile detection mechanisms are needed for future research.*

7.6. Effect of dynamic rating weight

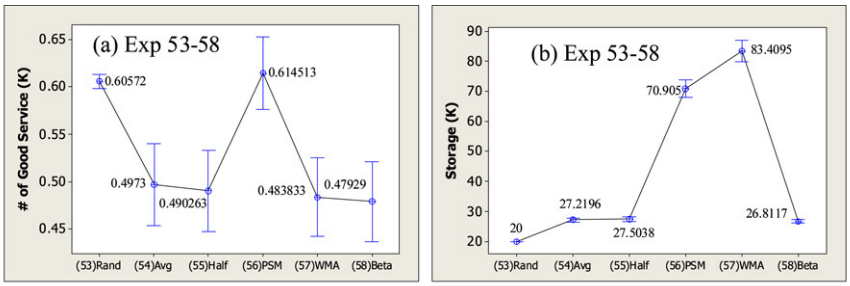
In Section 7.3, we have known that increasing the percentage of bad *SP*s quite a bit can drop η_g/η significantly. In this group of experiments, we use the same percentage, 70% oscillating *SP*s, as the case in Fig. 18(e). We also increase the percentage of malicious raters from 20% to 80%. If we keep the same weight of rating, that is $W = 0.8$, it can be imagined that the performance of the new scenario must be worse than the case in Fig. 18(e), for there are much more malicious raters. Here we decrease the weight of the rating from 0.8 to 0.2. From Fig. 21(a) and column "ζ" in the table, we find that η_g/η does not drop, but increases significantly instead. *Avg* and *Beta* even beat PSM considerably. However, from column "ζ" and column "Υ", we can see that the corresponding storage cost of *Avg* and *Beta* also increase significantly. The running time of these two algorithms is increasing as well, comparing with Fig. 18(e). Thus, when the weight of rating is set to be low and the environment is severe with many bad *SP*s and raters, the simple algorithm *Avg* and *Beta* can get same level or even better performance than the complex algorithm like PSM. The reason is in this situation, relying more on the self-experience which is more reliable than the rating, is more helpful to understand the surrounding environment.

8. Conclusions

In this paper, we systematically evaluate the effect of ratings on trust inference in open environments, in the context of a simple distributed trust inference model. Based on the simulation results, we find that:

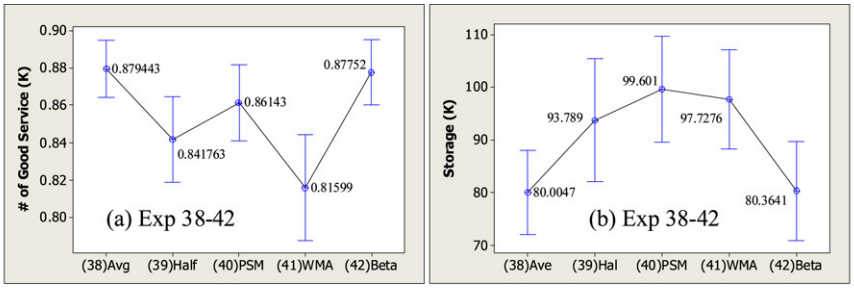
- *Design cost vs. performance*: In most cases, the most complex algorithm PSM outperforms other algorithms. But the other complex algorithm WMA has poor performance. Both algorithms are seeking a way to weight the ratings, but their performances have significant difference, which implies that using complex algorithms to rate ratings is not always as expected to have good performance. Furthermore, for the PSM design that replies on the common set to weight ratings, it is very difficult to decide the common set in a real system, such as P2P systems. Even if the common set can be found (through broadcast), the cost will be tremendous. So the cost of PSM design might lead to its unusability in a real system, even though it has the best performance. Regarding the low implementation cost and simple algorithms *Avg* and *Beta*, though their performances are not as good as PSM, both these two algorithms still have good performance, especially when there are a lot of bad *SP*s in the system, as shown in Sections 7.5 and 7.6. So if we aim to develop a general trust inference model, considering both the implementation cost and performance, the simple algorithms like *Avg* and *Beta* may be the better choice. In some cases where the ratings take the only role for the trust inference, like eBay, the complex algorithm like PSM might have more potential advantages.
- *Vulnerability*: Intuitively, if only the ratings are considered, the simple algorithm like *Avg* may be the most vulnerable, and the complex algorithm like PSM may be more robust. However, it does not mean that simple algorithms are more vulnerable if we compare them in the context of a trust inference model. Actually, we believe all algorithms have similar vulnerability. In open environments, we can not assume the ratings are reliable. Even we don't count the malicious raters, the ratings from honest raters still can be wrong. For instance, from peer A's point of view, peer C is a good *SP*, but for peer B, C may be a bad *SP*. This certainly can happen in P2P systems. For example, peer C is close to A, but far away from B, so C can provide fast service to A, but high-delayed service to B. Since peers provide the rating based on their own knowledge, and each peer's knowledge is different, it is hard to tell the precision of the ratings. From this angle, no matter what rating algorithms are used, it cannot change the fact that ratings are unreliable, in other words, the vulnerability is unavoidable in all rating algorithms. For example, PSM needs to calculate the personalized similarity based on the common neighbour set. However, peers decide this set based on the claim of raters. In the simulation, we make this common set accurate by abstracting it from the global information. But in real P2P systems, malicious/honest raters can give malicious/wrong information to confuse the weight updating. So we cannot only rely on the ratings. When the system has a lot of diversities (different services, locations, and peer qualities), tuning rating model may not have enough capacity to rescue the system. In this case, we need to tune the trust model instead of the rating model to limit the negativeness of the ratings by putting more weight on the self-experience, as we have seen in the comparison of Figs. 18 and 20.

There is one more thing we need to discuss about WMA. WMA has the largest cost, but with the worst performance, which is surprising. The possible reasons are: (1) In Yu's approach [30], the weight of rating is dynamic, while in this paper, the weight is fixed. The reason we fix the weight of rating is to fit WMA into our general trust model. (2)



Exp #	ω	ϕ_A	η	η_g/η	ζ	ϒ
53 (Rand)	0.000	0.000	257.353	0.706	6000.000	8.081
54 (Avg)	2.204	0.901	256.782	0.581	8165.892	20.489
55 (Half)	0.657	0.598	257.099	0.572	8251.128	18.667
56 (PSM)	114.897	0.745	257.966	0.715	21271.490	46.416
57 (WMA)	239.884	0.543	257.365	0.564	25022.864	59.545
58 (Beta)	0.150	0.487	256.986	0.560	8043.514	17.976

Fig. 20. Test when there are 70% dynamic *SP*s: (a) and (b) are the interval plots of η_g/η and the storage cost.



Exp #	ω	ϕ_A	η	η_g / η	ζ	ϒ
38 (Avg)	0.525	0.486	278.047	0.949	24001.414	39.677
39 (Half)	0.550	0.518	271.053	0.932	28136.702	58.664
40 (PSM)	33.316	0.570	277.538	0.931	29880.310	47.689
41 (WMA)	90.162	0.645	278.668	0.878	29318.288	42.491
42 (Beta)	0.513	0.483	276.743	0.951	24109.230	40.889

Fig. 21. Test when the weight of the rating is lower in a quite severe environment: (a) and (b) are the interval plots of η_g/η and the storage cost.

Studying carefully on Yu's approach, we find that in Yu's approach the weight update of raters always makes the weight decrease. Some mechanisms need to be introduced to increase the weight as well, when the raters give correct ratings. These two might be the reasons to lead to the poor performance of WMA in our paper.

Based on the discussions and the simulation results, we have the following future directions for the rating model design:

- Simple rating algorithms like *Avg* can be a general rating model to be used in any trust model.
- When the target environment has a lot of dynamics and malicious peers, employing the simple rating algorithms like *Avg* and *Beta* is better than the complex rating algorithms like PSM considering the overall effects of both the performance and implementation cost.
- When the target environment has a lot of dynamics and malicious peers, enlarging the neighbour list and increasing the weight of self-experience is very helpful to boost the performance.
- In some storage restrained systems like the sensor networks, simple algorithms like *Avg* and *Beta* are suggested considering their simplicity and low storage cost.
- From the perspective of system workload, the *push* model is better than the *pull* model.

Finally, we want to take the challenge to answer the question of *what kind of rating model/trust model is good?* We need to answer this question in the context of trust model building. It is worth noting that the rating model is only part of the trust model. Because the self-experience, the other part of trustworthiness value, is derived from the direct transaction, whose cost is relative high and the accumulative time might be long, we integrate the rating into the trustworthiness derivation, by making use of the efficiency brought by ratings. But the ratings essentially are not reliable, and the capacity of rating in trust inference is limited, especially when the system has a lot of diversities. So we will say that a rating model is good only if the rating model can improve the performance of the trust model. So the question turns to, what is a good trust model. From our analyses we conjecture that a good trust model must highlight the *adaptiveness*, and it is better to be *simple*:

(1) *Weight adaptiveness*. Since the rating is a double sword, we need to make use of its advantage (efficiency) while inhibiting its disadvantages (wrong ratings). In a good trust model, we can reach this goal by introducing the weight adaptiveness. When we find the effects of ratings are not positive any more, we need to reduce the weight of the rating and at the same time increase the weight of self-experience in the trustworthiness derivation. In this way, the system can self-adjust to find out the best trade-off point to maintain the efficiency and the cost.
(2) *Neighbour list size adaptiveness*. We have known that increasing the size of neighbour list can improve the performance of the trust model at the cost of large storage consumption. A good trust model must be able to self-adjust the size of neighbour list according to the environment, so that when the environment is severe (with a lot of diversities), the neighbour list is enlarged to hold more cooperator candidates; when the environment is healthy, the neighbour list can be reduced to save the storage cost.
(3) *Environment awareness*. To employ the two kinds of adaptability described above, the trust model should dynamically sense the quality of environment, and build the adaptability based on the environmental quality.
(4) *Simplicity*. Given the complexity of open environments, it is definitely desirable that the good trust model is a simple model. We hope that the simple trust inference model with the simple *Avg* rating model can be good enough if the adaptability is adopted.

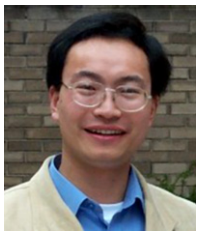
Designing such a good trust model is our future work, and it is expected to be a general-purpose trust model which can be applied to all kinds of open systems, raging from the storage-constrained systems like sensor networks to the storage rich systems like Collaborative High End computing, or from the dynamic systems like the P2P networks to the relatively static systems like ISP peering for Internet routing. Clearly, the work reported here has not exhausted the whole problem area, we hope this paper raises this issue to the community.

References

[1] K. Aberer, Z. Despotovic, Managing trust in a peer-to-peer inforamtion systems, in: Proceedings of the 10th International Conference on Information and Knowledge Management, CIKM'01, 2001.
[2] S. Buchegger, J.L. Boudec, A robust reputation system for p2p and mobile ad-hoc networks, in: Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems, 2004.
[3] F. Cornelli, E. Damiani, S.D.C. Vimercati, S. Paraboschi, P. Samarati, Choosing reputable servents in a p2p network, in: Proc. of the 11th International World Wide Web Conference, May 2002.
[4] Z. Despotovic, K. Aberer, P2p reputation management: Probabilistic estimation vs. social networks (Management in peer-to-peer systems: Trust, reputation and security), Computer Networks 50 (4) (2006) 485–500 (special issue).
[5] R. Dingledine, N. Mathewson, P. Syverson, Reputation in p2p anonymity systems, in: Proc. of the 1st Workshop on Economics of Peer-to-PeerSystems, June 2003.
[6] S. Ganeriwal, M. Srivastava, Reputation-based framework for high integrity sensor networks, in: Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks, Washington DC, USA, 2004.
[7] B. Gross, A. Acquisti, Balances of power on ebay: Peers or unequals, in: First Workshop on Economics of Peer-to-Peer Systems, June 2003.
[8] A. Jøsang, E. Gray, M. Kinateder, Analysing topologies of transitive trust, in: Proceedings of the Workshop of Formal Aspects of Security and Trust, FAST, Pisa, September 2003.
[9] A. Jøsang, R. Ismail, The beta reputation system, in: Proceedings of the 15th Bled Electronic Commerce Conference, June 2002.
[10] S. Kamvar, M.T. Schlosser, H. Gacia-Molina, The eigentrust algorithm for reputation management in p2p networks, in: Proc. of the 12th International World Wide Web Conference, May 2003.
[11] S. Lee, R. Sherwood, B. Bhattacharjee, Cooperative peer groups in nice, in: Proc. of IEEE Conference on Computer Communications, INFOCOM'03, March 2003.
[12] Z. Liang, W. Shi, Enforcing cooperative resource sharing in untrusted peer-to-peer environment (Non-cooperative wireless networking and computing), ACM Journal of Mobile Networks and Applications (MONET) 10 (6) (2005) 771–783 (special issue).

[13] Z. Liang, W. Shi, PET: A personalized trust model with reputation and risk evaluation for P2P resource sharing, in: HICSS-38, Hilton Waikoloa Village Big Island, Hawaii, January 2005.
[14] S. Marsh, Formalising trust as a computational concept, Ph.D. Thesis, University of Stirling, 1994.
[15] L. Mui, M. Mohtashemi, A. Halberstadt, A computational model of trust and reputation, in: Proceedings of the 35th Hawaii International Conference on System Sciences, 2002.
[16] P. Resnick, H.R. Varian, Recommender systems, Communications of the ACM 40 (3) (1997) 56–58.
[17] P. Resnick, R. Zeckhauser, E. Friedman, K. Kuwabara, Reputation systems, Communications of the ACM 43 (12) (2001) 45–48.
[18] J. Sabater, C. Sierra, Regret: Reputation in gregarious societies, in: ACM SIGecom Exchanges, vol. 3, 2002.
[19] M.P. Singh, Trustworthy service composition: Challenges and research questions, in: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent System, AAMAS, July 2002.
[20] M. Srivatsa, L. Liu, Securing decentralized reputation management using trustguard (Security in grid and distributed systems), Journal of Parallel and Distributed Computing 66 (9) (2006) 1217–1232 (special issue).
[21] M. Srivatsa, L. Xiong, L. Liu, Trustguard: Countering vulnerabilities in reputation management for decentralized overlay networks, in: Proceedings of 14th World Wide Web Conference, WWW 2005, 2005.
[22] Y. Wang, J. Vassileva, Bayesian network-based trust model, in: Proc. of IEEE/WIC International Conference on Web Intelligence, WI 2003, Halifax, Canada, October 2003.
[23] Y. Wang, J. Vassileva, Trust and reputation model in peer-to-peer networks, in: Third International Conference on Peer-to-Peer Computing, P2P'03, September 2003.
[24] A. Whitby, A. Jøsang, J. Indulska, Filtering out unfair ratings in bayesian reputation systems, in: Accepted for the Autonomous Agents and Multi Agent Systems 2004, AAMAS-04 Workshop on "Trust in Agent Societies", New York, July 2004.
[25] U. Wilensky, Netlogo. http://ccl.northwestern.edu/netlogo.
[26] L. Xiong, L. Liu, A reputation-based trust model for peer-to-peer ecommerce communities, in: Proceedings of the IEEE Conference on E-Commerce, June 2003.
[27] P. Yolum, M.P. Singh, Emergent properties of referral systems, in: Proc on Autonomous Agents and Multiagent Systems, 2003.
[28] B. Yu, M.P. Singh, A social mechanism of reputation management in electronic communities, in: Proceedings of Fourth International Workshop on Cooperative Information Agents, Berlin, 2000.
[29] B. Yu, M.P. Singh, Searching social networks, in: Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent System, AAMAS, Melbourne, July 2003.
[30] B. Yu, M.P. Singh, K. Sycara, Developing trust in large-scale peer-to-peer systems, in: Proceedings of First IEEE Symposium on Multi-Agent Security and Survivability, 2004.
[31] R. Zhou, K. Hwang, Gossip-based reputation aggregation in unstructured p2p networks, in: Proceedings of the 2007 International Parallel and Distributed Processing Symposium, IPDPS, Long Beach, March 2007.
[32] R. Zhou, K. Hwang, Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing, IEEE Transactions on Parallel and Distributed Systems 18 (5) (2007).

Zhengqiang Liang is a Ph.D. candidate in computer science at Wayne State University. His researches focus on trusted and cooperative resource sharing in the open environment, trust-based resource scheduling in open scientific discovery infrastructure, next generation internet, P2P systems, and computer economics. He received his B.S. degree in 2001 and M.S. degree in 2003 from Harbin Institute of Technology (HIT) in China, both in Computer Science and Engineering.



Weisong Shi is an Assistant Professor of Computer Science at Wayne State University. He received his B. S. from Xidian University in 1995, and Ph.D. degree from the Chinese Academy of Sciences in 2000, both in Computer Engineering. His current research focuses on high performance computing, distributed systems, and mobile computing. He is the author of the book "Performance Optimization of Software Distributed Shared Memory Systems". He is a recipient of Microsoft Fellowship in 1999, the President outstanding award of the Chinese Academy of Sciences in 2000, one of 100 outstanding Ph.D. dissertations (China) in 2002, "Faculty Research Award" of Wayne State University in 2004 and 2005, the "Best Paper Award" of ICWE'04 and IEEE IPDPS'05. He is a recipient of an NSF CAREER Award in 2007.