A Fine-Grained Reputation System for Reliable Service Selection in Peer-to-Peer Networks

Yanchao Zhang, *Member*, *IEEE*, and Yuguang Fang, *Senior Member*, *IEEE*

Abstract—Distributed peer-to-peer (P2P) applications have been gaining momentum recently. In such applications, all participants are equal peers simultaneously functioning as both clients and servers to each other. A fundamental problem is, therefore, how to select reliable servers from a vast candidate pool. To answer this important open question, we present a novel reputation system built upon the multivariate Bayesian inference theory. Our system offers a theoretically sound basis for clients to predict the reliability of candidate servers based on self-experiences and feedbacks from peers. In our system, a fine-grained quality of service (QoS) differentiation method is designed to satisfy the diverse QoS needs of individual nodes. Our reputation system is also application-independent and can simultaneously serve unlimited P2P applications of different type. Moreover, it is semidistributed in the sense that all application-related QoS information is stored across system users either in a random fashion or through a distributed hash table (DHT). In addition, we propose to leverage credits and social awareness as reliable means of seeking honest feedbacks. Furthermore, our reputation system well protects the privacy of users offering feedbacks and is secure against various attacks such as defaming, flattering, and the Sybil attack. We confirm the effectiveness and efficiency of the proposed system by extensive simulation results.

Index Terms—P2P, QoS, reliability, reputation, security, DHT.

1 Introduction

In many distributed peer-to-peer (P2P) applications, such as grid computing [1], it is essential that a client be able to predict the reliability of candidate servers in offering the desired quality of service (QoS). A natural solution is to leverage the reputations of candidate servers. This necessitates the design of a sound reputation system, which is the focus of this paper.

Reputation systems have been investigated extensively in the past, for which a comprehensive survey can be found in [2]. In such a system, users share QoS experiences and consult others' feedbacks on candidate servers before making a choice. Most previous proposals focus on devising *reputation engines* that derive dependable reputation scores for servers. Among them, the line of approaches [3], [4], [5], [6] based on single-variate Bayesian inference [7] are notable for their firm basis in statistics. This is in contrast to the intuitive and ad hoc natures of most other reputation engines, as noted in [4]. The main drawback of [3], [4], [5], [6], is that they all classify a service as either *good* or *bad* without any interim state. Such a binary QoS differentiation method limits their potential in many P2P applications in which servers have diverse capabilities and clients have various QoS demands. In addition, no strong incentives are designed to stimulate honest participation in the reputation system. These solutions do not consider protecting the privacy of references either, which is important for seeking honest feedback from them (see Section 4.4).

This paper presents the design of a novel reputation system with the following notable properties:

- *QoS-aware*. We devise a novel Dirichlet reputation engine based on multivariate Bayesian inference [7]. Firmly rooted in statistics, our design can satisfy the diverse QoS requirements of individual nodes by a fine-grained QoS differentiation method.
- *Incentive-aware*. We motivate honest participation in the reputation system by charging users who inquire about others' reputations and rewarding those who provide honest feedbacks on inquired servers.
- *Socially aware*. We incorporate the concept of social groups into the reputation system design as a reliable means of soliciting honest feedback and alleviating the *cold-start* problem (see Section 4.1). This design is motivated by the sociological fact that people tend to contribute to the associated social groups.
- *Application-independent*. Unlike many previous solutions all designed for a concrete P2P application, our reputation system can simultaneously serve unlimited P2P applications of different type. This can greatly amortize the design and development costs of the reputation system.
- *Semidistributed*. The proposed system features a central server[1] that maintains user accounts and answers reputation inquiries. All application-related QoS information, however, is stored across system users either in a random fashion or through a distributed hash table (DHT) [8].

1. The central server can be replaced with a distributed server cluster for reasons of fault tolerance, load balance, and resilience to DoS/DDoS attacks.

- *Y. Zhang is with Department of Electrical and Computer Engineering, New Jersey Institute of Technology, University Heights, Newark, NJ 07102. E-mail: yczhang@njit.edu.*
- *Y. Fang is with the Department of Electrical and Computer Engineering, University of Florida, 435 Engineering Building, PO Box 116130, Gainesville, FL 32611. E-mail: fang@ece.ufl.edu.*

Manuscript received 18 Nov. 2005; revised 5 July 2006; accepted 22 Sept. 2006; published online 9 Jan. 2007.
Recommended for acceptance by J.-P. Sheu.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0482-1105.
Digital Object Identifier no. 10.1109/TPDS.2007.1043.

- *Secure*. The proposed system can well protect the privacy of references and withstand various misbehaviors, such as defaming and flattering, by lightweight techniques like multivariate outlier detection [9] and symmetric-key cryptographic functions.

The rest of this paper is organized as follows: Section 2 introduces some related terms and definitions as well as the Dirichlet-Multinomial model on which our system is built. This is followed by the detailed system design in Section 3. Next, we illustrate a number of challenges facing the design of a practical reputation system along with the corresponding solutions. Section 5 evaluates the performance of our reputation system. We then survey related work in Section 6 and end with conclusions and future work.

2 Preliminaries

2.1 Terms and Definitions

Users of our reputation system can be classified as clients, servers, and *reference*s giving feedbacks on servers. A feedback reflects a reference's own QoS experiences with the referred server. Each user may play one of the three roles under different contexts. In this paper, we assume *rational* users who only attempt to misbehave if the expected benefit of doing so is greater than that of acting honestly. How to deal with *malicious* users who intend to interrupt the system functioning without considering their own gains is beyond the scope of this paper. We also refer to each interaction between a client and server pair as a *transaction*.

We define the *reputation* of a server as the probability that he[2] is expected to demonstrate a certain behavior, as assessed by a client based on self-experiences with and other users' feedbacks on him. This definition implies that a server can affect his reputation by different behaviors, but cannot decide it. Reputation is also *application-dependent*. For example, a user may be reputable in grid computing but notorious in file sharing. A client considers a server *reliable* if believing that server to meet his personal QoS need with a sufficiently high probability. The most important factor affecting the reliability decision-making process is the reputation of that server.

2.2 The Dirichlet-Multinomial Model

Our system is firmly rooted in the classical Bayesian inference theory used to estimate one or more unknown quantities from the results of a sequence of *multinomial trials*. For clarity, we outline the adopted Dirichlet-Multinomial model as follows and refer to [7] for more details.

A multinomial trials process is a sequence of independent, identically distributed (IID) random variables U_1, $\mathrm{U}_2, \ldots$, each taking one of k possible outcomes $\{o_i\}_{i=1}^{k}$. We then denote the common probability density function (PDF) of the trial variables by $p_i = \Pr(\mathrm{U}_j = o_i)$ for $1 \le i \le k$, where $p_i > 0$ and $\sum_{i=1}^{k} p_i = 1$. Let $\mathbf{p} = (p_1, \ldots, p_k)$ and $\mathbf{z} = (z_1, \ldots, z_k)$, which is the vector of observation counts of each outcome after N multinomial trials, namely, $\sum_{i=1}^{k} z_i = N$. The multinomial sampling distribution [7] states that

$$f(\mathbf{z}|\mathbf{p}) = \mathrm{Mult}(N|p_1, \ldots, p_k) = \frac{N!}{\prod_{i=1}^{k} z_i!} \prod_{i=1}^{k} p_i^{z_i}. \quad (1)$$

As a common practice in Bayesian inference, assume that $\mathbf{p}$ has a conjugate prior distribution[3] known as the Dirichlet,

$$f(\mathbf{p}) = \mathrm{Dir}(\mathbf{p}|\alpha_1, \ldots, \alpha_k) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} p_i^{\alpha_i - 1}, \quad (2)$$

where $p_i \neq 0$ if $\alpha_i < 1$ and Γ is the gamma function.[4] The positive parameters α_i can be interpreted as "prior observation counts" for events governed by p_i. Then, the posterior distribution is also Dirichlet [7]:

$$\begin{aligned} f(\mathbf{p}|\mathbf{z}) &= \frac{f(\mathbf{z}|\mathbf{p}) \times f(\mathbf{p})}{f(\mathbf{z})} \\ &= \frac{\Gamma(\sum_{i=1}^{k} (\alpha_i + z_i))}{\prod_{i=1}^{k} \Gamma(\alpha_i + z_i)} \prod_{i=1}^{k} p_i^{\alpha_i + z_i - 1} \\ &= \mathrm{Dir}(\mathbf{p}|\alpha_1 + z_1, \ldots, \alpha_k + z_k). \end{aligned} \quad (3)$$

The posterior distribution can be used to make statements about $\mathbf{p}$ considered as a set of random quantities. The posterior mean of p_i, which may be interpreted as the posterior probability of observing outcome o_i in a future multinomial trial, is

$$\mathrm{E}[p_i|\mathbf{z}] = \frac{\alpha_i + z_i}{\sum_{i=1}^{k} (\alpha_i + z_i)}. \quad (4)$$

To apply the Dirichlet-Multinomial model, we need to first specify the prior distribution $f(\mathbf{p})$. In our reputation system, we always use a uniform prior distribution by setting $\alpha_i = 1$ for $1 \le i \le k$. This is equivalent to a priori assuming that each o_i is likely to occur with the same probability, as $\mathrm{E}[p_i|0] = \frac{1}{k}$.

3 System Design

This section illustrates the design of our reputation system. We first give a design overview and explain how user accounts are maintained. Then we present the concept of fine-grained QoS experience vectors and how to efficiently store such vectors. Finally, we detail the process of reputation query and reliability assessment as well as how to stimulate participation in the reputation system.

3.1 Design Overview

Fig. 1 depicts the architecture of our reputation system consisting of a central server and distributed users. Extending our system to a fully distributed one is part of our ongoing work. The central server mainly comprises four subcomponents: an *account manager* in charge of registering users and crediting/debiting user accounts, a *query processor* dealing with reputation queries from system users, a *feedback collector* gathering feedbacks on queried servers from system users, and a *reputation engine* deriving reputation scores for queried servers based on the collected feedbacks. For reasons of fault tolerance, load balance, and resilience to DoS/DDoS attacks, the central server may be replaced with a distributed server cluster in practice. We, however, focus on the single server case in this paper for

2. No gender implication.

3. The property that the posterior distribution follows the same parametric form as the prior distribution is called *conjugacy* [7].

4. If x is an integer, $\Gamma(x) = (x-1)!$.

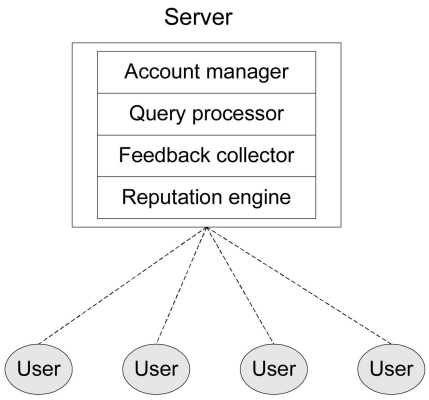


Fig. 1. The reputation system architecture.

ease of presentation. Each user logs QoS experiences with servers after each transaction. Upon a query from the central server, a user returns the QoS experiences (if any) with the queried server. In addition, a user may inquire the central server about the reputation of a candidate server before transacting with him.

The operations of our reputation system can be best illustrated by the following example in grid computing. Suppose that Alice desires some computation services from Bob, and that a higher QoS is associated with a greater monetary cost and vice versa. Since Alice is asked to prepay the service, she wishes to assess the reputation of Bob before transacting with him. She achieves this by making a query to the central server. Upon the query by Alice, the central server collects feedbacks on Bob from some other users, based on which to derive a reputation score for Bob which, in turn, is returned to Alice. In addition, the central server credits the accounts of the users offering honest feedbacks and debits Alice's account accordingly. Then Alice can assess Bob's reliability based on the reputation score and decide whether to transact with him.

Realizing the above procedure requires solutions to the following questions: First, how does the central server maintain user account information to ensure secure operations of the reputation system? Second, how do system users record their QoS experiences to enable fine-grained QoS differentiation? Third, how does a system user check with the central server about the reputation of a candidate server and determine his reliability? Fourth, how does the central server search feedbacks on queried servers and differentiate between honest and dishonest feedbacks? Last, how does the central server urge honest participation in the reputation system? In the following, we will answer the above questions one by one.

3.2 User Account Maintenance

Similarly to Gmail,[5] our reputation system adopts an invitation-based registration policy to reduce the amount of abuse, so misbehaving users cannot register near-infinite number of accounts as with a completely open registration policy. In particular, the central server only registers users with an invitation code which is received, for example, from an existing account holder or through their mobile phone.

5. Gmail is a free Web mail and POP e-mail service provided by Google, Inc.

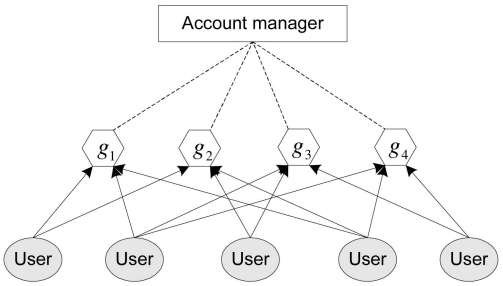


Fig. 2. A snapshot of the social network of system users with four social groups and five users, where the directed link indicates group affiliation.

We assume that the central server maintains a sufficiently long master key $\mathcal{K}$ which it keeps confidential. The central server assigns a unique identifier ID_α to each registered user α. Let $\mathrm{HMAC}_k(m)$ denote a keyed-hash message authentication code (HMAC) [10] of message m using a symmetric key k. The central server also sends a shared key $k_\alpha = \mathrm{HMAC}_{\mathcal{K}}(ID_\alpha)$ to user α through the Transport Layer Security (TLS) protocol [11]. The central server need not store all the individual shared keys to save its storage. Instead, it can derive any shared key using $\mathcal{K}$ on the fly as needed. Since $\mathrm{HMAC}_k(m)$ is computationally efficient, the computational overhead is negligible. Hereafter, when saying that a message is *securely* sent or transmitted between two entities, we mean that the message is encrypted and authenticated with efficient symmetric-key algorithms based on their shared key. For example, a secure message m from user α to the central server is of format $\langle \{m\}_{k_\alpha}, \mathrm{HMAC}_{k_\alpha}(m) \rangle$, where $\{m\}_k$ denotes the encryption of message m using a symmetric key k. Upon receipt of it, the central server derives k_α and then uses it to decrypt $\{m\}_{k_\alpha}$ and compute an HMAC on m. If the HMAC matches the received one, the central server is assured that m indeed came from user α.

In addition to user accounts, the central server maintains a social network for system users, of which a snapshot is shown in Fig. 2. Let $\mathcal{G} = \{G_i\}_{i=1}^{\infty}$ be the set of network social groups the central server maintains. In our system, each user is affiliated with at least one social group $G_i \in \mathcal{G}$, where the cardinality of G_i is $|G_i| \geq 1$. That is, each user at least belongs to a social group containing only himself. A joint request for group G_i needs the consensus of $\delta (0 \leq \delta \leq 1)$ fraction of existing group members, where δ is a system parameter chosen by the central server. For example, supposing that user α requests to join group G_i, the central server performs the following operations:

1. Randomly select $\lceil \delta |G_i| \rceil$ existing members of group G_i.
2. Poll each chosen user β about user α's join request.
3. Verify the response $\mathrm{HMAC}_{k_\beta}(ID_\alpha, ID_{G_i})$ from each chosen user β to make sure that the response was indeed sent by β, where ID_{G_i} indicates the unique ID of G_i.
4. Link user α to group G_i if all the responses are authentic.

Note that similar operations can be performed to expel a faulty group member. For example, the central server need receive $\lceil \delta |G_i| \rceil$ (or another predetermined number) authenticated requests from members of G_i to remove α. We will show the use of the social network in solving the *cold-start* problem in Section 4.1.

3.3 Fine-Grained QoS Experience Vectors

In this section, we introduce an important data structure, called a *QoS experience vector* (Q-vector for short), to record users' QoS experiences. For ease of presentation, we take client α and server s with regard to application $\mathcal{C}$ as an example hereafter.

Our system adopts a fine-grained QoS differentiation method in contrast to the binary one used in [3], [4], [5], [6]. Assume that the QoS of $\mathcal{C}$ is divided into publicly known $\varpi \geq 2$ levels and that any client can unambiguously map the QoS he experienced into one of the ϖ levels after each transaction. We denote by $E^{\mathcal{C}}_{\alpha,s} = (ID_s, \mathcal{C}, e^1_{\alpha,s}, \cdots, e^{\varpi}_{\alpha,s})$ the Q-vector of client α for server s with respect to application $\mathcal{C}$. Each $e^i_{\alpha,s} (1 \leq i \leq \varpi)$ is a counter corresponding to the ith QoS level and initialized to zero. After each transaction with s, user α maps his QoS experience into one of the ϖ levels and then increases the corresponding counter by one. Note that each user only needs to maintain a Q-vector for who has ever offered services to him.

Old service experiences may not always be relevant for determining the current reliability of servers that may vary their behaviors or service qualities over time. To deal with this situation, we introduce a *discount factor* ρ between [0, 1.0] to assign more weight to recency. At regular intervals, user α should update $\{e^i_{\alpha,s}\}^{\varpi}_{i=1}$ to $\{\rho e^i_{\alpha,s}\}^{\varpi}_{i=1}$, so the counter values may not be integers. If the counter values are too small to be meaningful, e.g., all much smaller than 1, user α can delete $E^{\mathcal{C}}_{\alpha,s}$ to save memory space. Discounting the past not only can help identify servers who offer good services initially and bad services afterwards, but also can permit a disreputable server to reform by starting to provide high-quality services. Following the same process, each user needs to periodically discount all the Q-vectors he maintains. Note that each user has the right incentive to properly discount his QoS-vectors that provide important inputs to his reliability decision-making process.

The discount factor ρ and interval are system parameters determined by the central server to control how fast past experiences are forgotten. Obviously, the smaller ρ, the shorter the discount interval, the more quickly past experiences fall into oblivion, and vice versa. We will further show the effects of these two parameters using simulations in Section 5.4.

3.4 Deriving Reputation Scores from Self-Experiences

Now, we discuss how user α derives a reputation score for server s based on his own QoS experiences with s, i.e., $E^{\mathcal{C}}_{\alpha,s}$. Let $p_{s,i}$ denote the probability of s providing the ith QoS level of application $\mathcal{C}$. Whenever updating $E^{\mathcal{C}}_{\alpha,s}$ other than using the discounting method, user α generates a reputation score $R^{\mathcal{C}}_{\alpha,s} = (ID_\alpha, ID_s, \mathcal{C}, r^1_{\alpha,s}, \cdots, r^{\varpi}_{\alpha,s})$, where

$$r^i_{\alpha,s} = \frac{e^i_{\alpha,s} + 1}{\sum_{i=1}^{\varpi} \left(e^i_{\alpha,s} + 1 \right)}$$

is the posterior mean of $p_{s,i}$ computed according to (4). $R^{\mathcal{C}}_{\alpha,s}$ is one of the factors affecting the decision of α on s's reliability, as shown in Section 3.7.

3.5 Storage of Reputation Scores

In our reputation system, $R^{\mathcal{C}}_{\alpha,s}$ also serves as α's feedback on s regarding application $\mathcal{C}$. The next question is how to store such reputation scores to enable efficient queries by the central server. We are aware of the following four approaches.

3.5.1 Approach 1

In the first approach, whenever deriving a new reputation score, a user securely sends it to the central server which saves all the received reputation scores for later use. This method, though simple, would cause significant storage overhead on the central server, as our reputation system may involve numerous P2P applications and users and thus contain thousands of millions of reputation scores. As a result, we discard this method in our design.

3.5.2 Approach 2

In this approach, each user independently stores his reputation scores. Upon receiving a query from the central server, he responds with the reputation score (if any) for the queried server. This approach can significantly reduce the storage overhead of the central server and the communication overhead of dynamically submitting reputation scores. The drawback, however, lies in the low query efficiency. The reason is that the central server has no knowledge about which users have the desired reputation scores, so it may need to send a number of queries. We will dwell on this point in Section 3.5.

3.5.3 Approach 3

The third approach is a novel combination of the first approach with a distributed hash table (DHT) [8]. In particular, the central server assigns to each user α a *virtual ID*, $\widetilde{ID_\alpha} = \lceil h(ID_\alpha) \rceil^\gamma$, where h indicates a fast hash function such as SHA-1 [12] and $\lceil m \rceil^\gamma$ denotes the first γ bits of value m. Hereafter, we may also refer to user α as ID_α or $\widetilde{ID_\alpha}$. When a user submits a reputation score, the central server dispatches the score to several users before discarding it. The purpose is to harness the storage capacity of all users to provide distributed storage of reputation scores. Consider, for example, user α who derived a new reputation score $R^{\mathcal{C}}_{\alpha,s}$. He securely transmits it to the central server which, in turn, forwards $R^{\mathcal{C}}_{\alpha,s}$ to selected users by the following process.

The central server first calculates λ values of γ bits called *score IDs*, $\{\lceil h(ID_s, \mathcal{C}, i) \rceil^\gamma\}^{\lambda}_{i=1}$, where $\lambda \geq 1$ is a system parameter called the *redundancy index*. Let n be an integer between 0 and $2^\gamma - 1$ and $successor(n)$ be the first virtual user ID clockwise from n, if virtual user IDs and score IDs are represented as a circle of numbers of 0 to $2^\gamma - 1$. Then, the central server securely transmits $\langle ID_s, \mathcal{C}, \{R^{\mathcal{C}}_{\alpha,s}\}_{\mathcal{K}} \rangle$ individually to users $\{successor(\lceil h(ID_s, \mathcal{C}, i) \rceil^\gamma)\}^{\lambda}_{i=1}$. The encryption of $R^{\mathcal{C}}_{\alpha,s}$ is to ensure that only the central server can

decrypt and know the content of $R^{\mathcal{C}}_{\alpha,s}$.[6] Upon receiving $\langle ID_s, \mathcal{C}, \{R^{\mathcal{C}}_{\alpha,s}\}_{\mathcal{K}}\rangle$, the chosen users save it for later queries by the central server. In addition, a timer needs to be set for $\langle ID_s, \mathcal{C}, \{R^{\mathcal{C}}_{\alpha,s}\}_{\mathcal{K}}\rangle$, which is deleted after the timer expires. Why does the central server let λ users store $\langle ID_s, \mathcal{C}, \{R^{\mathcal{C}}_{\alpha,s}\}_{\mathcal{K}}\rangle$? This is to improve system fault tolerance in case some users cannot respond to its query for various reasons, such as going offline. The choice of λ represents a trade-off between fault tolerance and system overhead: the larger λ, the higher fault tolerance, and the larger the communication overhead and the average storage cost of users, and vice versa.

Our system can well handle dynamic user sign-up or sign-off requests. For instance, when user β signs up for the reputation system, the successor of some score IDs allocated to user $successor(\widetilde{ID_\beta})$ may become $\widetilde{ID_\beta}$. The central server then redistributes such reputation scores from user $successor(\widetilde{ID_\beta})$ to user β. When user β signs off from the reputation system, all the reputation scores he stores are reallocated to user $successor(\widetilde{ID_\beta})$. No other actions need be taken in the presence of user sign-ups or sign-offs.

Another design issue is how to balance the load across system users. Since the distribution of user virtual IDs and reputation score IDs is unlikely to be uniform in practice, some users may store many more reputation scores than others, leading to load imbalance. A previous solution is to let each user have multiple virtual IDs [8], [13] to ensure a more uniform coverage of the range $[0, 2^\gamma - 1]$. In our system, this means that each user need store all the reputation scores with successor IDs equal to one of his virtual IDs. This technique, however, can only help balance the distribution of reputation score IDs on system users, and cannot address another reason for load imbalance which is unique to our reputation system. Note that score IDs are generated based on server IDs and application indexes. It is very possible that some servers are highly popular and serve many more users than others. As a result, there will be many more reputation scores associated with their score IDs and two users may have distinct storage and communication costs even when they are assigned the same number of score IDs.

We further alleviate the load imbalance by introducing a *popularity index* technique. In particular, for each application $\mathcal{C}$, the central server maintains a popularity index containing the top υ users for which the maximum reputation scores are submitted during the last time period ψ. Upon receiving a reputation score for any indexed user s, the central server picks a random $x \in [1, \phi]$ and sends the encrypted reputation score to users $\{successor(\lceil h(ID_s \| c \| i \| x)\rceil^\gamma)\}_{i=1}^{\lambda}$ as before. Therefore, the storage of the reputation scores for any indexed user is uniformly distributed to $\phi\lambda$ instead of λ users. The choice of ϕ, called the *popularity branching factor*, determines a trade-off between load balance and system overhead. It is also possible to use different values of ϕ according to a user's rank in the popularity index. A popularity index may change after each time period ψ, in which case the central server need redistribute reputation scores accordingly.

As compared to Approach 2, this approach can ensure deterministic queries by the central server because it knows exactly who store the desired reputation scores. This is achieved at the cost of increased communication overhead incurred by dynamic distributions of reputation scores.

3.5.4 Approach 4

In Approach 3, the central server is involved in distributing reputation scores among users. We can reduce the load of the central server by letting a user directly send a reputation score to corresponding other users instead of via the central server. For this purpose, a distributed routing protocol is required to enable a user to locate other users who should store his reputation score. This can be achieved by Chord [8] or any other distributed P2P lookup protocol. Note that each user needs to periodically download the popularity index for the desired application from the central server with the purpose of correctly achieving load balance.

An important problem Chord does not address is the secure communication between two users, as otherwise an attacker may easily impersonate authentic system users to disseminate or even harmful useless information. To address this issue, we require the central server to assign an ID-based key $IK_\alpha = \mathcal{K}H(ID_\alpha)$ to each user α upon registration, where $H(x)$ indicates a hash function mapping an input x to an element of a cyclic group $\mathbb{G}_1$ defined in the Appendix. Assume that user α computes a new reputation score $R^{\mathcal{C}}_{\alpha,s}$ to be sent to user β selected by the same method in Approach 3. He derives a shared key $k_{\alpha,\beta}$ by computing $k_{\alpha,\beta} = \hat{e}(IK_\alpha, H(ID_\beta))$, get β's IP address through Chord, and sends $\langle M, \mathrm{HMAC}_{k_{\alpha,\beta}}(M)\rangle$ to β, where $\hat{e}$ is the bilinear pairing function defined in the Appendix and $M := \langle ID_\alpha, ID_s, \mathcal{C}, \{R^{\mathcal{C}}_{\alpha,s}\}_{k_\alpha}\rangle$. The purpose of $\{R^{\mathcal{C}}_{\alpha,s}\}_{k_\alpha}$ is to ensure the privacy of user α (see Section 4.4) because only the central server can decrypt it using k_α.

Upon receiving the message, user β generates $k_{\beta,\alpha} = \hat{e}(IK_\beta, H(ID_\beta))$, which is equal to $k_{\alpha,\beta}$. The reason is that $k_{\alpha,\beta} = k_{\beta,\alpha}$ because

$$\begin{aligned}\hat{e}(IK_\alpha, H(ID_\beta)) &= \hat{e}(H(ID_\alpha), H(ID_\beta))^{\mathcal{K}} \\ &= \hat{e}(H(ID_\beta), H(ID_\alpha))^{\mathcal{K}} \\ &= \hat{e}(\mathcal{K}H(ID_\beta), H(ID_\alpha)) \\ &= \hat{e}(IK_\beta, H(ID_\beta)).\end{aligned} \tag{5}$$

The first-line and third-line equations are due to the bilinearity of $\hat{e}$, and the second-line equation is because of its symmetry. Then, user β calculates $\mathrm{HMAC}_{k_{\beta,\alpha}}(M)$ and compares it with $\mathrm{HMAC}_{k_{\alpha,\beta}}(M)$. If they are equal, β is assured that the reputation score indeed came from α, who is also a legitimate user of the reputation system. Note that an attacker may impersonate user α, but he would not have been in possession of IK_α, thus being unable to derive a correct $k_{\alpha,\beta}$.

In contrast to Approach 3, this approach can greatly reduce the load of the central server. The cost is that each user has to be involved in the Chord operations, which can

6. Note that the informative structure of $R^{\mathcal{C}}_{\alpha,s}$ means that $R^{\mathcal{C}}_{\alpha,s}$ is self-authenticated, as any modification on $\{R^{\mathcal{C}}_{\alpha,s}\}_{\mathcal{K}}$ will render a meaningless decryption result.

be greatly amortized if our reputation system is integrated with any P2P application (e.g., file sharing) built upon Chord.

3.6 Query for Reputation Scores

Assume that user α cannot predict the reliability of server s based on his own QoS experiences with s. He securely transmits a *reputation query* containing $\langle ID_s, \mathcal{C}, n_r \rangle$ to the central server, where n_r is the number of desired references to s. Upon receipt of the query, the central server performs the following different operations, depending on how reputation scores are stored.

3.6.1 Case 1

Consider first the case that the each user independently stores his reputation scores (Approach 2). The central server securely sends a query $\langle ID_s, \mathcal{C} \rangle$ to each randomly chosen user β. If user β has $R^{\mathcal{C}}_{\beta,s}$, he securely sends it back to the central server. Let p_s be the probability of each user having a reputation score for s, and let X be a random variable denoting the number of users the central server needs to inquire until obtaining n_r reputation scores. Then, we have

$$\Pr(X = x) = \binom{x-1}{n_r-1} p_s^{n_r} (1-p_s)^{x-n_r},$$

and $\mathrm{E}[X] = \frac{n_r}{p_s}$ and $\mathrm{Var}[X] = \frac{n_r(1-p_s)}{p_s^2}$. The inquiry overhead is in inverse proportion to the popularity of servers. For example, if $p_s = 0.005$ or 0.025 and $n_r = 20$, the central server has to on the average inquire 4,000 and 800 users, respectively, until obtaining 20 reputation scores.

3.6.2 Case 2

Now, we consider the case in which reputation scores are stored through the DHT (i.e., Approaches 3 and 4). For ease of presentation, below we assume that s is not on the popularity index of application $\mathcal{C}$, but the extension to any indexed server is straightforward. Let user $\widetilde{ID_\beta}$ be a random online one among users $\{successor(\lceil h(ID_s \| \mathcal{C} \| i) \rceil^\gamma)\}_{i=1}^{\lambda}$. The central server securely sends a query $\langle ID_s, \mathcal{C}, n_r \rangle$ to user β. Upon receipt of the query, if user β stores fewer than n_r encrypted reputation scores, he sends all of them to the central server; otherwise, he sends randomly selected n_r of them. After receiving the encrypted reputation scores, the central server decrypts them using either the master key $\mathcal{K}$ (Approach 3) or the individual shared keys (Approach 4). One may consider reducing the load on the central server by letting user α directly send reputation queries for s to other users. This is impossible in our system because all the reputation scores have been encrypted and can only be accessible to the central server for protecting references' privacy.

In both cases, let Ω be the set of reputation scores for s returned to the central server, where $|\Omega| \leq n_r$. The central server cannot simply aggregate these reputation scores because there might be *outliers* in Ω which are very different from the rest based on some measure. These outliers might have been created by users who attempt to defame or flatter user s, so it is necessary to identify and remove them from Ω. Assuming that nonoutliers are the majority in Ω, we can apply any existing *multivariate outlier detection* technique to find outliers in Ω. Below we describe a *distance-based* method [14] for its simplicity and efficiency. In particular, we calculate the sum of the euclidean distances of each reputation score in Ω from all the others. *A reputation score is said to be an outlier if there are no more than $\eta - 1$ other distance sums larger than its distance sum.* Here, η is called an *outlier index* between $\left[1, \lfloor \frac{n_r}{2} \rfloor\right]$ decided by the central server. There is also a trade-off regarding the choice of η: if η is large, more outliers will be removed but more nonoutliers may be mislabeled as outliers; and vice versa. Let Ω' be the set of reputation scores from which η outliers have been eliminated. The central server generates a reputation score $R^{\mathcal{C}}_s = \{ID_s, \mathcal{C}, r^1_s, \cdots, r^{\varpi}_s\}$, where

$$r^i_s = \frac{\sum_{x \in \Omega'} r^i_{x,s}}{|\Omega'|}.$$

Finally, the central server securely sends R_s to the requesting user α. If no reputation scores for s are found, the central server also need inform α about it.

3.7 Reliability Assessment

Assume that user α has computed $R^{\mathcal{C}}_{\alpha,s}$ and obtained $R^{\mathcal{C}}_s$ from the central server and that his lowest and highest tolerable QoS levels are ϑ^L_α and ϑ^H_α, both in$[1, \varpi]$. He computes a *reliability indicator* as

$$I^{\mathcal{C}}_{\alpha,s} = (1 - \epsilon_\alpha) \sum_{i=\vartheta^L_\alpha}^{\vartheta^H_\alpha} r^i_{\alpha,s} + \epsilon_\alpha \sum_{i=\vartheta^L_\alpha}^{\vartheta^H_\alpha} r^i_s. \quad (6)$$

Here, $\varepsilon_\alpha \in [0, 1.0]$ is called a *trust indicator* chosen by user α to reflect the level of his trust on others' QoS experiences with s. User α considers s reliable if $I_{\alpha,s} \geq \varphi_\alpha$ and unreliable otherwise, where $\varphi_a \in [0, 1.0]$ is a threshold chosen by α itself. In the former case, α starts to transact with s. Otherwise, he starts to check another candidate server, if any.

It is obvious that our fine-grained QoS differentiation method enables independent reliability decision-making of individual nodes with diverse QoS requirements. Consider grid computing [1] as an example whose QoS is defined as the time t (in seconds) taken to finish a unit computation task. Assume that if the binary QoS differentiation method [3], [4], [5], [6] is used, the QoS is classified as *good* for $t < 60$ and *bad* otherwise, namely, $\varpi = 2$. In contrast, in our reputation system, the QoS is classified as six levels $(\varpi = 6) : t \geq 60, \quad 48 \leq t < 60, \quad 36 \leq t < 48, \quad 24 \leq t < 36, \quad 12 \leq t < 24$, and $t < 12$. Suppose that the QoS requirements of users α_1 and α_2 are $t < 60$ and $t < 36$, respectively. Obviously, the binary method cannot enable α_2 to correctly assess the reliability of a candidate server, while our method can do so.

3.8 Stimulating Participation With Credits

Another issue left for discussion is how to motivate users to respond to reputation queries and participate in storing reputation scores if the DHT is used. This is important because users in the open system have individual interests and are generally reluctant to serve others for free [15]. We address this issue by a credit-based approach. In particular, upon the registration of any user α, the central server opens a credit account for him with zero balance. Below, we

continue with the previous example to illustrate the crediting/debiting process of users' credit accounts.

Upon receiving a reputation query $\langle ID_s, \mathcal{C}, n_r \rangle$ from user α, the central server processes it only when his credit balance exceeds n_r. After obtaining Ω', the central server debits α's account with $|\Omega'|$ credits and increases the account of each reference in Ω' by 1 credit. This means that user α needs to pay more to learn more QoS experiences of others. Since reputation queries require credits, a user is well urged to share his reputation scores to earn credits for future use. In addition, if the DHT is used, the central server can grant certain credits to the user who returns stored reputation scores. Alternatively, willingness to store and return reputation scores can be part of the user agreement that any user has to comply with.

Intuitively, one may think of punishing references who offer outlier reputation scores identified in Ω. This measure, however, may discourage good users from sharing their QoS experiences. The reason is that some users may have slightly different QoS experiences with the same server from most other users due to subjective judgment. Even when offering honest reputation scores, these users may still be classified as outliers. As a result, we believe it enough not to reward them without further punishing them.

4 Problems and Solutions

In this section, we describe some important problems to be aware of when designing a practical reputation system, as well as our corresponding solutions.

4.1 Cold-Start

There are two types of *cold-start* problems to be considered.

The first one is pertinent to our reputation system, by which we mean that a system newcomer with initially zero credit is unable to immediately perform reputation queries. He has no reputation scores to share with others for earning credits, either. Of course, he can gradually create some reputation scores by temporarily taking the risk of assuming that all servers are reliable. We provide a better solution by utilizing the social network the central server maintains (see Section 3.2). In particular, we allow credit transfer between system users so that existing members of a social group can lend credits to either a group newcomer or an existing member with insufficient credits. This measure guarantees that at least part of a social group has to actively share their reputation scores. It is possible that some selfish users may exploit this measure to free ride on others' efforts and frequently request credits from social group peers. We assume that social group members are alert to this situation. They can also actively check with the central server about the available credit of any group peer. If some peer always has very few credits, the rest of the social group can request the central server to remove that user from the social group after reaching a consensus.

The second cold-start problem relates to the target P2P applications of our reputation system. In particular, an application newcomer will struggle to act as servers because there are no reputation scores for him, and so clients have no way to assess his reliability. This problem can also be greatly alleviated by using the social network. More specifically, social group peers of the newcomer can initially offer reputation scores for him to make him considered reliable by other users. However, they are unable to overuse this feature because, as time goes on, there will be more reputation scores available that reflect the real QoS offered by the newcomer.

4.2 Change of Identities

A user may rejoin the system under a new identity if he can benefit from doing so. A more advanced version is know as the Sybil attack [16], where an ill-intended user is in control of multiple identities and uses them in concert to take advantage of the system. Such misbehavior is a challenging issue for any practical reputation system. We are not claiming to completely defeat such misbehavior, but attempt to make it more difficult to launch and less beneficial. First, by an invitation-based registration policy (see Section 3.2), our reputation system renders it difficult for an ill-intended user to freely register near-infinite number of accounts. In addition, we can prevent computer-automated registration requests by introducing a CAPCHA [17] into the registration process, which typically requires a user to type the letters of a distorted image, sometimes with the addition of an obscured sequence of letters or digits that appears on the screen. Furthermore, newcomers to our reputation system are granted zero credit, which is the lowest possible credit level. This measure can further discourage a user from changing his identity.

Another form of the above misbehavior is that a user may rejoin a P2P application under a new identity if he knows he has been considered disreputable by many other users.[7] Our reputation system can partially mitigate this misbehavior by the invitation-based registration policy and the consensus-based social group enrollment: Existing reputable group members may refuse the join request of a user who frequently changes his identity. Other countermeasures are totally application-dependent. For example, the application administrator can charge a new user an entry fee. Further investigation of this issue is beyond the scope of this paper.

4.3 QoS Variations over Time

Servers may offer varied QoS over time either deliberatively or unintentionally. For example, a server may initially provide good services to build up a good reputation and offer bad services afterward. On the other hand, a disreputable server may start to provide good services to regain a good reputation. Our reputation system can well accommodate QoS variations, be they good or bad, by discounting the past QoS experiences and deleting too old reputation scores. Also note that, as discussed before, the social network maintained by the central server can help a disreputable user to participate in service provision. That is, the disreputable user is treated as an application newcomer by his social group peers.

4.4 Privacy of References

Resnick and Zeckhauser reported some interesting statistics about eBay's reputation system [18]: Only 0.6 percent

7. The user can learn this by asking his social group peers to perform a reputation query on him.

and 1.6 percent of all the feedbacks provided by buyers and sellers, respectively, were negative, which seem too low to reflect the reality. A convincing explanation is fear of retaliation from the rated party. We address this issue by well protecting the privacy of references. In particular, reputation scores submitted to the central server are encrypted and can only be decrypted by it. Even if the DHT is used, no user can learn the content of the encrypted reputation scores he saves. In addition, the central server only returns to the querying user an aggregated reputation score instead of collected raw reputation scores. Therefore, it is impossible for any server to know the reputation score a particular client gives for him, and clients can be assured of offering honest reputation scores without incurring retaliation.

4.5 Dishonest Reputation Scores

Guaranteeing the privacy of references functions as a double-edged sword to our reputation system. More specifically, it not only encourages well-behaved users to share honest reputation scores, but also facilitates ill-behaved users to offer dishonest ones without fear of being punished. Dishonest reputation scores can be either unfairly positive ones used to flatter conspirators, or unfairly negative ones aimed at defaming other users, say, competitors.

Our reputation system is designed to defend well against dishonest feedbacks. Let us first revisit the process of collecting reputation scores for a queried server (see Section 3.6). If the DHT is not used, the central server inquires random users about desired reputation scores. Under the assumption that well-behaved users are always the majority, the probability of ill-behaved users being chosen will be very low. If the DHT is used, the contacted user returns to the central server some reputation scores randomly chosen from all the reputation scores he records for the queried server. The probability of dishonest reputation scores being selected will be low as well under the same assumption. It is also worth noting that the contacted user, if ill-behaved, cannot send purposefully selected reputation scores to the central server for his blindness to the content of encrypted reputation scores. In both cases, the central server can run a multivariate outlier detection technique to identify outliers or dishonest reputation scores (if any) as the last line of defense. We believe that our countermeasures above are sufficient to discourage ill-behaved yet rational users from propagating dishonest reputation scores, as they might only achieve their ill intention with a rather small probability.

5 Performance Evaluation

In this section, we evaluate the performance of the proposed reputation system through simulations.

5.1 Simulation Setup

Unless otherwise stated, we use grid computing [1] as an example and simulate 100 clients and two servers denoted by s_1 and s_2, respectively. Again, the QoS is defined as the time t (in seconds) taken to finish a unit computation task and classified as six levels: $t \geq 60$ (l_1), $48 \leq t < 60$ (l_2), $36 \leq t < 48$ (l_3), $24 \leq t < 36$ (l_4), $12 \leq t < 24$ (l_5), and $t < 12$ (l_6). We assume that the probabilities of s_1 and s_2 providing each QoS level are {0, 0.05, 0.05, 0.1, 0.3, 0.5} and {0, 0.5, 0.3, 0.1, 0.05, 0.05}, respectively. In addition, the clients are divided into five equally sized groups, of which the lowest tolerable QoS levels are $l_2/l_3/l_4/l_5/l_6$, respectively, and the highest one is all l_6. The simulation is divided into discrete sessions. At the beginning of each session, each client sends two reputation queries to the central server, each requesting 99 references to s_1 and s_2, respectively. To simplify the simulation, we assume that all the clients have the same trust indicator 0.99 (see (6)), which amounts to assuming that each client puts the same trust in his own QoS experiences and those of each other client. Then each client requests service from s_1 or s_2 who has the higher reliability indicator and records his QoS experience after the transaction.

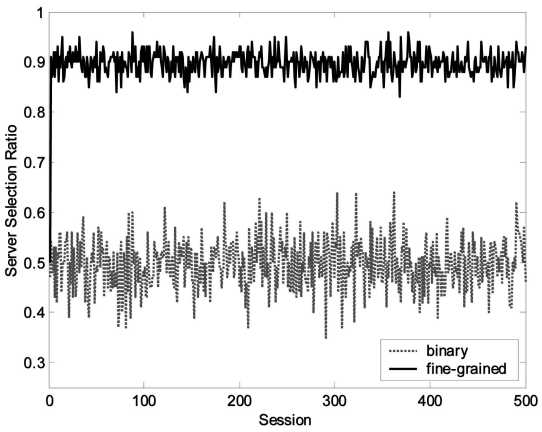


Fig. 3. Comparing server selection ratios.

5.2 Benefits of Fine-Grained QoS Differentiation

Here, we show the advantages of our fine-grained QoS differentiation method over the binary one [3], [4], [5], [6]. For the latter, we assume that the application specifies two QoS levels: *bad* for $t \geq 60$ and *good* otherwise. Based on the behavior profiles of s_1 and s_2 given above, all the clients will derive the same reliability indicator equal to 1.0 for s_1 and s_2, and thus will randomly choose one of them. In contrast, the application specifies six QoS levels defined in Section 5.1 for our fine-grained method. We are also interested in two performance metrics: *server selection ratio* (SSR), defined as the fraction of clients selecting s_1 in each session, and *service contentment ratio* (SCR), defined as the fraction of clients obtaining the desired QoS level in each session.

Fig. 3 shows the SSRs under both methods. In session 1, there were no reputation scores for s_1 and s_2 so that each client considered them equally reliable and randomly chose one of them. Therefore, both methods have almost the same SSR. As time goes on, however, more and more diverse reputation scores for both servers were available with our fine-grained method. As a result, increasingly more clients derived a higher reliability indicator for s_1 and thus selected him, which leads to the shown big SSR difference. In particular, the average and maximum SSR differences are 40.06 percent and 56 percent, respectively, and the variance is 0.32 percent.

Fig. 4 depicts the SCRs under both methods. As we can see, both methods have nearly identical SCRs in session 1 where clients randomly selected one of s_1 and s_2. Since most clients selected the more reliable s_1 afterward with our fine-grained method, there are many more clients whose QoS needs were satisfied across the subsequent sessions. More

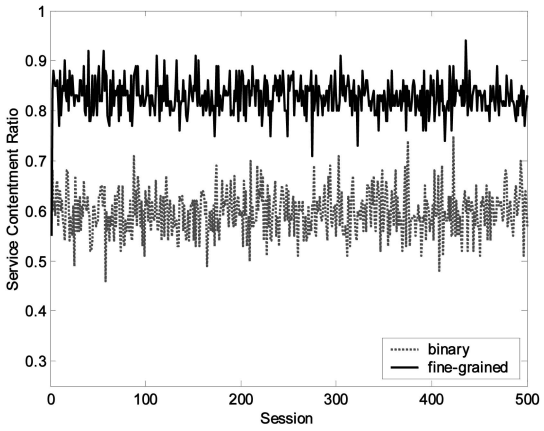


Fig. 4. Comparing service contentment ratios.

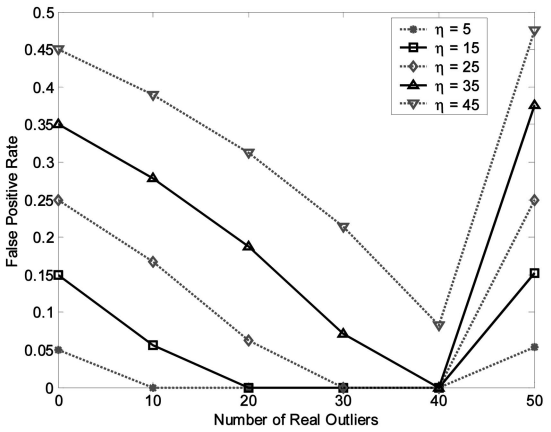


Fig. 6. The false positive rate.

specifically, the average and maximum SCR differences are 23.14 percent and 42 percent, respectively, and the variance is 0.29 percent.

To sum up, the above results clearly demonstrate the benefits of our fine-grained QoS differentiation method in meeting the various QoS needs of clients.

5.3 Filtering Outlier Reputation Scores

In the last section, each client is assumed to always offer honest reputation scores. This subsection studies the efficacy of distance-based multivariate outlier detection in eliminating dishonest reputation scores (or outliers). For lack of space, we focus on filtering defaming outliers, but it should be noted that our approach performs equally well in detecting flattering outliers. In the simulation, an outlier is generated as a set of six random numbers normalized by their sum, which follow the multinomial distribution {0, 0.5, 0.3, 0.1, 0.05, 0.05}. Similarly, a nonoutlier is generated with the multinomial distribution {0, 0.05, 0.05, 0.1, 0.3, 0.5}. We also assume that the central server aggregates the reputation scores for s_1 from all the 100 clients and then determines the reliability of s_1 in offering a QoS level higher than l_3. In addition to s_1's reliability indicator (denoted by I_{s_1}), we have interest in the *false positive rate*, defined as the proportion of nonoutliers that were mislabeled as outliers, and the *false negative rate*, defined as the proportion of outliers that were not detected.

We can see from Fig. 5 that, without outlier detection, I_{s_1} decreases dramatically with the increase of outliers, which is no surprise. In addition, the outlier detection technique can greatly alleviate the impact of outliers and make I_{s_1} approach its true value, 0.9. The larger the outlier index η, the higher the efficacy of outlier detection. Another observation is that outlier detection fails when the percentage of outliers reaches 50 percent, in which case there is no longer clear distinction between outliers and nonoutliers. We believe that this issue cannot be easily solved by any technical means. This scenario also reflects an underlying assumption of all practical reputation systems: Outliers are assumed to be always the minority.

Figs. 6 and 7 depict the false positive and negative rates with varying outlier indexes, respectively. As we can see, when the number of outliers is fixed, a larger outlier index η can result in a higher false positive rate but a lower false negative rate, and vice versa. We opt for a larger η because it may lead to more trustable reliability decisions, though it may cause some nonoutliers to be unable to get deserved credits. In practice, the central server can decide η and dynamically adjust it by empirical means. Further investigation on how to choose η is part of our future work.

5.4 Effects of Discounting the Past

In this section, we evaluate the impact of the aforementioned discount method on the reliability decision-making process. For this purpose, we assume that each client always selects s_1

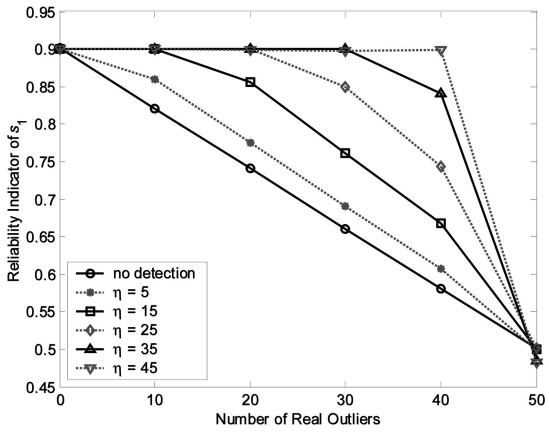


Fig. 5. The reliability indicator for s_1 offering a QoS level higher than l_3.

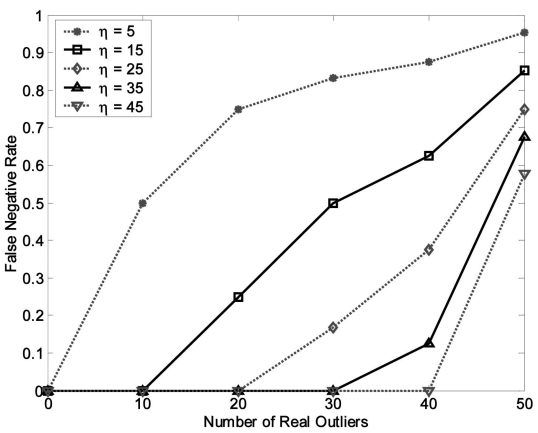


Fig. 7. The false negative rate.

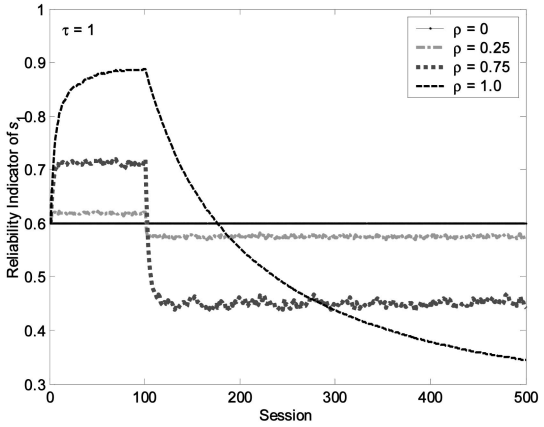


Fig. 8. Average reputation indicators for s_1 with varying discount factors, where the discount interval τ is fixed to be 1 session.

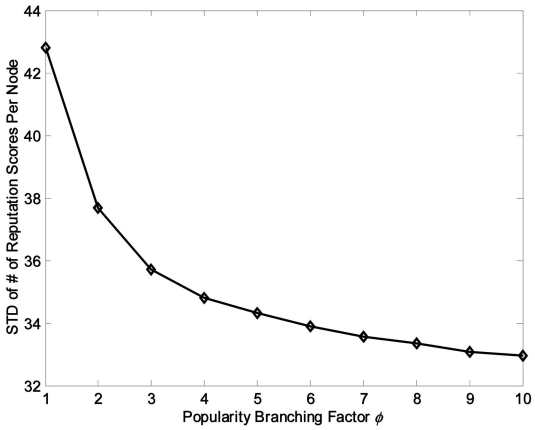


Fig. 10. The standard deviation of the number of reputation scores per node.

as the server and evaluates its reliability in offering a QoS level higher than l_3 based on his own QoS experiences in past sessions. We also assume that the behavior profile of s_1 is {0, 0.5, 0.3, 0.1, 0.05, 0.05} for the first 100 sessions and changes to {0, 0.05, 0.05, 0.1, 0.3, 0.5} afterward.

Fig. 8 shows the average reputation indicators for s_1 with varying discount factors across sessions, where the discount interval τ is fixed to be one session. It is obvious that a smaller discount factor, say $\rho = 0.25$, can help catch the QoS variations of s_1 more quickly at the cost of generating reputation indicators far from the true value. In particular, when $\rho = 0$, each client discards all his past QoS experiences and derives a reliability indicator 0.6 for s_1. On the contrary, a larger ρ can lead to a more trustful reputation indicator at the cost of accommodating the QoS variations of s_1 more slowly.

Fig. 9 depicts the impact of τ, in which ρ is fixed to be 0.75. Since a larger discount interval is equivalent to a larger discount factor, we can observe the similar trend as in Fig. 8.

5.5 Load Balance with the Popularity Index

Here, we show how the popularity index technique (see Section 3.5) can help achieve load balance when the DHT is used to store reputation scores. We assume that there are 100 clients and 100 reputation score IDs, and that the redundancy index λ is 1. The popularity index contains 10 reputation score IDs whose numbers of associated reputation scores are uniformly distributed between [80, 99]. For the rest of reputation score IDs, the numbers of associated reputation scores are uniformly distributed between [10, 30]. Since the usefulness of multiple virtual IDs in improving load balance has been validated in [8], we assume that each client has one virtual ID to simplify the simulation.

Fig. 10 shows the standard deviation (STD) of the number of reputation scores per node, where each point represents the average of 150 runs. As we can see, the popularity index can obviously result in a more balanced usage of client resources: the greater the popularity branching factor ϕ, the higher the level of load balance we can achieve.

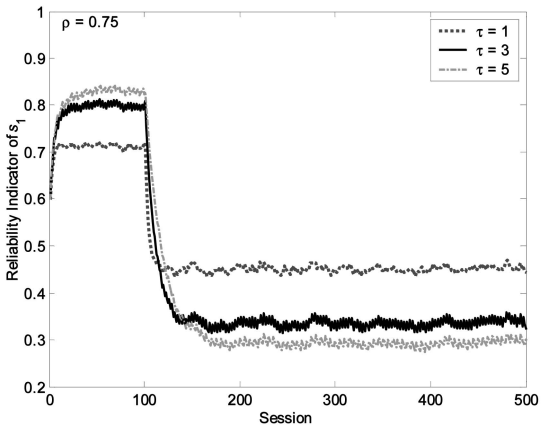


Fig. 9. Average reputation indicators for s_1 with varying discount intervals, where the discount factor ρ is fixed to be 0.75.

6 Related Work

Recent years have witnessed a growing interest in reputation systems research. Due to space constraints, we only discuss prior art that is more germane to our work and refer to [2] for a comprehensive survey.

Previous proposals [3], [4], [5], [6] use single-variate Bayesian inference [7] to build reputation engines and thus are all special cases of our Dirichlet reputation engine. Built upon multivariate Bayesian inference, our system can satisfy the diverse QoS needs of individual nodes. Our work also differs significantly from [3], [4], [5], [6] by stimulating honest participation in the reputation system by credits and social awareness. In [19], Whitby et al. propose an iterative method for filtering dishonest feedbacks, but their scheme is only applicable to the single-variate Beta reputation systems [3], [4], [5], [6]. Fernandes et al. [20] propose rewarding users for active and honest participation in the reputation system but do not consider fine-grained QoS differentiation or most of the issues presented in Section 4. Furthermore, the issue of service differentiation in P2P networks is addressed in [21], where peer reputation scores are mapped to various levels of service. By contrast, our scheme considers the QoS differentiation issue when deriving the reputation scores via multivariate Bayesian inference. Damiani et al. [22] present a reputation-based approach for choosing reliable resources in P2P file sharing applications, in which separate reputations are associated with resources and servers who share resources, respectively. This idea can help further alleviate the application-related cold-start problem when our system is applied to file-sharing-like P2P applications.

7 Conclusion

In this paper, we present a novel fine-grained reputation system to support reliable service selection in P2P networks. Firmly rooted in statistics, our system offers a theoretically sound basis for clients to choose reliable servers based on their self-experiences and peers' feedbacks on candidate servers. In addition, it can meet the diverse QoS requirements of individual nodes via a fine-grained QoS differentiation method. Our system is also application-independent and can simultaneously serve an arbitrary number of P2P applications. Moreover, we design strong incentives to motivate honest and active participation in the reputation system. We also propose various methods to ensure efficient storage and queries of users' feedbacks. Furthermore, our reputation system is designed to provide strong defense against various attacks. The effectiveness and efficiency of our system are confirmed by extensive simulation results.

As the future work, we first plan to build an experimental reputation system on PlanetLab [23] to further evaluate our design in a more realistic setting. We will also seek ways to reduce the involvement of the central server and extend our current system to a fully distributed one.

Appendix

The Pairing Technique

The pairing technique is finding growing applications in cryptography [24], [25]. Let $\mathbb{G}_1$ be an additive cyclic group of prime order q and $\mathbb{G}_2$ be a multiplicative cyclic group of the same order. Assume that the discrete logarithm problem (DLP) is hard[8] in both $\mathbb{G}_1$ and $\mathbb{G}_2$. A pairing is a *bilinear map* $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ if, for all $P, Q, R, S \in \mathbb{G}_1$, we have[9]

$$\hat{e}(P+Q,\ R+S) = \hat{e}(P,R)\hat{e}(P,S)\hat{e}(Q,R)\hat{e}(Q,S). \quad (7)$$

Modified Weil [24] and Tate [25] pairings are examples of such bilinear maps, for which the *Bilinear Diffie-Hellman Problem* (BDHP) is believed to be hard.[10] Also note that $\hat{e}$ is *symmetric*, i.e., $\hat{e}(P,Q) = \hat{e}(Q,P)$ for $\forall P,\ Q \in \mathbb{G}_1$, which follows immediately from the bilinearity and the fact that $\mathbb{G}_1$ is a cyclic group. We refer to [24], [25] for a more comprehensive description of how the pairing parameters should be chosen in practice for both efficiency and security.

Acknowledgments

This work was supported in part by the US National Science Foundation under grants CNS-0626881, DBI-0529012, and ANI-0093241 (CAREER Award).

References

[1] Open Grid Forum, http://www.ogf.org/, 2007.

[2] A. Jøsang, R. Ismail, and C. Boyd, "A Survey of Trust and Reputation Systems for Online Service Provision," *Decision Support Systems,* http://sky.fit.qut.edu.au/josang/papers/JIB2006-DSS.pdf, 2006.

[3] L. Mui, M. Mohtashemi, C. Ang, P. Szolovits, and A. Halberstadt, "Ratings in Distributed Systems: A Bayesian Approach," *Proc. 11th Workshop Information Technologies Systems,* 2001.

[4] A. Jøsang and R. Ismail, "The Beta Reputation System," *Proc. 15th Bled Electronic Commerce Conf.,* June 2002.

[5] S. Buchegger and J.-Y. Le Boudec, "A Robust Reputation System for P2P and Mobile Ad-Hoc Networks," *Proc. Second Workshop Economics of Peer-to-Peer Systems,* June 2004.

[6] S. Ganeriwal and M.B. Srivastava, "Reputation-Based Framework for High Integrity Sensor Networks," *Proc. ACM Workshop Security of Ad Hoc and Sensor Networks (SASN '04),* Oct. 2004.

[7] A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin, *Bayesian Data Analysis,* first ed. Chapman & Hall/CRC, 1995.

[8] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM Special Interest Group Data Comm. (SIGCOMM '01),* pp. 149-160, Aug. 2001.

[9] S.D. Bay and M. Schwabacher, "Mining Distance-Based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 29-38, Aug. 2003.

[10] H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication,* IETF RFC 2104, Feb. 1997.

[11] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol,* IETF RFC 4346, Apr. 2006

[12] *Digital Hash Standard, Federal Information Processing Standards Publication 180-1,* Nat'l Inst. of Standards and Technology, Apr. 1995.

[13] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured p2p Systems," *Proc. IEEE INFOCOM '04,* pp. 2253-2262, Mar. 2004.

[14] F. Angiulli and C. Pizzuti, "Outlier Mining in Large High-Dimensional Data Sets," *IEEE Trans. Knowledge Data Eng.,* vol. 17, no. 2, pp. 203-215, Feb. 2005.

[15] Y. Zhang, W. Lou, and Y. Fang, "A Secure Incentive Protocol for Mobile Ad Hoc Networks," *Wireless Networks,* to appear, available online: http://dx.doi.org/10.1007/s11276-006-6220-3.

[16] J.R. Douceur, "The Sybil Attack," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* pp. 251-260, Mar. 2002.

[17] L. von Ahn, M. Blum, N.J. Hopper, and J. Langford, "CAPTCHA: Using Hard AI Problems for Security," *Proc. Int'l Conf. Theory and Applications of Cryptology (EUROCRYPT '03),* pp. 294-311, May 2003.

[18] P. Resnick and R. Zeckhauser, "Trust among Strangers in Internet Transactions: Empirical Analysis of Ebay's Reputation System," *The Economics of the Internet and E-Commerce,* M.R. Baye, ed., Applied Microeconomics, vol. 11, Elsevier Science, 2002.

[19] A. Whitby, A. Josang, and J. Indulska, "Filtering Out Unfair Ratings in Bayesian Reputation Systems," *The Icfaian J. Management Research,* vol. 4, no. 2, pp. 48-64, Feb. 2005.

[20] A. Fernandes, E. Kotsovinos, S. Östring, and B. Dragovic, "Pinocchio: Incentives for Honest Participation in Distributed Trust Management," *Proc. Int'l Conf. Trust Management (iTrust '04),* pp. 63-77, Mar. 2004.

[21] M. Gupta and M. Ammar, "Service Differentiation in Peer-to-Peer Networks Utilizing Reputations," *Proc. ACM Fifth Int'l Workshop Networked Group Comm.,* Sept. 2003.

[22] E. Damiani, S. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks," *Proc. ACM Conf. Computer Comm. Security (CCS '02),* pp. 207-216, Nov. 2002.

[23] B.N. Chun, D.E. Culler, T. Roscoe, A.C. Bavier, L.L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: An Overlay Testbed for Broad-Coverage Services," *Proc. ACM SIGCOMM Computer Comm. Rev.,* vol. 33, no. 3, pp. 3-12, July 2003.

[24] D. Boneh and M. Franklin, "Identify-Based Encryption from the Weil Pairing," *Proc. Int'l Cryptology Conf. (CRYPTO '01),* pp. 213-229, Aug. 2001.

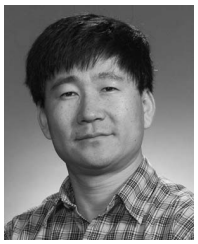
[25] P. Barreto, H. Kim, B. Bynn, and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems," *Proc. Int'l Cryptology Conf. (CRYPTO '02),* pp. 354-368, Aug. 2002.

8. It is computationally infeasible to extract the integer $x \in \mathbb{Z}_q^* = \{i | 1 \leq i \leq q-1\}$, given $P, Q \in \mathbb{G}_1$ (respectively, $P, Q \in \mathbb{G}_2$) such that $Q = xP$ (respectively, $Q = P^x$).

9. In particular, $\forall P,\ Q \in \mathbb{G}_1,\ \forall a,\ b \in \mathbb{Z}_q^*$, $\hat{e}(aP, bQ) = \hat{e}(aP, Q)^b = \hat{e}(P, bQ)^a = \hat{e}(P, Q)^{ab}$, etc.

10. It is believed that, given $< P, xP, yP, zP >$ for random $x, y, z \in \mathbb{Z}_q^*$ and $P \in \mathbb{G}_1$, there is no algorithm running in expected polynomial time, which can compute $\hat{e}(P,P)^{xyz} \in \mathbb{G}_2$ with nonnegligible probability.

Yanchao Zhang received the BE degree in computer communications from Nanjing University of Posts and Telecommunications, Nanjing, China, in July 1999, and the ME degree in computer applications from the Beijing University of Posts and Telecommunications, Beijing, China, in April 2002, and the PhD degree in electrical and computer engineering from the University of Florida, Gainesville, in August 2006. He is currently an assistant professor in the Department of Electrical and Computer Engineering at the New Jersey Institute of Technology, Newark. His research interests include network and distributed system security, wireless networking, and mobile computing. He is a member of the IEEE and the ACM.



Yuguang Fang received the PhD degree in systems, control and industrial engineering from Case Western Reserve University in January 1994 and the PhD degree in electrical engineering from Boston University in May 1997. He held a postdoctoral position in the Department of Electrical and Computer Engineering at Boston University from June 1994 to August 1995. From June 1997 to July 1998, he was a visiting assistant professor in the Department of Electrical Engineering at the University of Texas at Dallas. From July 1998 to May 2000, he was an assistant professor in the Department of Electrical and Computer Engineering at the New Jersey Institute of Technology. In May 2000, he joined the Department of Electrical and Computer Engineering at the University of Florida, Gainesville, where he received early promotion to associate professor with tenure in August 2003 and to full professor in August 2005. He holds a University of Florida Research Foundation (UFRF) Professorship from 2006 to 2009. His research interests span many areas, including wireless networks, mobile computing, mobile communications, wireless security, automatic control, and neural networks. He has published more than 100 papers in refereed professional journals and more than 100 papers in refereed professional conferences. He received the US National Science Foundation Faculty Early Career Award in 2001 and the Office of Naval Research Young Investigator Award in 2002. He was the recipient of the Best Paper Award at the IEEE International Conference on Network Protocols (ICNP) in 2006 and the recipient of the IEEE TCGN Best Paper Award at the IEEE High-Speed Networks Symposium, IEEE Globecom in 2002. Dr. Fang has actively engaged in many professional activities. He is an editor for several journals, including the *IEEE Transactions on Communications*, the *IEEE Transactions on Wireless Communications*, the *IEEE Transactions on Mobile Computing*, *ACM Wireless Networks*, and the *Journal of Computer Science and Technology*, and a technical editor for *IEEE Wireless Communications Magazine*. He was also an editor of the *IEEE Journal on Selected Areas in Communications: Wireless Communications Series*, an area editor of the *ACM Mobile Computing and Communications Review*, an editor of *Wireless Communications and Mobile Computing*, and a feature editor for "Scanning the Literature" in *IEEE Personal Communications*. He also served on the Technical Program Committee of many professional conferences, such as ACM MobiCom '02 (committee cochair for Student Travel Award), MobiCom '01, IEEE INFOCOM '08, IEEE INFOCOM '07, INFOCOM '06, INFOCOM '05 (vice-chair for technical program committee), INFOCOM '04, INFOCOM '03, INFOCOM '00, INFOCOM '98, IEEE WCNC '04, WCNC '02, WCNC '00 (technical program vice-chair), WCNC '99, IEEE Globecom '04 (symposium cochair), Globecom '02, and the International Conference on Computer Communications and Networking (IC3N, technical program vice-chair). He is a senior member of the IEEE.