The Design of A Distributed Rating Scheme for Peer-to-peer Systems

Debojyoti Dutta†, Ashish Goel‡, Ramesh Govindan†, and Hui Zhang†

Abstract— **There exist many successful examples of on-line reputation (or rating) systems, such as on-line markets and e-tailer ratings. However, for peer-to-peer applications, an explicit ratings subsystem has often been ignored in system design because of the implicit assumption of trust and altruism among P2P users. This assumption might be true (or might not matter) when a P2P network is still in its infancy and is relatively small in size. But the assumption might break down with increase in the size and diversity of the P2P network. In this paper, we discuss issues in the design of rating schemes for P2P systems. In keeping with the design philosophy of many of these system, we consider the design of *distributed* rating systems. As a case study, we illustrate two different approaches to a distributed rating system aimed at tackling the *free-rider* problem in P2P networks. A key challenge in designing such rating schemes is to make them collusion-proof: we discuss our efforts in this direction.**

I. Introduction

Rating people, goods, and services is a fundamental way to add value to entities. Rating a user of a system creates a perception that influences how users interact with each other within the system. Consider widely used rating systems such as Ebay.com, Epinions.com and Resellerratings.com. These sites thrive on providing ratings to sellers, goods, and e-stores respectively. They provide valuable advice to users based on the collective experience of other users. Such ratings (or, more generally, *reputation systems* [1]) are useful especially when there are communities of independent users each with their own preferences, and who can choose between several users to interact with.

A reputation system like that in Ebay.com is a popular approach to build trust on the Internet. Most existing examples of such systems are centralized in nature. In this paper, we posit that there will be some applications that need to have *distributed* rating schemes, for the following inter-related reasons.

1) Many systems themselves are distributed in nature. Building a centralized rating mechanism may not be compatible with the philosophy of these open distributed systems. For example, truly distributed systems promote free exchange of information at the grass-roots level, while making it difficult for authorities to monitor transactions occurring in these systems.
2) Users might prefer to have distributed systems which do not have a central point to keep track of their private actions. In a centralized system, it is easier to monitor users' activities. However, in a distributed system, no single user will be aware of all the transactions of another user. The absence of a central server avoids a central point of failure.

In this paper, we discuss issues that arise in the design of distributed reputation schemes (Section II). We present (Section III) efficient techniques based on randomization that can be used to design efficient distributed mechanisms for rating users.

We then show how some of these techniques can be used to design distributed reputation systems that improve the performance and scalability of peer-to-peer (P2P) systems. During the last few years, P2P file sharing applications such as Napster, Gnutella, KaZaA, Freenet, and Morpheus have gained tremendous popularity in the Internet. P2P systems are classified either as *unstructured* (such as Napster, Gnutella, and Freenet) or as *structured* (such as Distributed Hash Table (DHT) systems [2], [3]) based on the type of the overlay topologies they create.

We focus on a distributed rating scheme for tackling the *free-rider* phenomenon that has been observed in P2P systems such as Napster and Gnutella [4], [5] . We present (Section IV) one distributed rating mechanism along with two different ratings validation schemes. In the first validation scheme, we present a Structured Verification Scheme (SVS) which relies on a supervisory overlay. The second validation scheme is a Lightweight Unstructured Verification Scheme (LUVS) that uses a simple random sampling technique, and does not require

†Department of Computer Science, University of Southern California. {ddutta, ramesh, huizhang}@usc.edu.

‡Department of Management Science and Engineering and (by courtesy) Computer Science, Stanford University. ashishg@stanford.edu.

Ashish Goel, Hui Zhang, and Ramesh Govindan's research was supported in part by NSF grant CCR0126347. Goel's research was also supported in part by NSF Career grant No. 0133968.

a structured overlay. Finally, we note that a key challenge in designing such rating schemes is to make them collusion-proof: we discuss our efforts in Section V.

II. Problem Statement

One of the objectives of any open economic community is to maximize the quality of service QoS of different entities (*users*). This might translate into more file downloads or fast data transfers in a P2P system, for example, or might translate into better customer service provided by e-tailers. Thus, the performance of the whole system will be better when more users participate actively by providing better services to others.

One basic characteristic of any open economic system is that users are given the freedom to decide how much they want to contribute to the system. Users might choose not to cooperate. Thus, users can be classified as well-behaved and misbehaving. Misbehaving users can be further divided into

- Selfish (Greedy): Selfish users are interested in their own benefit. The moment they get less *utility*, they change their actions or *strategies*. Systems are called *strategy-proof* if they are resistant to single user misbehaving. The users might also *collude* in order to satisfy their individual selfish needs. Systems that are resistant to collusion are called *group strategy-proof.*
- Adversarial (Malicious): Adversarial users are not bothered about their own good. Their only goal is to subvert the system. Subverting the system in a P2P context may include the introduction of worms or fake files, for example.

Even though ratings systems rely on the altruism of users to rate their interactions with other users, we must expect that any rating system will need to protect itself against selfish and malicious users.

A. The Free-Rider Phenomenon

One practical problem that brings out the need for a distributed rating system is the *free-rider* phenomenon observed in file-sharing P2P systems. This phenomenon has the following characteristics:

- Most files (*e.g.*, 98% reported in [4]) belong to a small percentage of the users (20%, respectively). The ratio of the number of queries to the number of users that respond to queries is similar. Hence, the quality and availability of the system is dependent on a small subset of the users.
- A majority of the users are “one-time, one-hour” users. They just download files and do not make their resources available to others. Thus, many users do not really participate in the network, which goes against the basic premise of P2P systems.

The prevalence of free-riders might make it much harder to deploy large, robust P2P systems Also, if there were only a few good users, a P2P system might effectively become clustered and centralized (in a loose sense), which leads to the possibility of a system collapse and increased vulnerability.

Clearly, a rating scheme whereby a user is given a level of service that is concomitant with his or her participation in the P2P system can help alleviate the free-rider problem. As we have argued before, such a rating scheme needs to be distributed in keeping with the grass-roots nature of such systems. Before discussing a distributed ratings scheme for P2P systems, we list some general issues in the design of such schemes.

III. Solution Space

In this section, we discuss the basic design issues of building a reputation system in a distributed fashion. We address the first two of the following questions:

1) What kind of rating information do we use?
2) How do we manage rating information in distributed fashion?
3) Is our system resilient to greed?
4) Is our system resilient to malice?

deferring to Section V a discussion of the latter two.

A. Rating

The first component of a rating based architecture is the property of the rating information itself. This is a well-studied issue [6]. Here, we briefly discuss the different options for ratings:

- Positive vs. Negative: Positive rating quantitatively describes the services a user has provided to the community. When there is no negative rating and users start at zero, there is no incentive for these users to change their names or identifiers. See [7] for a discussion of the social cost of cheap pseudonyms.
  A user might be assigned negative ratings for bad or fraudulent transactions. We might want to reprimand users in such cases. This is also a way to detect and isolate misbehaving users. But negative rating alone is not effective when malicious users can change their identities.
  Both positive rating and negative rating have their own design goals, and might be needed to handle different types of users in a practical reputation system.

- Continuous vs. Steps: The raw rating value of a user may represent the accumulation of feedback from other users, and, hence, may be represented by an integer variable that increases monotonically. Instead of using it directly, we can also use rating values based on steps. For example, a 4-level positive rating system can be defined where the difference between each level would require a user to get an order of magnitude more recommendations from other users. We will show later that such discrete rating system is useful when handling collusions.
- Full window vs. Sliding window: We can either consider the complete history of a user's transaction or use ratings received in the recent past (sliding window). A sliding window motivates users to contribute to the community continuously. Also, ratings based on sliding windows can alleviate the effect of user dynamics (users joining/leaving) on the rating verification techniques discussed later.

B. Managing Rating Information without a Central Server

We now consider possible approaches to answer the second question. Clearly, there are two general approaches we could take. One is self-maintaining, the other is supervisor-based.

- Self-maintaining: In this approach, each user maintains all the related information about its own rating locally. Whenever a user x wants to know another user y's rating information, x asks y directly. This is an efficient approach since all information is stored locally at y. Also, users are willing to keep track of their own positive rating information for their own benefit. However, this approach has two problems. First, a user might exaggerate its positive rating if it has the complete control of his rating information. One solution to this is to verify y's rating information by randomly sampling a few of the users y claims to have served, which in turn requires each user to record a list of servers from which it has received services successfully. The second problem is that a user is unwilling to record any negative information for itself. This could be solved by complicated monitoring scheme, or just maintaining negative information at other users, like in the following approach.
- Supervisor-based: In this approach, each user x's behavior will be monitored by a few other users, called its supervisors. Any message related to x's rating is directed to those supervisors. To make this approach robust to malicious users and collusions, all decisions that the group of supervisors takes are subjected to voting. The supervisors are chosen so that they appear to be some random users to the supervised user. This will dissuade cheating. In addition, there should be no small loops in the supervisory graph in order to prevent the formation of small groups that control its own ratings.

IV. A Distributed Rating Scheme to Incentivize Cooperation

In this section, we present a positive rating based distributed rating scheme for the free-rider problem.

A. A Verification-based Rating Scheme

Assume a network of n selfish users. We place an important restriction: all users must support the basic routing operations to keep the system running in a normal state. Therefore, a request might fail due to the lack of the resources it asked for, but it should not fail because it is discarded in the middle of a routing operation. This restriction is reasonable, and has been observed in the real P2P systems by past studies of measurements [5]. We also assume that the routing operations are not compromised by the users.

Each user will rate other users based on the service it receives from them and will store its own ratings. This service includes factors like the number of successful requests it makes, response delay, and download speed of the file transferring transaction, *etc.* The rating R_i of user u_i is defined by the set of all users it served within some sliding window. For each satisfied request r from a certain user x, R_i will be increased by a certain value. Therefore, the more actively a user participates in the system, the more requests it should satisfy, and the higher its R_i should be. We believe that by giving better services to the users with higher rating information, we introduce incentives for cooperation.

In a generic description of our distributed rating scheme, each requesting user u_i will claim a rating value R'_i and encapsulate it in the query message. The responder u_j will decide how to satisfy the request with the claimed rating R'_i. The exact mechanism for satisfying a user's request based on its rating is somewhat application dependent. For example, u_j could keep a single queue and order all requests according to the rating values. Or u_j could provide differentiated service (*e.g.*, withhold some information, or return only the initial part of a music file) to users with lower ratings. However, a key problem with our generic description is: how can u_j *verify* u_i's rating? We now discuss two kinds of ratings verification schemes.

B. Structured Verification Scheme

In the structured scheme, each user has k (random) designated supervisors. After every transaction, the user u_i's supervisors will update the rating value R_i to an appropriate value.

We break down the verification problem into two subproblems:

- How do we set up the supervising network in a distributed fashion?
- How do we perform cheat-proof supervising?

B.1 Supervising overlay – As mentioned in Section III-B, the supervising network should not contain small loops in order to dissuade collusion. Many graphs satisfy the above condition, such as rings, binary trees and star topologies. We choose the ring as the overlay topology where each user supervises its k immediate successors. For simplicity, we choose $k = 2$; thus each user i supervises its two immediate successors on the ring. i is the main supervisor for its immediate successor only. For its successor's successor, i works as a backup supervisor. All reputation related transactions happen only between a user and its main supervisor, and a backup supervisor periodically refreshes the rating information from the corresponding main supervisor. However, we can always choose $k > 2$ and give equal responsibility to all supervisors when considering malice.

Chord [3] is an ideal infrastructure for setting up and maintaining such ring based supervisory overlays. Chord organizes users on a ring, and each user is mapped into a fixed point on the ring by hashing its IP address. Therefore, two neighbors in Chord may not have any relation with each other. Thus, there should be no relationship between a user and its supervisor.

B.2 Cheat-proof supervising – Suppose a user x sends out a request which can be satisfied by the user y. Let $Super(y)$ denote the main supervisor of user y. Additional supervising operations incorporated into the regular request operation will be one of the following:

- After y receives x's request, and it decides to accept the request, it sends a message to x, and notifies $Super(y)$ of the ongoing transaction with x.
- After the transaction is over, x notifies $Super(y)$ (by following Chord protocol) to update the rate R_y with some transaction information. A three-way hand-shaking procedure is needed to confirm x's ID.

Clearly, the above operations will not add any extra latency to the regular request operations as are done after the transaction is over.

B.3 Rating verification – With the Chord infrastructure, the verification of the rating information becomes simple. When user y needs to verify x's rating information, it gets x's ID on the ring topology by hashing its IP address, and sends a rating query into the supervising overlay with x's ID as the target key. The Chord protocol guarantees that this query will never be intercepted by x itself, and will take $O(logN)$ hops to deliver to x's main supervisor, which implies that it will be hard for a user to indulge in a small-group collusion.

C. Lightweight Unstructured Verification Scheme

In this section, we propose a lightweight Unstructured Verification Scheme (LUVS). LUVS does not require an additional supervising overlay and the verification is based on a simple random sampling technique.

C.1. Random sampling based verification – The basic idea in LUVS is simple. In LUVS, each user will keep a list of the customers she has served and a list of servers from whom she has been served, along with the details of the transactions. Now, when a user x wants some service from a user y, x sends the list of its customers along with the request. If user y decides to verify x's rating, it samples a subset of x's customer list to conform if they have received the claimed service from x or not. If most members in this sample say a yes, y trusts x and provides service depending on the rating of x.

The cheat-proof properties of LUVS is due to random sampling. However, the above scheme raises privacy concerns due to the disclosure of the customer list (certificate). Besides, the communication cost associated with transfer of the customer list could be high. Instead of transferring the complete customer list, our LUVS scheme incurs little communication cost to randomly sample the customers in the requesting user's customer list.

C.2. Randomly sampling without accessing the complete customer list – Assume a is the requesting user, and b is the server. Now b wants to verify a's rating information by randomly sampling a's customers.

- Onto a one-dimensional key space $0 - (2^k - 1)$(say $k = 64$), a hashes each of its customers onto a point based on their IP addresses. The customers will be distributed randomly and evenly onto this space.
- The key space is divided into M (say $M = 256$) equal bins. a counts the number of customers hashed into each bin, and records those numbers in a M-entry vector.
- a sends the above vector to b, with a constant communication cost $M * 4$ bytes (1KB for $M = 256$) despite the real size of its customer list.

- b randomly picks one or several bins with the probability proportional to the number of customers in that bin, and asks a to send the detailed information of those customers in the chosen bins.
- b then follows the rest of the steps described in the previous section.

V. Collusion

Clearly, the above two verification schemes can easily detect those non-cooperative users when they lie. Thus they are strategy-proof. However, they cannot avoid collusion. There is because with positive rating alone, a user can give credit to other users effortlessly. Now we discuss two techniques which alleviate the collusion problem.

A. Discrete Rating

If we choose the four-star rating system as mentioned in Section III-A, a clique or group of lying users has to have enough members before they can benefit from the collusion. The dynamic characteristic of P2P users and the high cost or effort of organizing a large colluding group might prohibit such behavior.

B. Negative Rating

Here a negative rating is defined as the *payment* a user makes for the service it receives. If we use negative ratings along with positive ratings, selfish users will have no incentives to give credits to others, and collusion is discouraged in a group of selfish users since the net gain of the group rating (summation of the rating for all group members) will be equal to or below 0 due to the collusion.

This can be easily implemented in our proposed structured rating scheme. The additional operation after a successful request is to notify the requestor's supervisors so that its rating will be reduced accordingly.

In our unstructured rating scheme, a naive implementation of the above idea is to design the rule that a user x can claim another user y as its customer only if x has given more service to y than y to x. Then x and y cannot be in each other's customer at the same time. But simple colluding schemes can defeat this rule. For example, a colluding group can be organized into three ordered subgroups (e.g., $G_A \rightarrow G_B \rightarrow G_C \rightarrow G_A$) so that a member in a subgroup may claim all members in the next subgroup as its customers. Improving the design of our unstructured rating scheme to be collusion-proof is our ongoing work.

C. Non-selfish collusion

In a non-selfish collusion, a group tries to act in such a way such that at least one or a few of the members can benefit from the collusion. One example of a non-selfish collusion is that of a single physical user generating multiple virtual IDs (users) such that at least one of his IDs gets higher rating. Another example of non-selfish collusion is that of a group of adversarial users teaming up to subvert the system, and not bother about their individual reputation.

The first thing to note is that collusion will not benefit from an individual's negative rating. A user can only get rid of its bad reputation by changing its ID. However, when the reputation system is equipped with both positive rating and negative rating, and biases new users based on positive rating, it is always costly to commit a successful malicious attack.

We must also note that it is hard to detect non-selfish collusions efficiently. In a P2P network, when user x claims it has successfully downloaded a file from y, it's difficult to determine whether there indeed was such a download. Even if we could detect this download (e.g., through proxy forwarding/monitoring), it would be difficult to determine whether this was normal action or not.

VI. Related Work

A quick introduction to ratings is nicely presented in [1]. In [8], the authors present schemes to calculate reputation vectors similar to those in [9]. However, [8] does not study whether the methods are cheat-proof, or whether the scheme can be distributed.

P2P architectures have been extensively studied in the recent past. Example systems are Napster, Gnutella (unstructured) and CAN [2] and Chord [3] (structured). A survey of P2P research is beyond the scope of this paper.

There has been a recent trend toward incentive compatible distributed system design [10]. In [11], the authors present a concise survey of the use of incentives to design better wireless ad-hoc networks. In [9], the authors present a reputation based scheme to isolate malicious content providers in P2P systems. However, their scheme's protection from collusion relies on a set of well-known pre-trusted users, and the calculation of global reputation values requires synchronization of the whole network. Besides, it's not clear how the reputation calculation algorithm might scale with the network size.

VII. Conclusions and Future Work

In this paper, we have discussed general design issues involved in building a distributed rating scheme.

These schemes have wide applicability. For a specific problem in P2P systems, the free-rider problem, we have presented a positive rating based reputation system along with two distinct mechanisms to validate the rating information. Our current direction is to design efficient distributed rating schemes that are collusion free.

REFERENCES

[1] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, "Reputation systems," *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, 2000.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *ACM SIGCOMM*, 2001.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for Internet applications," in *ACM SIGCOMM*, 2001.

[4] E. Adar and B. Huberman, "Free riding on gnutella," in *First Monday 5*,Oct. 2000.

[5] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," *Tech. Report UW-CSE-01-06-02, University of Washington*, 2001.

[6] P. Kollock, "The production of trust in online markets," *Advances in Group Processes (Vol. 16), edited by E. J. Lawler, M. Macy, S. Thyne, and H. A. Walker. Greenwich, CT: JAI Press.*, 1999.

[7] E. Friedman and P. Resnick, "The social cost of cheap pseudonyms," in *Journal of Economics and Management Strategy*, 10(2):173-199. 2001.

[8] G. Zacharia, A. Moukas, and P. Maes, "Collaborative reputation mechanisms in electronic marketplaces," in *HICSS*, 1999.

[9] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The EigenTrust algorithm for reputation management in P2P networks," to appear in *the 12th WWW conference*, 2003.

[10] J. Feigenbaum and S. Shenker, "Distributed algorithmic mechanism design: Recent results and future directions," in *the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, ACM Press, New York, 2002.

[11] P. Obreiter, B. Koenig-Ries, and M. Klein, "Stimulating cooperative behavior of autonomous devices - an analysis of requirements and existing approaches," in *Second International Workshop on Wireless Information Systems (WIS2003)*, 2003.