5-10-2009

An Evaluation Framework for Reputation Management Systems

Andrew G. West
University of Pennsylvania, westand@cis.upenn.edu

Sampath Kannan
University of Pennsylvania, kannan@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

An Evaluation Framework for Reputation Management Systems

Abstract
Reputation management (RM) is employed in distributed and peer-to-peer networks to help users compute a measure of trust in other users based on initial belief, observed behavior, and run-time feedback. These trust values influence how, or with whom, a user will interact. Existing literature on RM focuses primarily on algorithm development, not comparative analysis. To remedy this, we propose an evaluation framework based on the trace-simulator paradigm. Trace file generation emulates a variety of network configurations, and particular attention is given to modeling malicious user behavior. Simulation is trace-based and incremental trust calculation techniques are developed to allow experimentation with networks of substantial size. The described framework is available as open source so that researchers can evaluate the effectiveness of other reputation management techniques and/or extend functionality.

This chapter reports on our framework's design decisions. Our goal being to build a general-purpose simulator, we have the opportunity to characterize the breadth of existing RM systems. Further, we demonstrate our tool using two reputation algorithms (EigenTrust and a modified TNA-SL) under varied network conditions. Our analysis permits us to make claims about the algorithms' comparative merits. We conclude that such systems, assuming their distribution is secure, are highly effective at managing trust, even against adversarial collectives.

Keywords
reputation management, reputation algorithm, EigenTrust, TNA-SL, a priori trust, transitive trust, malicious collective, bandwidth throttling, feedback, peer-to-peer network, decentralized topology, trust management, network trace, Zipf distribution

An Evaluation Framework for Reputation Management Systems*

Andrew G. West
westand@cis.upenn.edu
University of Pennsylvania, USA

Sampath Kannan
kannan@cis.upenn.edu
University of Pennsylvania, USA

Insup Lee
lee@cis.upenn.edu
University of Pennsylvania, USA

Oleg Sokolsky
sokolsky@cis.upenn.edu
University of Pennsylvania, USA

ABSTRACT

Reputation management (RM) is employed in distributed and peer-to-peer networks to help users compute a measure of trust in other users based on initial belief, observed behavior, and run-time feedback. These trust values influence how, or with whom, a user will interact. Existing literature on RM focuses primarily on algorithm development, not comparative analysis. To remedy this, we propose an evaluation framework based on the trace-simulator paradigm. Trace file generation emulates a variety of network configurations, and particular attention is given to modeling malicious user behavior. Simulation is trace-based and incremental trust calculation techniques are developed to allow experimentation with networks of substantial size. The described framework is available as open source so that researchers can evaluate the effectiveness of other reputation management techniques and/or extend functionality.

This chapter reports on our framework's design decisions. Our goal being to build a general-purpose simulator, we have the opportunity to characterize the breadth of existing RM systems. Further, we demonstrate our tool using two reputation algorithms (EigenTrust and a modified TNA-SL) under varied network conditions. Our analysis permits us to make claims about the algorithms' comparative merits. We conclude that such systems, assuming their distribution is secure, are highly effective at managing trust, even against adversarial collectives.

Keywords: reputation management; reputation algorithm; EigenTrust; TNA-SL; a priori trust; transitive trust; malicious collective; bandwidth throttling; feedback; peer-to-peer network; decentralized topology; trust management; network trace; Zipf distribution.

INTRODUCTION

At the start of the network-age the client-server (centralized) model was the dominant topology. Trust in these servers was implicit and security measures focused on access control and user permissions. More recently, new network architectures and computing paradigms have emerged such as distributed systems, peer-to-peer (P2P) networks, and ad-hoc mobile computing.

Frequently, all network nodes have the ability to both request services from *and* provide services to other users. This is inherently risky since decentralized models typically lack the notions of authenticity, reliability, and accountability that monolithic servers provide. Nonetheless, well-behaved decentralized systems are beneficial in comparison to their client-server counterparts. Advantages include increased service diversity, availability, scalability, and bandwidth.

Enforcing good behavior is the task of a *trust management* (TM) system. The seminal work of Blaze, Feigenbaum, and Lacy (1996) introduced the term -- their system consisted of using cryptographically delegated credentials and policies to specify static access control rights. In *reputation management*[1] (RM), rather than determining if a user has the authority/permission to do some action, we instead ask: Given permission, how do we expect a user to behave (*i.e.*, what is his/her reputation)? A systems treatment of these expectations gives rise to a *dynamic* access control mechanism which is categorically different than that provided by TM. Reputation management is implemented by a *RM system* or *reputation algorithm*[2] (RA).

Almost universally, RAs work by using past behavior as a basis for future conduct. Transitive trust is often exploited, especially in the absence of prior interaction between two parties. To promote a well-intentioned network either bad behavior is punished, good behavior rewarded, or both. EigenTrust (Kamvar, Schlosser, & Garcia-molina, 2003) and Trust Network Analysis with Subjective Logic (TNA-SL) (Jøsang, 2001; Jøsang, Hayward, & Pope, 2006) are two RAs that will be given particular attention herein. For a broader survey of available systems, readers should review the work of Li and Singhai (2007). One should note that the need for RM is not confined to purely digital dealings. In fact, eBay manages one of the largest RM systems (Resnick, & Zeckhauser, 2001), pertaining to the exchange of physical commodities.

Research concerning RM has been focused on algorithm development with little attention given to *quantitative* comparative analysis between existing RAs (*qualitative* analyses are often seen, but we feel, insufficient). Tests on some systems, like EigenTrust, use briefly-described, proprietary, or closed-source simulators (Schlosser, Condie, & Kamvar, 2003). Others, like TNA-SL, opt for a more theoretical description with no evaluation results. In order to compare systems such as these and verify author's claims, an objective simulator is needed. While network and P2P simulators exist, having the additional overheard of simulating Distributed Hash Tables (DHTs), latency, network hops, *etc.*, in addition to trust calculation make their use computationally inappropriate. Furthermore, such simulators offer little abstraction, making the implementation of RAs inconvenient. Therefore, in this chapter we describe the construction of an evaluation framework specific to reputation management.

This chapter is organized as follows: We will begin by standardizing terminology and justifying our architecture of choice. Trace generation and simulation under this architecture will then be discussed. Next, evaluation metrics with regard to effectiveness and efficiency will be introduced. Then, test runs will be used to exemplify behavior and identify potential shortcomings in our design. Finally, future work will be noted and concluding remarks made.

OVERVIEW OF EVALUATION FRAMEWORK

There are many challenges in building a general-purpose evaluation framework for RM systems. First, RM is used in a variety of network architectures such as, peer-to-peer (P2P), service-oriented (SOA), and social networks. Second, there is a tension between simulating realistic behavior and excessive parameterization. We must find a tractable compromise that yields

accurate trust values. Third, it is a challenge to define user behavior models, especially those pertaining to malicious users. While good users behave in a predictable manner, malicious users, especially those acting in a collective manner, may behave in erratic and dynamic ways.

In this chapter we will address precisely these challenges. Though many approaches are possible, we will present one that our research and intuition has found most appropriate – while still giving attention to alternative strategies.

Terminology

We consider systems that consist of *users*, *nodes*, or *peers*. These users are part of a *network*. Certain pairs of users have a communication channel between them. In graph theoretic terms, users are the nodes and these channels are the edges. The graph need not be connected. At any time, a user may be acting as a *provider* (server), a *requester* (client), or both. The items being requested and traded are termed *files* but could be representative of services or remote-procedure calls. Files are either *valid* or *invalid* and we assume this is a determination users can make accurately. A file's validity is permanent, and multiple copies of the same file can exist on a network. Users are evaluated according to the quality of files they provide to other users. Files are stored in a user's *library* and a user enters the network with an *initial library*. Broadcasting to determine the set of owners who possess some file is termed a *query* and the actual request and acquisition a *transaction*. Following each transaction, binary[3] *feedback* is expected. Finally, a user is able to remove files from his/her library, an action we term *clean-up*.

Architectural Justification

We now describe the infrastructure on which our implementation is based. As our terminology may have hinted, we believe an underlying P2P network (*e.g.*, a file trading network, akin to Gnutella) is most appropriate because of its expressiveness. Such an architecture is capable of emulating several other system models by limiting functionality appropriately.

For example, suppose one wants to test a system with a mutually exclusive set of clients and service providers where clients use RM to determine provider reliability. With only slight modification a P2P framework can simulate such a model. The *files* being traded become representative of *services*. Service providers enter a network with an initial library (of services) but never make requests. Clients enter the network with an empty initial library, request and receive services, but *clean-up* (do not store) every service they receive. By cleaning up, clients prevent themselves from becoming providers. *Feedback* can then encode not only if services are valid or invalid, but also a measure of the quality of service (QoS) provided. Other emulation scenarios are possible but not described here. For a survey of reputation strategies grouped by application-domain, see the work of Zhang, Yu, and Irwin (2004).

Comparability of reputation algorithms is realized via our trace-simulator approach. Crucially, traces are *static* in nature; they are pre-generated and not modified during the simulation phase. The generation of trace files encodes network parameters and is independent of the RA being tested. These traces are then given to the simulator, which implements the RA, and other, minimal, dynamic considerations. Thus, multiple simulations, implementing different RAs, can be run from the same trace. Figure 1 shows this overall architecture.

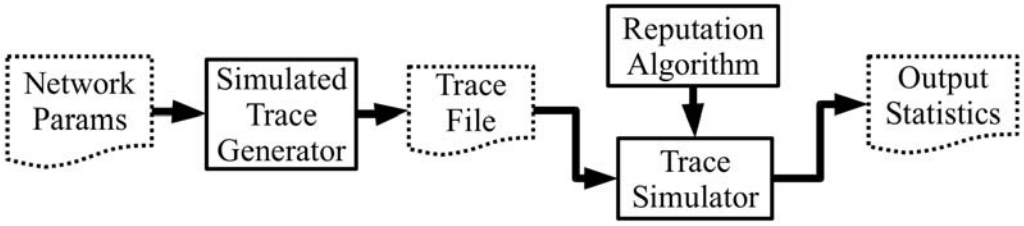

Figure 1: Overview of evaluation architecture

By fixing every aspect of a network run except those specific to RAs, the only differences between runs will be reputation-algorithm specific – ideal for comparison. Adherence to the static constraint does present some problems, as we will later describe. Nonetheless, comparability should not be sacrificed as it is precisely the motivation/objective of this work.

TRACE GENERATION

The trace generator is a program that takes network parameters and outputs a static script of a network run. There is a tension between simplicity and modeling realistic behavior. On one hand, a rich feature set can model the most complex and subtle of behaviors. On the other, excessive parameterization might obfuscate results or make it difficult to derive useful statistical inferences. Also, such exacting detail may be unnecessary for comparing the merits of different RM methods. We opt for a reasonably simple design and describe it in depth below.

Trace Generation Summary

In our generations/simulations we model a network of users. We have a library of user models, some corresponding to *good* users and others to *malicious* ones. Input parameters tell us how many users to pick per model. In this paper we assume there is a link between every pair of users. Each user is also given an initial endowment of files, some *valid* and others *invalid*. There can be many copies of any file. The initial file distribution is also governed by input parameters.

Generally speaking, a trace is a sequence of queries. Each query specifies a filename and the user seeking that file (the client). The choice of server from which to download the file will be made at simulation runtime, taking into account input from the reputation algorithm.

Users have behavioral choices along two dimensions. First, for each file they download they can choose to clean-up, *i.e.*, get rid of the file. Good users will tend to clean up invalid files with high probability to reduce the proliferation of bad files. Malicious users may do just the opposite. We assume that no clean-up happens on the initial endowment and that users have just one opportunity to clean-up a file -- immediately after they download it. Second, after each transaction the client provides universally-observable feedback regarding the server with respect to the nature of the file served. Good users will tend to provide honest feedback, submitting positive marks if they received a valid file. Malicious users may tend to do the reverse.

To account for the rich variety of strategies and motives for adversarial behavior we have many malicious user models. These models vary with respect to clean-up and honest-feedback probabilities. We also allow for more subtle malicious behaviors where users behave badly only

some of the time. Malicious collusion is a complex topic reserved for a later section. We also make the simplifying assumption that malicious behavior is purely probabilistic and independent of the client or file requested in a particular transaction. We now proceed with a more in-depth discussion of the trace generation process, beginning with the physical file format.

Trace Files

Our goal is to use the minimum parameterization necessary to realistically exercise the RM systems being evaluated. Given at the command-line, the most critical arguments are:

1. Number of users in network
2. Behavior model for each user
3. Number of distinct files
4. Probability of file ownership
5. Number of queries/transactions to simulate
6. Maximum number of user connections
7. Bandwidth period, in time units

These topics will be covered in greater depth in their respective sections. The output of trace generation is a terse text file with four distinct types of data:

1. Header: The command line arguments are printed. The simulator needs these to size data structures so they are provided upfront.

2. User Initializations: Triples of the form (u, c, h) where u is the user being initialized, c is the percentage of the time the user removes an *invalid* file from his/her library (clean-up), and h is the percentage of the time the user provides honest feedback. Derived from this tuple, inverse clean-up, (100 - c), is the percentage of the time a user removes a *valid* file from his/her library.

3. Library Initializations: Triples of the form (u, f, v) that state that user u has file f in his/her initial library with validity v (a Boolean).

4. Static Queries: Pairs of form (u, f) stating user u wishes to obtain file f.

User Models

As alluded to in the previous section, there are only two dimensions of user behavior, the feedback-honesty and clean-up rates. Table 1 describes the initializations of several user models. Our selection of user models is not comprehensive. Additional user models can be imagined and easily implemented in our framework.

User Type	Clean-up%	Honesty%
Good	90-100%	100%
Purely Malicious	0-10%	0%
Malicious Provider	0-10%	100%
Feedback Malicious	90-100%	0%
Disguised Malicious	50-100%	50-100%
Sybil Attacker	0-10%	Irrelevant

Table 1: User initialization parameters

It should be noted that several of these models were inspired by the threats detailed by Kamvar *et al.* (2003) and Hoffman, Zage, & Nita-Rotaru (2008). Because clean-up is a passive action, clean-up rates are expressed as a range – a precise percentage assignment is made when a user is initialized. Feedback, meanwhile, is required following each transaction so we expect that most user models will provide feedback according to a strict pattern.

We should pause to consider the motivations behind user behavior. A good user simply wants to *obtain* a valid copy of the file they query for. Bad users want to *deliver* invalid files. The notion of *invalidity* will vary based on the intentions of the malicious user. Files that are corrupt or contain viruses will be traded by users who wish to instigate havoc, as these behaviors are not self-serving. However, the intentional mislabeling and distribution of a music file as a guerilla advertising tactic could constitute a selfish form of invalidity. Stepping outside of the P2P realm, manipulation of RM systems can prove much more profitable. Subverting eBay's RA could have large monetary consequences. We now describe our user models in detail:

- Good: Initializing well-behaved users is straightforward. First, they will provide honest feedback. Secondly, they will be attentive about their file libraries, removing invalid files that reside there. A clean-up rate between 90-100% permits some degree of apathy, as we cannot expect *good* users to be ideal ones.

- Purely Malicious: A user misbehaving in both dimensions is termed *purely malicious*. Such users retain invalid files, dispose of valid ones, and consistently lie about the nature of the files they receive. Because they misbehave with such consistency a RA may quickly identify such users and take preventive measures.

- Single Dimension Malicious: *Malicious providers* and *feedback malicious* are complementary user models that misbehave along a single axis. Such strategies can be devastating in systems where trust is evaluated along only one dimension. For example, providing dishonest feedback is not of first order consequence (*i.e.*, liars are not punished) in a system like EigenTrust. Feedback dishonesty seems particularly hard to detect because systems rely on non-automated and user-provided *soft-feedback*[4]. The deficiencies and game-theoretic strategies of soft-feedback are well investigated by Resnick and Zeckhauser (2001) in their analysis of the eBay system. The DIRECT (Zhang, Lin, & Klefstad, 2006) and TrustGuard (Srivatsa, Xiong, & Liu, 2005) approaches attempt to discover outlying feedback patterns but require continuous feedback variables. However, the *feedback-malicious* model that exploits this weakness does not directly benefit the user implementing it. Because the *feedback-malicious* user

maintains a valid library he/she cannot distribute the invalid files of his/her choice (*i.e.*, the motive). Instead, by exploiting transitive trust such users can act as a gateway, using the trust others likely have in them to direct users to invalid libraries. This is a malicious approach we later validate in our description of cooperative strategies.

In contrast, effective strategies involving *malicious providers* are harder to envision. Trust can still be managed in networks overrun with such users because they are honest about where bad files reside. Thus, it is trivial to avoid downloading from such providers (for those who desire to do so). Nonetheless, the model is included for completeness.

- Disguised Malicious: In the absence of a sophisticated collective strategy (see below), users consistently distributing invalid files will likely be identified as such. Often, this means they will be sent a smaller amount of request traffic from *good* users and be unable to distribute their files. An effective strategy may be to act well-behaved slightly more often than bad. This way, systems that rely on normalization and casual relationships between positive and negative feedbacks will think of them as (weakly) *good* users. This is the strategy of the *disguised malicious* user. Some systems, like TNA-SL combat this by using beta-PDF strategies that consider the raw number of feedbacks.

- Sybil Attacker: Based on the model described by Douceur (2002), the *Sybil attacker* takes advantage of the low-entry threshold present in most networks. With an invalid library, the attacker waits until he/she is a provider in some transaction, deletes his/her account, and then re-appears on the network under a different username. The neutral trust associated with the new account is preferable to the (presumably) negative trust of the old one. For implementation purposes we simply prohibit feedback from being recorded by Sybil users, or concerning them. This prevents expansive network growth from disposable 'single-use' users. Having the *Sybil attacker* model could help analyze the effectiveness of RM systems that give preference to experienced users.

- Malicious Collectives: A group of cooperating malicious users presents the most severe threat to a decentralized system. Isolated malicious users participate in *casual* cooperation, for example, when they provide positive feedback to an anonymous user that sent them an invalid file. However, organized *tight* cooperation, characterized by intelligent and peer-aware strategies, is the real danger. RM systems deter this threat by using anonymous identifiers and DHTs with local voting to blur the network topology. Thus, such users may have an out-of-band communication method to coordinate their activities. Effectively modeling such behavior is a topic given considerable attention in a later section entitled 'Empowering Malicious Collectives'.

Of course, the raw number of users in a trace is dependent on the needs of whoever is generating it. Simulating larger networks will produce more consistent results, while sacrificing efficiency. Similarly, the number of users per model will vary depending on the trust scenario being tested. We can, however, offer some guidance in this selection. We should assume we are modeling a network organized with good intentions. Malicious users rarely create networks; they try to infiltrate existing ones[5]. Therefore, we wish to study an infection model and invalid file propagation. Furthermore, no RA will be able to revive a network that is severely compromised

from the outset. For these reasons, individuals should not generate traces that are initially amok with invalidity, *i.e.*, with an excess of malicious peers.

Library Initialization

Library composition is modeled as a Zipf distribution (Zipf, 1949). Many studies have been completed, including that of Breslau, Cao, Fan, Phillips, and Shenker (1999) that demonstrate Zipf frequencies accurately model file/service popularity in many Internet domains. Zipf parameter α is provided at the command line such that file *i* has a $(1 / i^{\alpha})$ probability of being owned by a particular user[6]. The validity of an initial file copy is determined by the clean-up-rate of the user who owns it. For example, a user with cleanup-rate *c* has a *c* percent chance of an initial file being valid. Clean-up is *not* performed during initial library construction.

Under such a distribution every user will have an initial library of identical expected size. This is an unrealistic assumption but one we are willing to make for the sake of simplicity and because we believe that it will not affect our empirical results significantly.

Choosing a good α value is a challenge. Arguments can be made for both a high and low α value. On one hand, studies like that of Breslau *et al.* (1999) suggest α=0.8 or higher is the best reflection of real-life data patterns. On the other, such a high α value may be problematic. If one wants to 'schedule' a large number of queries/transactions there must be sufficient resources such that every user does not acquire every file. A high α value makes initial libraries small and non-diverse (due to the Zipf distribution's tail), exacerbating such a problem. There are three remedies. First, more users can be added, but this creates an acute efficiency problem. Second, the file set can be made larger. However, the nature of Zipf's law is such that the file set would need a substantial increase, and substantial memory footprint, to make user's libraries marginally more diverse. Third, α can be decreased. We prefer this course of action and default to α=0.4, a value simulation has shown to be a good compromise.

Query Generation

The final consideration is who should be requesting files, which files, and in what quantity? Our generator supports two modes, which we call *naïve* and *intelligent* query generation. In the naïve version, a random user requests a random file, and this is recorded as a query in the trace file. The intelligent version adds stipulations to prevent a large number of incomplete-able transactions at runtime. First, a user may not request a file they already possess or requested in the past. This eliminates the need for any user to store multiple copies of the same file or determine which copy should persist. Second, a requested file must exist in the network. A query which returns no results is inconsequential. This is not, however, a guarantee a file will be available when it is requested as all owners may have no bandwidth (see below).

Every user has an equal probability of being a file requester. Though some user models may request files at varying rates, this is not something we attempt to model here. Which file is requested is dictated by the same Zipf distribution used to populate initial libraries.

We envision several future improvements in this area. First, P2P interactions usually occur in *cliques*, with users only interested in some genres of files as opposed to all those available. This is a point revisited in the 'Future Work' section. Second, it is intuitive to expect a good user who

receives a bad file to re-query and attempt to obtain a valid copy. Unfortunately, the static nature of our trace nature cannot support such an operation.

TRACE SIMULATION

We next describe the *simulator* wherein the trace is dynamically run and relevant statistics are output. After initialization is complete the simulator proceeds in the following simplified loop:

```
While more queries remain:
   Read query from trace file
   Broadcast query to determine potential providers
   Compute trust-values for relevant user pairs
   Select bandwidth-available source user
   Copy source file to requester's library
   Requester submits feedback concerning source
End
```

Bandwidth & Load Distribution

With RAs it is possible for the global trust view to converge and identify a single or small set of users as 'most trustworthy.' These nodes are likely to be overwhelmed with provider requests. Bandwidth limitations will prevent some of these requests from being fulfilled or will serve the files at a very slow rate (poor QoS) (Papaioannou & Stamoulis, 2004). Some notion of *load distribution* is needed. Systems like EigenTrust handle this implicitly in their reputation management systems. Others, like TNA-SL, give the topic no attention. Load balancing inherently lowers reputation algorithm performance. Decreasing the load of the most trusted users means the load will increase for less trusted users, and probabilistically speaking, more invalid files will be traded. Even so, load balance is a practical necessity.

We propose that one should ignore load balancing suggestions when implementing a RM system for the simulator. Instead, the RA should export a relative ordering of users based on trust values. The simulator's included bandwidth manager will then objectively handle load balancing by allowing only bandwidth-available users to participate in transactions.

Bandwidth restrictions are set by two command-line parameters. Summarily, a user may have at maximum x connections at any time, and the transaction of a single file requires y time units. Our simulator has a weak notion of time that permits this description, namely, each query requires a single *time unit* and there is only one query per clock tick. A *connection* is a distinct upload/download. For example, if $x = 2$ and $y = 100$, a user who begins a download at time 21 and another at time 56 will have no download bandwidth available until time 121 when a single connection becomes available. Since file transmission is not instantaneous, feedback is delayed until the transaction is complete. Separate upload and download queues are maintained.

Indeed, our approach to load distribution is a basic one. If one wishes to model variable file sizes, connection speeds, *etc.*, matters get very (and perhaps unnecessarily) complex. However, it succeeds in enforcing a tunable and objective model of load balance.

Source Selection

After trust computation it is the requester's responsibility to use the relative ordering to choose a source/provider. As Table 2 shows, the desired source varies depending on user model.

Requester Model	Source
Good	BEST
Purely Malicious	WORST
Feedback Malicious	RANDOM
Malicious Provider	WORST
Disguised Malicious	RANDOM
Sybil Attacker	WORST

Table 2: Source selection by requester model

The *worst* option is chosen by users who wish to increase the scope of their invalid libraries, whereas a *random* approach allows feedback-focused malicious users to boost the global opinion of other bad users and mar that of good ones. Intuitively, *good* users want the *best* provider. Recall that only bandwidth-available users can serve as providers. Ties between users with identical trust values are broken randomly.

Feedback Database

Reputation algorithms aggregate interaction histories to produce a trust value characterizing some user-to-user relationship. We now describe how these interaction histories are stored. In our architecture the feedback database (DB) is *centralized*, so it appears identical from every user's viewpoint. A centralized DB means trust computation can be centralized, as well. Our framework will assume such a centralized *trust service* exists, for the time being.

In practice, P2P networks (among other topologies) are not centralized and the DB and/or computation must be distributed. Nonetheless, when feasible (and responsibly hosted) centralized DBs are secure and convenient, as distribution may expose holes for exploitation. In later sections we relax our fully-centralized approach to aid in describing malicious attacks.

Minimally, a feedback DB needs to store the number of positive and negative prior interactions between directed (non-symmetric) user pairs. Other data may also be stored, as required by different RAs. For example, associating a time-stamp with each feedback might allow an algorithm to weigh recent interactions more heavily than those occurring long ago. Similarly, context information might be entered, *e.g.*, user *X* had a negative experience with user *Y* with respect to his download of file *Z*. Such entries may permit trust to be computed at fine granularity. By querying the DB and retrieving only those rows where *Z* was the file involved, and letting the RA aggregate over only these entries, it may be possible to compute trust in individual files. This is a complex topic we revisit in our 'Future Work' section.

Because we have control over the environment in which our feedback DB operates its security can be assumed. Real DBs would be afforded no such luxury. It is likely DBs will be interfaced via a RPC. Ballot-stuffing attacks (Srivatsa *et al.*, 2005) and identity management are two practical challenges that must be addressed, but are beyond the scope of this work.

REPUTATION ALGORITHMS

There are many RAs in existence (Li & Singhai, 2007) though we will only give attention to EigenTrust (Kamvar *et al.*, 2003) and a modified TNA-SL (Jøsang *et al.*, 2006). Precise details of these systems can be found in their respective papers; our concern is with their general behavior and efficiency. Why these two algorithms? One (EigenTrust) is terse, scalable, efficient, implementation-ready, and designed with P2P in mind. The other (TNA-SL) is theoretical, expressive, and inefficient in a P2P setting. By designing our framework with these two radically different systems in mind, we believe our work will be applicable to the vast expanse of other RAs that lie between them. Furthermore, the significant differences will make for more interesting test runs in forthcoming sections.

The simulator is designed to promote the easy addition of new RAs; a simple calling interface is used. Most important is the interfaced `computeTrust()` method. Given a source user and access to feedback data, it returns a vector quantifying the trust user *source* has in other users in the network. At a high-level, we will now describe three RAs for the sake of familiarizing the reader with their basic operation and implementation.

None (No RA Present)

None simulates the absence of a reputation system and uses exclusively random file-providers. At initialization all trust values are set to an identical value; every call thereafter allows these original trust values to persist. Thus, a user model selecting the best/worst source will result in a tie between all available users that will be broken randomly. This ‘system’ is of little interest, aside from providing a baseline for comparison purposes.

EigenTrust

EigenTrust (Kamvar *et al.*, 2003) is a system that uses local normalization combined with global convergence via vector-matrix multiplication. Prior interaction is best visualized as an $n \times n$ matrix where n is the number of peers. Vectors are local to users. For example, the entry in the i-th row and j-th column describes user j's directed dealings with user i. We will next describe how trust is calculated in EigenTrust. Figure 2 provides a concrete example.

$$A = \begin{bmatrix} \binom{pos:0}{neg:0} = 0 & \binom{pos:3}{neg:1} = 2 & \binom{pos:3}{neg:2} = 1 \\ \binom{pos:9}{neg:3} = 6 & \binom{pos:0}{neg:0} = 0 & \binom{pos:8}{neg:1} = 7 \\ \binom{pos:2}{neg:4} = 0 & \binom{pos:5}{neg:4} = 1 & \binom{pos:0}{neg:0} = 0 \end{bmatrix}$$

$$A' = \begin{bmatrix} {}^{0}/_{6} & {}^{2}/_{3} & {}^{1}/_{8} \\ {}^{6}/_{6} & {}^{0}/_{3} & {}^{7}/_{8} \\ {}^{0}/_{6} & {}^{1}/_{3} & {}^{0}/_{8} \end{bmatrix} \quad p = \begin{bmatrix} {}^{1}/_{3} \\ {}^{1}/_{3} \\ {}^{1}/_{3} \end{bmatrix} \quad t_{\infty} = \begin{bmatrix} 0.35 \\ 0.49 \\ 0.16 \end{bmatrix}$$

Where $t_0 = p$ and $t_{k+1} = (0.5 \times A'^{T} \times t_k) + (0.5 \times p)$

Figure 2: An example EigenTrust computation

A *relation* stores the number of positive (`pos`) and negative (`neg`) interactions between two users. For each relation, a single feedback integer is calculated as `max(0,(pos-neg))` as in example matrix *A*. Next, these values are normalized on a vector basis (example *A'*). Normalized vector *p* of size *n* is then initialized to encode a-priori notions of trust. Next, t_k is calculated using the above formulae. For sufficiently high *k*, vector t_k will converge to the left principal eigenvector of *A'*, the *global trust vector* (t_∞). The *i*-th position of t_∞ is *i*'s trust value. For our purposes, *k* need only be so high that the relative order of trust values is fixed.

The proper initialization of *p* is critical to EigenTrust success. Greater weight is given to those users deemed 'pre-trusted'. In practice, a small but highly trusted subset of *z* users are designated pre-trusted. Then, $p_i = (1/z)$ if user *i* is pre-trusted, and $p_i = 0$, otherwise.

Trust Network Analysis with Subjective Logic (TNA-SL)

Trust Network Analysis with Subjective Logic (TNA-SL) (Jøsang, 2001; Jøsang *et al.*, 2003) is a system that places greater emphasis on prior direct interaction. Here, trust is stored as an *opinion*, a 4-tuple (b, d, u, α) that represents belief, disbelief, uncertainty, and a base-rate, respectively. At all times ($b + d + u$) = 1 and α (a real) in [0...1] is used to store a-priori notions of trust. Converting past interaction into an opinion is done as shown in Table 3. Though there are many logical operators, two, *discount* and *consensus*, are of note. Table 4 shows their calculation.

belief	=	(*pos* / (*pos* + *neg* + 2.0))
disbelief	=	(*neg* / (*pos* + *neg* + 2.0))
uncertainty	=	(2.0 / (*pos* + *neg* + 2.0))
base-rate	=	1.0 - if user is pre-trusted 0.5 - otherwise

Table 3: Calculating opinions from prior interaction

Discount is used to evaluate transitive chains. Discount would be used, for example, if user *A* wanted to calculate an opinion about user *C* using information at intermediate user *B*. Notionally this would be written as

$$\omega_C^{A:B} = \omega_B^A \otimes \omega_C^B.$$

Consensus is used to average together two opinions. Suppose user *A* and user *B* both have opinions about user *C*. To consolidate these, consensus would be used, with notation

$$\omega_C^{A \diamond B} = \omega_C^A \oplus \omega_C^B.$$

Discount: $\otimes$	Consensus: $\oplus$
$b_C^{A:B} = b_B^A b_C^B$	$b_C^{A \diamond B} = b_C^A u_C^B + b_C^B u_C^A / denominator$
$d_C^{A:B} = b_B^A d_C^B$	$d_C^{A \diamond B} = d_C^A u_C^B + d_C^B u_C^A / denominator$
$u_C^{A:B} = d_B^A + u_B^A + b_B^A u_C^B$	$u_C^{A \diamond B} = u_C^A u_C^B / denominator$
$a_C^{A:B} = a_C^B$	$a_C^{A \diamond B} = a_C^A$
Where $denominator = u_C^A + u_C^B - u_C^A u_C^B$	

Table 4: Defining discount and consensus operators

Computing trust given these operations and a digraph of opinions (constructed from a feedback DB) is not straightforward. The approach given in the describing paper (Jøsang *et al.*, 2003) requires one to find an acyclic direct series-parallel graph (DSPG) between the requester and potential source that minimizes uncertainty. Then, given that DSPG, a single characterizing opinion is derived by applying the SL-operators (note that *discount* corresponds to series composition and *consensus* to parallel composition). For export to a single trust value, the *expected value* of that opinion is calculated as $EV = b + au$.

While theoretically sound, applying this approach to our simulator is computationally infeasible for three reasons. (1) Our network is fully connected, yielding an exponential number of paths. (2) The DSPG recognition problem, while solvable in linear time[7], becomes burdensome due to the sheer number of times it must be called (though caching can help). (3)

The described procedure computes trust between just two users. To populate the output vector the above reduction and analysis would need to be computed n times.

While the DSPG approach does not apply well to this scenario, the expressiveness of opinion objects could be beneficial if harnessed in a different manner. How to best and efficiently aggregate trust remains an open question. When cycles are present it is difficult to exhaustively analyze a graph. Finding a single most-trusted (or most certain) path is possible, but ignores data the other (perhaps contrary) paths provide. We have found it effective to use an aggregation strategy that *maximizes the consensus certainty at some depth of search*. We do not claim this is the best means by which to evaluate trust – rather, that it is a reasonable one. Our method requires no graph refinement, is mathematically elegant, and takes a large amount of global opinion into account. Furthermore, simulations will demonstrate our technique's effectiveness.

To compute trust for a network of n users, a matrix, A, of opinion objects with dimension $n \times n$ is created. Just as in the EigenTrust example, each matrix element defines a user-user relationship. Table 3 describes how to compute opinions from feedback histories. Then, compute A^x for large x, using the *discount* and *consensus* operators to overload multiplication and addition, respectively. Because of the non-monotonic nature of the consensus operator and our lack of normalization, taking A to a high power will not result in convergence (as with EigenTrust). Instead, *belief* values will tend towards 0 because the discount operator weakens trust, and thus, many discounts in a long transitive chain will severely weaken it. Instead, we create a separate opinion matrix A', to store the opinion with maximum confidence seen at each position throughout the multiplication process. We define $EV(A')$ to be the global trust matrix and $EV(A'_{i,j})$ represents the trust user j has in user i.

Because our technique uses the subjective logic (SL) operators to analyze opinion digraphs, *TNA-SL* remains our term of choice for the modified strategy. The majority of TNA-SL behavior (in either case) is captured by the SL-operators, not the method by which they are applied. Thus, we see foresee no serious conflict arising from our redundant terminology. In situations where a significant difference is foreseen between our approach and the original, note will be made.

EVALUATION METRICS

The primary thrust of our simulator is to compare RA effectiveness. Below, we define a succinct metric for this purpose. However, the framework affords us other assessment opportunities. In particular, algorithm efficiency can be examined and speed-up strategies tested.

Effectiveness Metric

When comparing the effectiveness of RM simulations it is helpful to have a single evaluation metric, which we define as:

$$\text{Metric} = \frac{\text{\# valid files received by 'good' users}}{\text{\# transactions attempted by 'good' users}}$$

Effective RM systems do not clean-up networks; they only provide *accurate* trust information. Such a system will not only help good users find good providers, it will succeed in

helping malicious users find bad providers. However, we do not care about the success of malicious users so our objective function is based solely on the *good* user success rate.

Simulation Efficiency

Through examination of simulation run times one can gain insight into the efficiency/scalability of the implemented RAs. Timings of simulations run over realistic data are arguably more useful than the asymptotic complexity analysis found in the papers describing these systems. Additionally, speed-up strategies can be developed and their effectiveness can be validated. Such strategies have been given considerable attention in our simulator development because inefficient algorithms create inefficient simulations. We now describe why efficiency is a particularly acute problem and introduce a general-purpose strategy for improving it.

For a network of n users, there are n^2 user-to-user relationships, each with their own trust value. Since RAs often exploit transitive trust, there are a multitude of network paths to explore, often implemented as matrix multiplication. Examining long transitive chains takes many multiplications and trust is recomputed after every transaction. As a result, some reputation systems, especially expressive and theoretic ones, pose a large computational burden. This inhibits experimentation with large networks and high transaction counts.

EigenTrust is quite scalable because of its globally convergent values via vector-matrix multiply. TNA-SL (our version), with its user-centric trust values and matrix-matrix multiplication is not so fortunate[8]. For example, the 50-user 50k transaction traces like those analyzed herein take ≈ 2 seconds to run under EigenTrust but 2.5 minutes under TNA-SL. An increase to 100 users takes 5 seconds for EigenTrust and over 10 minutes for TNA-SL.

Our framework's scalability is disadvantaged in two ways. First, trust computation is often distributed among nodes, a notion our centralized simulator cannot take advantage of. Second, our network's fully-connected nature may lead to an unrealistic explosion of network paths.

While burdensome, these difficulties pale in comparison to the efficiency hurdles faced in actual networks. Consider that a popular P2P application might have hundreds-of-thousands of users. Exhaustive trust analysis in such situations is impossible and approximation techniques must be used. For example, a RA may consider only feedbacks from a random subset of users, utilize feedbacks from only the most experienced users, or limit path search-depth. The first approach is one tested and advocated in the work of Papaioannou and Stamoulis (2004).

Fortunately, as Kamvar *et al.* (2003) suggests and our own work confirms, simulations scale intuitively. That is, an x user simulation with y malicious peers produces statistical ratios and evaluation metrics comparable to that of a $2x$ user simulation with $2y$ malicious peers. This fact, combined with our (simulator-based) heuristic efficiency improvements introduced below, allow the production of meaningful results in a reasonable time frame.

Heuristically Improving Simulation Efficiency

By taking advantage of the static nature of traces and the preponderance of consistent behavior in simulations, efficiency improvements can be made while maintaining near-correctness. First, consider that user models are constant during a run, *i.e.*, users behave in consistent ways. Second, as a test run progresses each successive feedback has less influence because it is being

considered along with a growing body of previous feedbacks. Over time the network becomes 'solved' and trust values change very little. This allows one to use slightly aged trust-values with minimal consequence and not have to recompute trust after every transaction.

As a preliminary experiment, we decided to confirm that the trust networks we generate are indeed solvable. To do so, we generated 50-user, 50k transaction traces with between 0-100% purely malicious users. We then ran the traces (using TNA-SL) and recomputed trust after every cycle for the first n = {0, 250, 1k, 2.5k, 5k, 50k} cycles. After that point, transactions were executed until all 50k were complete, but the persistent (aged) trust values were used for source selection. The trials where n = 50k act as a control and define correctness, since trust is always recomputed in that case. Figure 3 displays how lower n values fared[9].

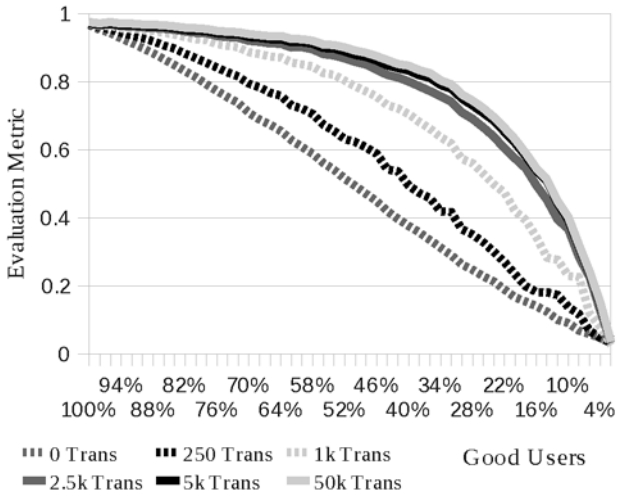

Figure 3: Trust convergence rate (TNA-SL)

By approximately n = 2.5k, and certainly by n = 5k, the margin of error is < 1% at all data points. Therefore, about 90% of the trust computations performed are unnecessary on runs of this nature, which were designed to difficult to 'solve.' In practice, only computing trust up to some point is not the best idea, instead, an evaluation interval should be used, which we term *skip*. Initially `skip:=1` and every time `transac_num % skip == 0`, trust is recomputed and a snapshot of the trust values taken. At each recalculation, the new trust values are compared against those of the previous snapshot. If each entry deviates less than some ε, then `skip *= 2`, else `skip /=2`. Bounding `skip` to `[1...32]` helps maintain correctness.

Tests have shown that at the beginning of a run the value of `skip` fluctuates and remains low, but quickly saturates leading to a speedup of ≈ 32×. The adaptive nature of this technique should make it conducive for use with any algorithm. Furthermore, this adaptivity would permit speedups in more dynamic systems, perhaps even those present in real-life.

Indeed, the described strategy is an approximation. Those trust (re-)calculations skipped over may have changed the relative trust ordering, and thus source selection, and the resulting metric. Our tests have found this gap to be minimal. Since correctness is not guaranteed, however, the performance graphs presented later in this chapter were not generated using the described heuristic. The approximation, however, was helpful in the rapid prototyping of test runs.

More subtle speed-up strategies are often algorithm-specific. Lots of persistent storage is used to minimize redundant calculations at each call. For matrix-multiplication strategies it is

important to do the minimal amount of multiplies necessary to reach convergence/saturation. The static approach of simply doing *x* multiplies each time is unacceptable because *x* must be high enough to buffer for worst-case behavior. In the EigenTrust case, multiplication ceases once convergence is indicated via a precision ε on the global vector. For (our version of) TNA-SL, if a multiplication makes no update to the maximal matrix, no more are needed.

TEST RUNS AND OBSERVATIONS

Using simulator runs we will now exemplify behavioral aspects of RAs. This gives us an opportunity not only to test RAs but our simulator itself. We begin by examining how RAs manage adversarial user models, like the *purely malicious* one, and then look at how varying network conditions, *i.e.*, bandwidth and interaction density, affect the evaluation metric.

Simple Test Runs

The simplest test is how a RA handles *malicious providers*; those that own and acquire invalid files but give honest feedback. Figure 4 demonstrates how trivially managed such a system is.

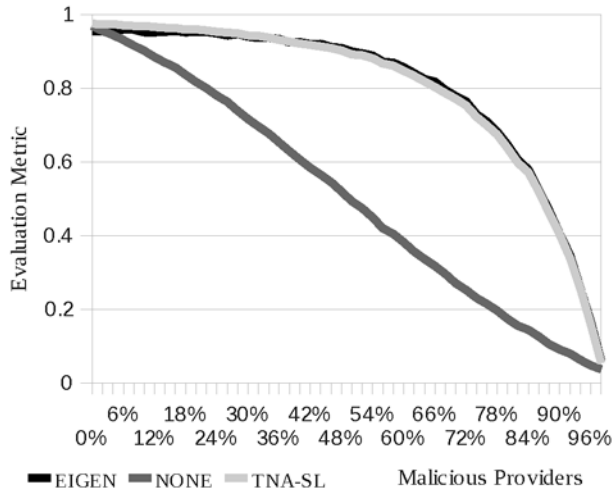

Figure 4: RAs vs. Malicious Providers

The graph was produced from 50-user, 50k transaction traces with no pre-trusted peers, and infinite bandwidth. The *y*-axis corresponds to the evaluation metric of the previous section. The *x*-axis represents the number of malicious users present, *i.e.*, a data point with 72% malicious providers has (0.72*50) = 36 *malicious providers*, and the remaining (50-36) = 14 users are *good* ones. The data sets correspond to the different reputation algorithms being evaluated.

As shown, NONE always regresses in linear fashion. As expected, all RAs converge to a metric near 0% when the network is completely saturated with bad users, since they remove good files from their libraries. Any RA with a metric scoring above control line NONE is considered a success. *Malicious providers* do not present a serious threat to a network because they are honest about their poor intentions. Thus, the algorithm success demonstrated in Figure 4 is not seen as significant, but rather, a sanity-check on implementation correctness.

Figure 5 shows a more interesting graph where invalid files *and* feedback dishonesty are present. It is parameterized just as Figure 4, except that *purely malicious* users now replace *malicious providers*. While TNA-SL is still very successful, EigenTrust is less impressive, narrowly improving upon the control metric. Why does EigenTrust perform so poorly?

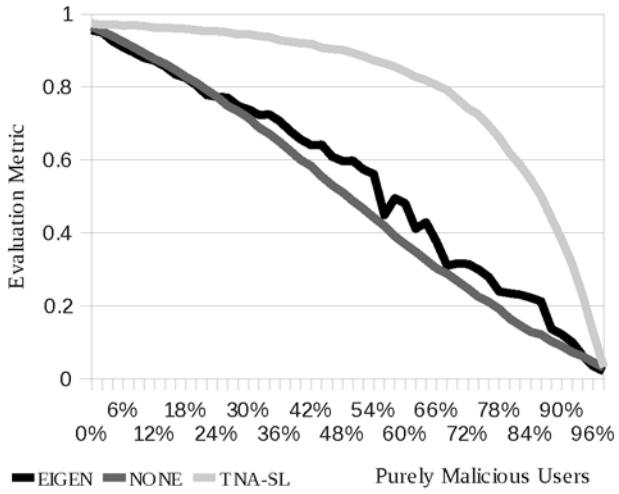

Figure 5: RAs vs. Purely Malicious users

As we shall see, the notion of *pre-trusted peers* corrects this EigenTrust deficiency. For now, we concentrate on the simpler case where they are not included. So then, why does TNA-SL so dramatically outperform EigenTrust in this example? TNA-SL weighs local (*i.e.*, direct) interaction history more heavily than transitive feedback data due to the nature of the discount operator. In contrast, EigenTrust computes a global 'average.' From a *good* user's perspective, the average global view of a network overrun with malice is probably an inaccurate one. In such cases, the only person you should trust is yourself, and TNA-SL encodes precisely this notion by using user-centric trust values. When available and in quantity, direct interaction data is more valuable than that received transitively because it is known to be accurate.

EigenTrust has the capability to weigh local information more heavily. By performing a weighted average of the normalized (pre-multiplication) local vector with the converged (post-multiplication) global trust vector, the local:global preference can be shifted. This average is a less powerful notion than that TNA-SL provides and is not a feature we examine in this chapter.

Another surprising aspect of Figure 5 is that EigenTrust still provides improvements over NONE even when the network is overwhelmed by malicious users. It is unintuitive a super-majority of bad users can't manipulate the system for their benefit. Recall these users are in non-organized *casual* cooperation. Purely malicious users lie, making bad guys appear good, and good guys appear bad. With lots of bad guys the whole notion of good and bad becomes reversed. Bad guys trying to download from other bad guys end up downloading from good users because misinformation is so plentiful. As we shall see, malicious users need additional capabilities for their misbehavior to have a serious effect. Fortunately, the simplistic approach of simulator permits the analysis and description of such second and third order behaviors.

Pre-Trusted Peers

We now introduce *pre-trusted peers* into our simulations. Figure 6 presents just such a test run, parameterized precisely as those above except with n pre-trusted users where n in {0, 5}. Note when $n = 0$ the data is identical to that presented in Figure 5. For implementation purposes, pre-trusted peers are a subset of *good* users. For EigenTrust, the inclusion of pre-trusted peers produces a staggering improvement. Pre-trust makes global aggregation less naïve and permits other subtle improvements discussed by Kamvar *et al.* (2003). For example, new users with *no* network experience place all of their initial trust in the pre-trusted user set.

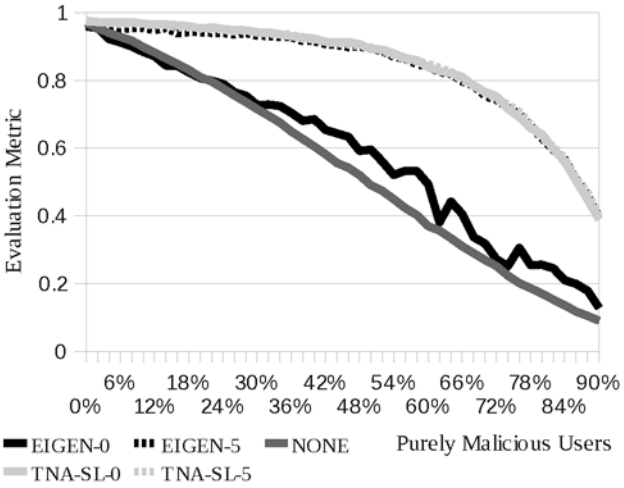

Figure 6: Introducing pre-trusted nodes

For TNA-SL the gain is minimal, however, there is little room to improve. When $n = 5$, both algorithms perform near-ideally, that is, the evaluation metrics are the highest attainable when one considers valid file copies may be scarce in malice networks. Notice in Figure 6 that the evaluation metric begins a rapid decline for the high performing algorithms, *i.e.*, datasets EIGEN-5 and TNA-SL-5, when the network has 66% or more *purely malicious* users. These low metrics are a side effect of our constraint that transactions must be completed, when possible. Thus, *good* users are likely aware that they are about to download from a poor user, but simply have no better alternative. Enabling users to 'decline' a transaction after examination of trust values would present a myriad of challenges. Not only would this strain the simulator's static nature, but trust values would need an absolute interpretation (EigenTrust's are relative).

Reduced Interaction Density

The 'near-ideal' performance exhibited to this point is unsatisfactory. More interesting are the parameterizations that challenge the RAs. Let us begin with interaction density. The 50-user, 50k transaction traces we have been examining produce ≈ 20 interactions between each user pair; an abundance of direct experience. While such densities do allow us to study convergent algorithm behavior, actual P2P networks are far sparser. A reduced transaction count will simulate a sparse network, and the less complete information should test the transitive functionality of the RAs.

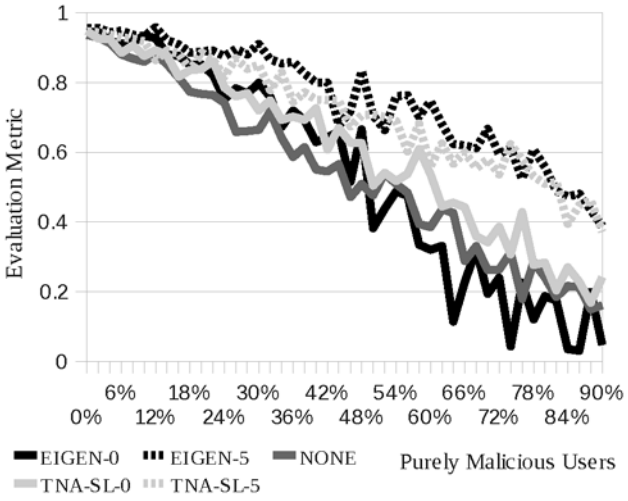

Figure 7: RAs in a sparse network

Figure 7 is set up just as Figure 6 except there are only 500 transactions, leading to ≈ (1/5) interactions between each user pair. The graph shows that pre-trust is particularly important when data is not plentiful. Without it, both algorithms perform comparably to the control line.

It is possible for EigenTrust to outperform TNA-SL, as Figure 7 shows at many data points. In general, however, TNA-SL's expressiveness (*e.g.*, the ability to quantify negative trust) compared to EigenTrust results in higher evaluation metrics while sacrificing efficiency.

Tightening Bandwidth Constraints

To this point, we have only examined parameterizations with unlimited bandwidth. Just as with plentiful interaction densities, this condition is unrealistically beneficial to the RA being studied. When bandwidth constraints are enforced, users may be unable to source-select their most satisfactory choice. We demonstrate that bandwidth limitations are necessary via Figure 8; the cumulative distribution function (CDF) of uploads per the percentage of network participants.

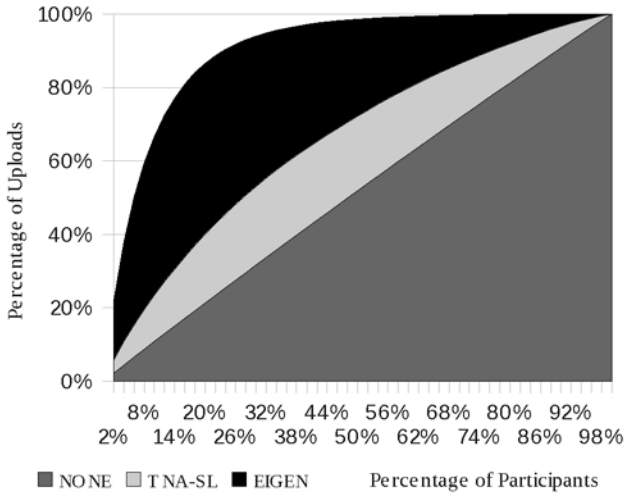

Figure 8: CDF of upload frequency

Figure 8 is parameterized with 50 *good* users and unlimited bandwidth over 50k transactions. As the graph shows, during EigenTrust runs just 20% of users account for 90% of all uploads; evidence that a subset of users become identified as 'most trusted.' While good for trust metrics this is clearly not the best utilization of network bandwidth. More interesting is the graph of Figure 9, showing how identical traces fared under varying upload bandwidth availability.

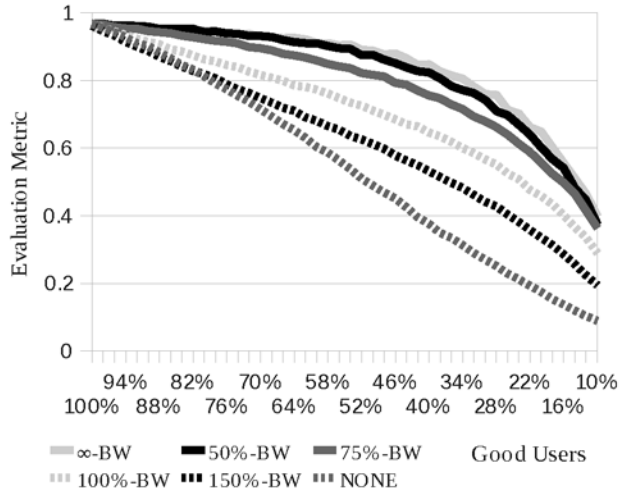

Figure 9: RAs under varying bandwidth constraints

As Figure 9 shows, tight bandwidth inhibits RA performance. The percentages associated with data sets are indicative of the bandwidth utilization, *i.e.*, dataset '100%-BW' means the trace could be run with no incomplete-able transactions if-and-only-if every user utilized the full allotment (100%) of bandwidth given to them. Tighter bandwidth constraints make RA performance tend towards the control line (NONE). Just as bandwidth may force *good* users to select *less trusted* providers, malicious users may have to select *more trusted* ones. Bandwidth, important for realism, is largely uninteresting from a trust perspective; only acting as a scalar on

the metric gap between systems. Bandwidth constraints also produce incomplete-able transactions from which no useful statistics can be derived. For these reasons, we have -- and will continue to -- present scenarios with unlimited bandwidth.

EMPOWERING MALICIOUS USERS

To this point test runs have shown impressive RA performance. However, the fact that networks with upwards of 75% malicious users can still be managed may suggest our malicious user models are not sufficiently powerful. Rather than being an endorsement of RAs our simulator should strive to find their breaking points, even if by unrealistic means.

Assumptions Benefiting Bad Users

Starting on the complementary end, several of our assumptions are *beneficial* to malicious users. First, our simulator has a 'closed world.' Users do not come and go from the network; all participants are available to participate in transactions from the outset. In a more realistic setting, networks would be built by well-intentioned users and some basis of trust would be established between *good* users before malicious ones appeared. Furthermore, it is unlikely a significant number of bad users will enter the network at a single time. Therefore the RA would be able to 'cast aside' malicious users as they are identified and not have to deal with the cumulative effect of simultaneous arrival. Secondly, as discussed previously, complete-able transactions must complete. Thus, when limited file copies exist or in tight bandwidth situations, users further down the relative ordering (*i.e.*, less trustworthy) will be given the chance to disseminate files.

Distributed Schema

On the other hand, some of our assumptions *inhibit* malicious users from gaining an advantage. First, RAs are usually implemented in a distributed setting and thus interaction and trust value storage are distributed, as well. It is insecure to let users store and report their own data. In practice, *score management systems*, like TrustMe (Singh & Liu, 2003) or that of Yang, Kamvar, and Garcia-molina (2003) are used. For security reasons, data is stored redundantly. When discrepancies arise, a vote determines how to proceed. Thus, a large and cooperative group of malicious users can agree on dishonest values and subvert a reputation framework entirely.

Our simulator uses a centralized feedback-store and trust service, and therefore cannot fall prey to such subversion. This fact may unfairly hamper malicious user efforts. However, a centralized approach is simple and efficient for simulation purposes. Also, readers must realize that an RA and the distribution scheme it employs are two separate entities. While the EigenTrust paper uses MOTHERS (Yang *et al.*, 2003) for secure distribution, other score managers may work just as well. One should be careful not to apply the deficiencies of distribution strategies to reputation management algorithms.

A centralized reputation database does have a severe weakness; everyone has a consistent view of the network. Intuitively, malicious users want to lie to *good* users but provide their

malicious colleagues good information. We now describe how such differing perspectives can be realized without having to implement score managers or distribution schema.

Empowering Isolated Malicious Users

To exemplify our architecture's weakness, let us examine the case of a single, isolated, *purely malicious* user, *u*. Suppose TNA-SL, with its preference for direct experience, is the RA being used. User *u* receives a valid file from a *good* user, *v*, and dishonestly submits negative feedback. Now, the next time user *u* queries and needs to compute trust he will go the feedback database and aggregate over feedback(s), including those that he/she previously submitted. In particular, `trust(u->v)` may be low as it is characterized by one or more negative feedbacks. User *u*, being malicious, is then likely to use *v* as a source. However, *v* is actually a good user, *u* gets a good file, *u* has less bad files, and the network is slightly better on the whole. Our (static) source selection criteria (Table 2) operate on the assumption that stored interactions are honest – behavioral patterns become uncharacteristic when this is not the case.

Luckily, this it is not difficult to correct. We previously assumed the feedback DB and trust computation were centralized. Relaxing this to allow *distributed trust computation* gives malicious users more power. Now, we assume a user who wants to make a trust inquiry will export a copy of the centralized feedback DB to his/her own machine and perform trust computation locally. Further, in addition to submitting (perhaps dishonest) feedbacks to the global-DB, users will also locally store a vector of *completely honest* interaction history.

After the centralized feedback DB has been imported, and before trust computation is done, users will override 'their' vector in the global matrix with their local honest one. For example, in Figure 10, we assume user *u* (ID=0) is computing trust. Therein, *A* is the (summarized) feedback DB, *w* is *u*'s local honest vector, and *A'* is subsequently given to the RA to compute trust[10].

$$A = \begin{bmatrix} \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} & \begin{pmatrix} pos:3 \\ neg:1 \end{pmatrix} & \begin{pmatrix} pos:3 \\ neg:2 \end{pmatrix} \\ \begin{pmatrix} pos:9 \\ neg:3 \end{pmatrix} & \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} & \begin{pmatrix} pos:8 \\ neg:1 \end{pmatrix} \\ \begin{pmatrix} pos:2 \\ neg:4 \end{pmatrix} & \begin{pmatrix} pos:5 \\ neg:4 \end{pmatrix} & \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} \end{bmatrix}, \; w = \begin{bmatrix} \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} \\ \begin{pmatrix} pos:3 \\ neg:9 \end{pmatrix} \\ \begin{pmatrix} pos:4 \\ neg:2 \end{pmatrix} \end{bmatrix}, A' = \begin{bmatrix} w_{0,0} & A_{0,1} & A_{0,2} \\ w_{1,0} & A_{1,1} & A_{1,2} \\ w_{2,0} & A_{2,1} & A_{2,2} \end{bmatrix}$$

Figure 10: Overwriting global feedback data

Test runs have demonstrated this benefits malicious users by the *smallest* of margins. This is intuitive: The local vectors of *good* users are identical to those in the global-DB. *Good* users compute trust and source-select precisely as before. Malicious users compute trust differently, but the evaluation metric does not concern them. For malicious users to deteriorate the evaluation metric, they must skew the opinions of *good* users. Enter the malicious collective.

Empowering Malicious Collectives

A *malicious collective* is a set of nodes cooperating to increase the number of invalid files *good* users receive. Assume collective members are aware of all other participating members. Now, malicious users do not just use their *honest interaction vector* for local trust computation, they also broadcast it to others in the collective so they may gain a more accurate network view.

This capability alone is insufficient for a malicious collective to deliver invalid files. Imagine a network has a set of *good* users and a group of *purely malicious* ones. Simulation shows the two groups become disjoint subgraphs. Good users trade with other good users, with great success, and because of the enhanced capability, bad users become really good at finding bad files. Still, interaction rarely crosses this boundary, as is necessary for the metric to decrease.

For a malicious collective to succeed its members need to have well-defined and coordinated roles. We now describe one such strategy. Suppose a network has a set of good users, *G*, and malicious collective, *M,* which consists of a set of *feedback malicious* users, *FM*, and a set of *purely malicious* users, *PM*. The nodes of *FM* distribute good files and therefore `trust(G->FM)` is high. *FM* members are also liars so their global vectors indicate `trust(FM->PM)` is high. Now, from a transitive perspective `trust(G->FM->PM)` is high, so in the absence of contrary direct experience of the form `trust(G->PM)`, *good* users are likely to obtain bad files in this scenario, which we term the *malicious gateway* strategy.

Simulation confirms we have alas given malicious users sufficient power to subvert a reputation algorithm. Figure 11, implementing the *malicious gateway* strategy, shows EigenTrust (w/o pre-trust) is particularly vulnerable to this approach. The addition of pre-trusted peers corrects this deficiency (by-in-large) and seems to suggest that networks implementing well-chosen pre-trusted peers are robust to even collective attacks.

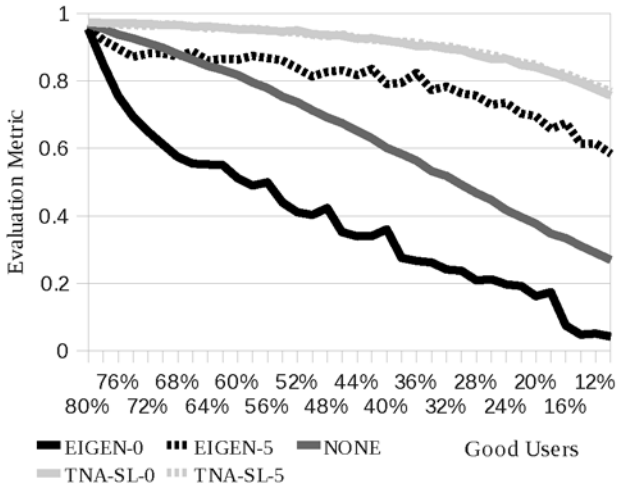

Figure 11: Simulating the malicious gateway strategy

Having demonstrated our simulator has sufficient power to model a group of cooperating malicious users overtaking a network -- albeit one managed by a slightly naïve global average algorithm -- our examination concludes. We leave the invention, implementation, and testing of other collective strategies as an exercise for the reader. Our own attempts have shown this task to be a difficult one. When confined to only algorithm-based attacks, (*i.e.*, not attacking feedback mechanisms or distribution strategies) RAs appear remarkably robust.

CONCLUSIONS

Emerging decentralized topologies offer benefits over their centralized counterparts. However, to take advantage of these, reputation management needs used to limit corruption and malicious behavior. Though many such RM systems exist, prior to our proposal of one in this chapter, there was no convenient means to objectively compare them or verify authors' claims.

Herein, we reported on our framework's design decisions. First, we explained static network trace generation. Particular attention was given to user initialization; utilizing two behavioral dimensions – clean-up and feedback-honesty. Next, we demonstrated how traces are simulated with bandwidth, trust computation, and source selection among the most dynamic aspects.

New reputation algorithms may be easily interfaced into our framework. Two in particular (EigenTrust and a modified TNA-SL) were discussed and have been implemented. These RAs provided us the opportunity to discuss the implementation details, efficiency hurdles, and behavioral qualities that characterize many systems. Example analyses increased our understanding of existing RAs and helped us refine malicious user strategies.

These test simulations were an endorsement of the current state of RAs. Malicious collectives, even large ones, were unable to subvert the robust forms of either algorithm. Further, in the vast majority of parameterizations, the systems enforced near ideal behavior. Though we did not intend this study to be an endorsement of the current methodology, it ended up being just that, as we encountered significant challenges in deteriorating algorithm performance.

Future Work

Though RA analysis with the simulator has been beneficial, improvements could make it an even more powerful tool. Though it was our intention to simulate simplified networks, some of our departures from realism may be too significant. First, our 'closed world' approach may need re-thinking. Allowing users to enter and exit the network, change user models mid-trace (Srivatsa *et al.*, 2005), and insert new files into the network are all future considerations. While possible, such notions strain the framework's static nature[11]. Similarly, our notion of time needs enhanced.

Second, network connectivity/topology needs examined. In our simulator, we assume everyone trades with *everyone* else in the network. Limiting connectivity limits direct experience, which in turn, tests transitive trust propagation. *Cliques m*ay be one solution to the problem, *i.e.*, genres of files. One should consult the work of Saroiu, Gummandi, and Gribble (2002) to learn about such properties in actual P2P networks.

Third, feedback mechanisms need to be refined. It is clear that the receiver (client) of a file (service) should enter feedback regarding the provider (server). Entry of additional feedbacks may allow for more powerful methods of trust computation. Suppose user *A* downloaded a bad file from *B*, because users *C* and *D* suggested that *B* was a good choice. In addition to the obvious `feedback(A->B, NEG)`, *A* may also want to submit feedbacks indicating that *B* and *C* were poor referrers. With entries of this style, one can determine referral trust of users. Properly implemented, this could defend against the *malicious gateway* strategy. Similarly, trust in individual files may be calculated via context and feedback filtering. Such analysis can likely be performed to infinite specificity, but less relevant feedback will exist as scope narrows. Adaptive approaches will need to be developed to consolidate values from varying granularities.

Similarly, we need to develop methods to verify that whoever is entering feedback in a DB is justified in doing so. Perhaps a token issued at service completion will need to be presented at feedback submittal in order to verify this fact. Still, this does not entirely prevent ballot stuffing attacks. When the costs and risks of an exchange are high, cost-benefit analyses and decision meta-policies might need to be implemented.

Fifth, we have demonstrated that pre-trusted peers can be critical to the effectiveness of a RA. Research needs focused on how such users should be selected and how varying quantities affect algorithm metrics. From the malicious perspective, it would be interesting to examine how pre-trust can be exploited. For example, malicious users may choose to always deliver valid files to pre-trusted users (if they can identify them), dramatically boosting global opinion. Lastly, out-of-band manipulation of a pre-trusted peer, *i.e.* a botnet takeover, could prove significant.

Sixth, additional RAs need interfaced into our framework, allowing us to confirm the applicability of our design. Furthermore, small, algorithm-specific, enhancements might need made. For example, EigenTrust and TNA-SL both encode the notion of pre-trust. What about algorithms that cannot take advantage of pre-trust, but are dependent on other properties? For example, an RA may want to store feedback timestamps and treat older entries as less relevant. Such properties should be included, so long as they are perceived as being realistic.

Finally, we would like to note the above identification of shortcomings does not invalidate the significance of our simulation results or the usefulness of our current implementation. Rather, they are a series of improvements that should be simple additions to our modular design.

Source Availability

The evaluation framework described herein has been implemented in both C and Java; their functionality is equivalent. The Java code is fully documented and intuitive -- it is the suggested viewing for anyone with casual interest. The C code is not as straightforward but is significantly quicker. The source code and supporting documents are available at `http://rtg.cis.upenn.edu/qtm/`

REFERENCES

Blaze, M., Feigenbaum, J., & Lacy, J. (1996). Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (pp. 164-173). IEEE Computer Society Press.

Breslau, L., Cao, P., Fan, L., Phillips, G., & Shenker, S. (1999). Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of INFOCOM '99: Eighteenth Annual Conference of the IEEE Computer and Communications Societies* (pp. 126-134).

Douceur, J.R., & Donath J.S. (2002). The Sybil Attack. In *First IPTPS* (pp. 251-260).

Hoffman, K., Zage, D., & Nita-Rotaru, C. (2008). A Survey of Attack and Defense Techniques for Reputation Systems. To appear in *ACM Computing Surveys*.

Jøsang, A. (2001). A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness, and Knowledge Based Systems*, 9(3) (pp. 279-311).

Jøsang, A., Hayward, R., & Pope, S. (2006). Trust Network Analysis with Subjective Logic. In *Proceedings of the 29th Australasian Computer Science Conference*.

Kamvar, S.D., Schlosser, M.T., & Garcia-molina H. (2003). The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the Twelfth International World Wide Web Conference* (pp. 640-651). ACM Press.

Li, H., & Singhai, M. (2007). Trust Management in Distributed Systems. *IEEE Computer*, 40(2).

Papaioannou, T.G., & Stamoulis, G.D. (2004). Effective Use of Reputation in Peer-to-Peer Environments. In *Fourth International Scientific Workshop on Global and Peer-to-Peer Computing* (pp. 259-268).

Resnick, P., & Zeckhauser, R. (2001). Trust Among Strangers in Internet Transactions: Empirical Analysis of eBay's Reputation System. Working Paper for *NBER Workshop on Empirical Studies of Electronic Commerce*.

Saroiu, S., Gummadi, P.K., & Gribble, S.D. (2002). A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking*.

Schlosser, M.T., Condie, T.E., & Kamvar, S.D. (2003). Simulating a File-Sharing P2P Network. In *Proceedings of the Workshop on Semantics in Peer-to-Peer and Grid Computing*.

Singh, A., & Liu, L. (2003). TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems. In *Peer-to-Peer Computing 2003* (pp. 142-149).

Srivatsa, M., Xiong, L., & Liu L. (2005). TrustGuard: Countering Vulnerabilities in Reputation Management for Decentralized Overlay Networks. In Proceedings *of the International World Wide Web Conference*.

Valdes, J., Tarjan, R.E., & Lawler, E.L. (1979). The Recognition of Series Parallel Digraphs. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing* (pp. 1-12).

West, A.G., Aviv, A.J., Chang J., Prabhu, V.S., Blaze M., Kannan, S., Lee, I., Smith, J.M., & Sokolsky O. (2009). QuanTM: A Quantified Trust Management System. In *EUROSEC 2009*, pp. 28-35. Nuremberg, Germany. March 2009.

Yang, B., Kamvar, S.D., & Garcia-molina, H. (2003). *Secure Score Management for P2P Systems*. Technical report, Stanford University.

Zhang, Y., Lin, K.J., & Klefstad, R. (2006). DIRECT: A Robust Distributed Broker Framework for Trust and Reputation Management. In *Proceedings of the 8th IEEE Conference on E-Commerce Technology*.

Zhang, Q., Yu, T., & Irwin, K. (2004). A Classification Scheme for Trust Functions in Reputation-Based Trust Management. In *Proceedings of the ISWC Workshop on Trust, Security, and Reputation on the Semantic Web*.

Zipf, G.K. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press, Reading, MA.

ADDITIONAL READING

Aberer, K., & Despotovic, Z. (2001). Managing Trust in a Peer-2-Peer Information System. *In Proceedings of the 9th Intl. Conference on Information and Knowledge Management*.

Aringhieri, R., Damiani, E., De, S., Vimercati, C., Paraboschi, S., & Samarati, P. (2006). Fuzzy Techniques for Trust and Reputation Management in Anonymous Peer-to-Peer Systems. In *Journal of the American Society for Information Science and Technology*, (57) (pp.528-537).

Buchegger, S., & Boudec, J.V. (2004). A Robust Reputation System for Peer-to-Peer and Mobile Ad-hoc Networks. In *Proceedings of P2PEcon* 2004.

Cornelli, F., Damiani, E, Capitani, S., & Paraboschi, S. (2002). Choosing Reputable Servants in a P2P Network. In *Proceedings of the 11th World Wide Web Conference* (pp. 376-386).

Garg, A., & Battiti, R. (2005). *WikiRep: Digital Reputations in Virtual Communities*. Technical Report, University of Trento (Italy).

Grandison, T., & Sloman, M. (2009). A Survey of Trust in Internet Applications. In *IEEE Communications Surveys and Tutorials* 3(4) (pp. 2-16).

Gutscher, A., Heesen, J., & Siemoneit O. (2008). Possibilities and Limitations of Modeling Trust and Reptuation. In *WSPI 2008*.

Jøsang, A., Bhuiyan, T., Xu, Y., & Cox, C. (2008). Combining Trust and Reputation Management for Web-Based Services. In *Proceedings of TrustBus2008* (pp. 69-75).

Jurca, R., & Faltings, B. (2003). An Incentive Compatible Reputation Mechanism. In *Proceedings of the IEEE Conference on E-Commerce 2003* (pp. 285-293).

Kinateder, M. & Rothermel, K. (2003). Architecture and Algorithms for a Distributed Reputation System. In *Proceedings of the 1st International Conference on Trust Management* (pp. 1-16).

Lin, K., Lu, H., Yu, T., & Tai, C. (2005). A Reputation and Trust Management Broker Framework for Web Applications. In *International Conference on e-Technology, e-Commerce, and e-Services* (pp. 262-269). IEEE Press.

Marti, S., & Garcia-molina, H. (2006). Taxonomy of Trust: Categorizing P2P Reputation Systems. In *Computer Networks*, 50 (pp. 472-484).

Vu, L.H., Hauswirth, M., & Aberer, K. (2005). QoS-Based Service Selection and Ranking with Trust and Reputation Management. In *Proceedings of the OTM Confederated International Conferences 2005*, 3760 (pp. 446-483).

Xiong, L., & Liu, L. (2004). PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. In *IEEE Transactions on Knowledge and Data Engineering* (16).

---

[*] This research was supported in part by ONR MURI N00014-07-1-0907.

[1] Systems in the *reputation management* domain are sometimes (erroneously, we believe) called *trust management systems*. We prefer the term *reputation management* since *trust management* was coined to describe policy-based access control. No matter what these systems are called, the values computed by them are almost always called *trust values*. Further complicating the terminology, so called *quantitative trust management* (QTM) systems have been developed, selectively combining desirable TM/RM features (West *et al.*, 2009).

[2] We will use these terms interchangeably. Indeed, *reputation manager* is a convenient term common in literature, but this is avoided as it would introduce acronymic confusion.

[3] RM systems are by no means limited to binary feedback. Continuous variables or ordered sets are also common. We use only positive/negative feedback because it interfaces well will the two RM systems we discuss and it simplifies discussion.

[4] The major challenge is not so much getting users to provide *honest* feedback, as it is getting users to provide *any* feedback. Feedback submission is optional in almost all applications implementing it. Our simulator makes the assumption that feedback is *always* provided.

[5] To support this notion, our simulator allows a *warm-up* period to be specified, in which trust relationships/values can stabilize before statistics begin to be tabulated.

[6] Files are labeled numerically [1…NUM_FILES].

[7] Efficient (linear-time) DSPG recognition is a rather complex topic. The casual reader may find the earlier work of Valdes, Tarjan, and Lawler (1979) to be helpful.

[8] Attempting to bring the DSPG approach to bear on a fully-connected, 50-user network is not the wisest approach. Consider the fact 1.12×10^{15} paths would exist between each user pair.

[9] All data points on the graphs herein are the average of 5+ simulations, where each simulation is produced from a different trace, and each trace is generated using an identical parameterization (aside from the random seed). This is an attempt to reduce variance and produce the most characterizing plots possible. The speed-up heuristic was not used for graph generation.

[10] Distributed trust computation complicates our speed-up heuristic. To adapt, every user would need to store his/her own snapshots and skip values. This is not a feature we explore.

[11] Such ‘strains’ on the static nature are not difficult to overcome, but they would massively increase trace file size. Comparability must be preserved. For example, stating a user will be present *x*% of the time, then letting this presence be determined randomly at runtime is insufficient. Instead, every user entry-to/exit-from the network needs written to the trace.