A Testbed for Evaluating Computational Trust Models

by

Partheeban Chandrasekaran

A Thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfilment of

the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for

Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada

July 2012

Your file Votre référence
ISBN: 978-0-494-93484-5

Our file Notre référence
ISBN: 978-0-494-93484-5

Abstract

Trust is increasingly being used as a mechanism in many online applications to defend against malicious uses, such as free riding, spamming, cheating, etc. Some of these systems use social trust models to distinguish honest users from malicious users. However, a lack of a universal model makes evaluation of these social trust models difficult.

In this work, we propose a generic testbed for evaluating social trust models and we show how existing models can fit our tesbed.

To showcase the flexibility of our testbed design, we have implemented a prototype and evaluated three trust algorithms, namely EigenTrust, PeerTrust and Appleseed, for their vulnerabilites to attacks. As a result, we are able to confirm previously known vulnerabilities in trust algorithms and also discover new ones such as vulnerabilities to white-washing and normalization-based attacks in PeerTrust, and bootstrapping-based attacks in EigenTrust.

To my parents, whose dreams were abruptly halted...

Acknowledgments

First and foremost, I would like to thank my supervisor, Dr. Babak Esfandiari, for his iterative approach that made this work possible. His guidance and feedback were invaluable and contributed not only to this thesis but also to my personal growth. I would also like to thank Vicky Bell's help in editing this thesis.

I am indebted to my family and my friends for their support and encouragement. In particular, I would like to thank my beloved, Lydia, for her help during the final stages of this thesis. Last but not least, I would like to express my gratitude to my company, Entrust, for their financial assistance.

Table of Contents

List of Tables

List of Figures

List of Equations

Chapter 1

Introduction

1.1 Motivation

With the growth of online community-based systems such as peer-to-peer file-sharing systems, commercial websites, and social network websites, there is an increasing burden on such systems to provide quality data and goods. To do so, they may use computational trust models to determine which users or agents are honest and which ones are malicious. Some computational trust models rely on public key infrastructure (PKI)-based approaches, where trust is asserted using certificates. More recently, there have been many trust models that support building trust over a period of interactions. These trust models use metrics that range from the simple averaging of ratings on eBay to flow-based scores in the Advogato website to Facebook "friending". Thus, for a researcher to evaluate and compare his or her latest model against existing ones, a comprehensive test tool is needed. However, our research shows that the tools that exist to assist researchers are not flexible enough to include different trust models and their evaluations. Moreover, these tools use their own set of application-dependent metrics to evaluate a reputation system. This means that a number of trust models cannot be evaluated for vulnerabilites against certain types of attacks. Thus, there is still a need for a generic testbed to computation trust models.

1.2 Overview of Our Solution and Contributions

Our solution to the above-mentioned problem is a testbed to evaluate trust algorithms that implement a model that takes a graph as input and outputs another graph. These graphs can then be chained with other algorithms, including evaluation algorithms.

Our first contribution is to show that by viewing trust as a process occurring in stages, existing reputation systems can be adopted under a single model, but they work at different stages of the trust workflow. This allows us to present a new classification scheme for trust models based on where they fit in the trust workflow. The second contribution of our work is that this workflow can be described using Petri nets, and by doing this, we show that it is possible to model a variety of evaluation schemes. Finally, we built and used our testbed to evaluate three reputation systems against known attacks. We then present our assessments of vulnerabilities to these attacks. In particular, we confirmed the following known vulnerabilities:

- Vulnerability due to a priori trust in EigenTrust
- Vulnerability from attacks due to indifference to negative feedbacks versus no feedbacks in EigenTrust
- Attack resistance properties of Appleseed

We also found the following new vulnerabilities:

- Vulnerability in EigenTrust due to its bootstrapping method
- Vulnerability in PeerTrust due to its normalization method
- Vulnerability in PeerTrust to white-washing attacks

1.3 Organization

This thesis is organized as follows:

- Chapter 2 provides background and state of the art on trust models, attacks against them, and existing testbeds for evaluation.

- Chapter 3 formulates the research problem of this thesis and proposes our model for a testbed.

- Chapter 4 describes the implementation details of our testbed prototype and presents evaluation results of three different trust algorithms, namely EigenTrust, PeerTrust, and Appleseed.

- Chapter 5 concludes this thesis and summarizes the contributions and limitations of our work.

Chapter 2

State of the Art

Trust can be used as a tool for rational decision-making, and lack of trust can cause society to malfunction. Trust in itself is such a broad concept, reaching many fields—such as sociology, philosophy, psychology and commerce—that it cannot be used in computational societies such as Peer-to-Peer (P2P) file-sharing networks, blogging websites, and e-commerce systems without narrowing down its scope.

In this chapter, we provide definitions and properties of trust, an introduction to trust models, an in-depth analysis of the state of the art on trust models, attacks against them, and testbeds used to evaluate them.

2.1 Definitions of Trust

Our use of computational trust (or, simply, trust, in this work) adheres to the following definitions.

Castelfranchi et al. [1] define trust between agents as: "An agent i trusts agent j to achieve a goal by performing an action α if and only if:

1. i wants to achieve the specified goal

2. i expects that:

- j has the opportunity to ensure the goal by performing α and
- j is willing to do α and
- the internal preconditions for execution of α by j hold and
- the external preconditions for execution of α by j hold"

Meanwhile, Marsh [2] defines trust in agent i by another agent j in a particular situation as a function of utility of the situation to j, importance of the situation to j, and i's previous trust-based knowledge of j. The situation refers to the context of the action and the utility of the situation is a measurable estimate of how much an agent gains by participating in an action, whereas the importance of the situation is a "an agent-centered or subjective judgement of a situation" [2].

Both Casterfranchi et al.'s and Marsh's definitions agree that trust is contextual and subjective, but a key difference between them is that, using Marsh's definition, we can "measure" trust. Thus, trustworthiness is defined as the degree to which an agent is trusted.

Gambetta [3] defines trust as "subject probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends." Combining this definition with Marsh's, we will view that trustworthiness as the probability of a favourable outcome of a future interaction.

2.1.1 Reputation

There is a difference between trust and reputation. Wang and Vassileva [4] define trust as "a belief in another's peer's capabilities, honesty and reliablity based on its own direct experiences" whereas reputation as "a peer's belief in another peer's capabilities, honest and reliability based on recommendations received from other peers". This view of reputation is further supported by Abdul-Rahman's definition as follows [5]:

"A reputation is an expectation about an agents behaviour based on information about or observations of its past behaviour."

In this work, we view reputation as trustworthiness in an agent gained through both direct and indirect interactions.

2.2 Properties of Trust

Some of the properties of trust are that it is *dynamic*, *transitive* (to some extent), and *composable* [6]. In this section, we explain these properties.

2.2.1 Dynamic Property

According to Marsh [2], "When considering a simple trusting agent (i.e., one who does use rules of reciprocation), the trust he has in a trustee will ordinarily increase if cooperation occurs, and decrease otherwise." That is, if trust exists between two parties, then cooperation between them reinforces trust by increasing some trust score. If there was no trust, then cooperation builds trust. Furthermore, "a sudden defection from a trusted friend can result in a drastic reduction of trust, to the extent that a lot of work is necessary to build that trust up again" [2]. That is, if cooperation does not ensue, then trust is lost at a rate higher that it was gained.

Using this property, one can evaluate the trustworthiness of an agent based on other agents' past interactions with it.

2.2.2 Transitive Property

Trust can be seen as a flow that propagates between agents. However, it is not perfectly transitive, as shown in [7, 8]. That is, if A trusts B and B trusts C, it is not necessarily true that A would trust C [9, 10]. Furthermore, trust degrades as the

length of the trust path increases. That is, assuming trustworthiness is a non-binary measure, if A trusts B by x amount, B trusts C by y amount, and C trusts D by z amount, then A would trust C more than D, but this may not be sufficient in both cases to actually perform an action.

2.2.3 Composable Property

In both real and virtual societies, it is possible that there may be more than one trust path between any two agents. In this case, one can aggregate these paths' to derive a single trust score [2].

Using these properties as guidelines, trust models can be formulated to promote good and honest behaviour in a multi-agent society. In the next section, we will look at a few of these models.

2.3 Trust Models

Trust management systems aid agents to establish and assess mutual trust. However, the actual mechanism used in these systems vary. For example, in Public Key Infrastructure (PKI), certificates are used whereas in reputation-based trust management systems, experiences of earlier direct and indirect interactions are used [5]. Thus, the underlying models of these systems vary. Nevertheless, there are some general requirements for trust models, which are listed below [1]:

1. The trust model should aid agents to achieve their goals. That is, when an agent i trusts agent j, i expects that j has the "right" properties to fulfill i's goal. For example, in a P2P file-sharing network, this requirement can mean that i believes that j is correct in its claim that it has a file of interest to i (and not malware, for example).

2. The trust model should be able to specify the agent properties (abilities, willingness, etc.) that a truster can use for trust decision. However, most trust models in the literature simplify to one property, which is the agent's ability.

3. The trust model should provide a means to compare the trustworthiness of agents in order to choose a particular agent to perform an action. For instance, on an e-commerce website like eBay, we need to be able to compare the trustworthiness of sellers in order to pick the most trustworthy one to buy a product from.

In the rest of this section, we provide a brief description of the public key infrastructure (PKI)-based trust model and social trust models. The PKI trust model is often the most chosen in the security world. It does not depend on history, but on authority, and leverages the strength of cryptographic algorithms to provide trust among agents. Meanwhile, the social trust models are dependent on the agent's history of behaviour in the system.

2.3.1 The Public Key Infrastructure Trust Model

The PKI trust model is predominantly used in enterprises and banks to secure email, transactions and for physical and logical access. It achieves the aforementioned by providing confidentiality, integrity, and non-repudiation of data to its users.

The PKI trust models typically fall into Certificate Authority (CA)-based or Pretty Good Privacy (PGP)-based [11]. A CA, acting as a pre-trusted third party, issues certificates to its users, thereby establishing a trust relationship. On the other hand, PGP trust models do not have a central trusted party to issue certificates. Instead, users issues certificates to each other to attest to their trust.

A major limitation in CA-based PKI trust models is that they have a high cost for maintenance, required administration, and hardware. Also, these models do not

maintain anonymity, which may be preferred in many virtual environments. But more importantly, there is often no real-world relationship to begin with. How do we bootstrap the trust process in a virtual world?

Social trust models focus on solving these problems. In the following section, we introduce concepts in existing social trust models.

2.3.2 Social Trust Models

Instead of a strict process-oriented means of getting a certificate, social trust models rely on past experiences of agents to produce trust assertions. That is, the agents in the system interact with each other and record their experiences, which are then used to determine whether a particular agent is trustworthy. This model is self-sufficient because it does not rely on a third party to propagate trust, like in CA-based PKI trust models. However, there are drawbacks to having no root of trust. For instance, agents evaluating the trustworthiness of agents with whom there has been no interaction must use recommendations from others and, in turn, evaluate the trustworthiness of the recommenders. Social trust models must address this problem.

According to the definitions of trust in Section 2.1, we know that trust can be measured. Social trust models use algorithms [1] to measure the trustworthiness of agents, but their inputs vary. Examples of input are satisfaction ratings between agents [10] [12] [5], certifications among agents [13], trust ratings of agents [14] [8], and votes on exchanged goods based on authenticity and satisfaction [15]. Note that trust ratings differ from satisfaction ratings. Satisfaction ratings are ratings provided after each transaction, whereas trust ratings represent an aggregated measure for trustworthiness.

[1] We refer to these algorithms as trust algorithms. In the rest of the thesis, the terms "trust algorithms" and "social trust models" are interchangeable.

In PeerTrust [12] and EigenTrust [10], agents rate their satisfaction after a transaction (e.g., downloading a file in a P2P file-sharing network). These ratings are used to obtain a trust score that represents the trustworthiness of the agent. In ManagingTrust [5], agents may file complaints (can be seen as dissatisfaction) about each other after a transaction, whereas in Advogato [13], whose goal is to discourage spam on its blogging website, users explicitly certify each other as belonging to a particular level in the community. Continuous trust ratings are used in [8] and AppleSeed [14]. In the specific context of P2P file-sharing, Credence [15] uses the votes on file authenticity to calculate a similarity score between agents and uses it to measure trust. The trust score is then used to recommend files.

The truster may use some or all of its own and other agents' past experiences with the trustee to obtain a trust score. Trust algorithms often use gossip to poll agents with whom the truster has had interactions in the past. In the example shown in Figure 2.1, agent A needs to calculate the trust score of E, but has had no interactions with it. The dotted arcs between the agents represent interactions between agents in the past, and solid arcs represent the flow of recommendations.

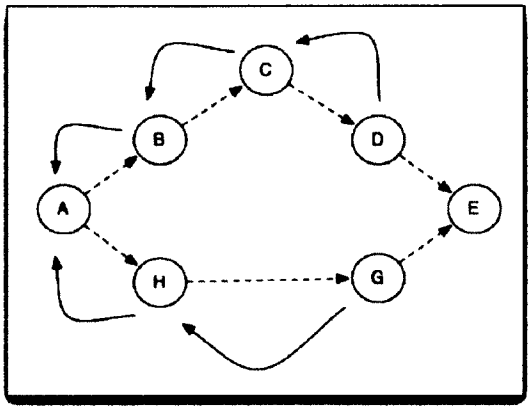


Figure 2.1: Recommendation polling

The trust score calculated using only the experiences from direct interactions is

called the *direct* trust score, while the trust score calculated using the recommendations from other agents is called the *indirect* trust score [16]. As mentioned earlier, reputation systems use different input (satisfaction ratings, votes, certificates, etc.) to calculate direct trust scores and indirect trust scores. PeerTrust uses satisfaction ratings to calculate both direct and indirect trust scores, whereas EigenTrust uses satisfaction ratings to calculate direct trust scores, but uses direct trust scores to calculate indirect trust scores. Therefore, we can categorize the trust algorithms based on the input required. But how do trust algorithms calculate the trust scores of agents using the above information? It again varies from algorithm to algorithm. For instance, PeerTrust, EigenTrust, and ManagingTrust use simple averaging of ratings, whereas Appleseed [14] is based on the Spreading Activation model. For example, PeerTrust calculates the trust score of an agent by first collecting the recommendations by other agents and weighing them by the trust scores of the recommenders, which we will describe in detail further in this thesis.

Depending on the trust algorithm, it may output a global trust score or a local trust score. A global trust score is one that represents the general trust that all agents have on a particular agent, whereas local trust scores represents the trust from the perspective of the truster [14,17] and thus each truster may trust an agent differently. In our survey, we found PeerTrust, EigenTrust, and ManagingTrust to be global trust algorithms whereas Credence [15], Advogato [13], TidalTrust [8], Appleseed [14], Marsh's [2] and Abdul-Rahman's [8] to be local trust algorithms.

Once the trust score is calculated, it can be used to decide whether to trust the agent. It can be as simple as a comparing the trust score against a threshold. If the trust score is above a certain threshold, then the agent is trusted. Marsh [2], and ManagingTrust [5] use thresholding techniques. If the trust algorithm outputs normalized trust scores of agents as in EigenTrust [10], then the trust scores of agents are ranked. In this case, one may consider a certain percentage of the top ranked

agents as trustworthy. In Appleseed [14], a graph is first obtained with trust scores of agents as edge weights, and then, the truster agent is "injected" with a value called the activation energy. This energy is spread to agents with a spreading factor along the edges in the graph and the algorithm ranks the agents according to their trust scores. Trust decisions can also be flow-based such as in Advogato [13], which calculates a maximum "flow of trust" in the trust graph to determine which agents are trustworthy and which are not.

In short, social trust models focus on the following:

1. What is the input to calculate the trust score of an agent?
2. How is the trust score of an agent represented?
3. Is the trust score of an agent global or local?
4. How does one decide whether to trust an agent or not?

Table 2.1 summarizes how each trust model we surveyed answers the above questions. The next sections in this chapter provide detailed descriptions of the trust models surveyed (so that a more general and abstract model of them can be devised for benchmarking purposes), attacks against them that prevent them from achieving their goals (again, so that those attacks can be modeled in our benchmarking tool), and existing testbeds that attempt to evaluate such algorithms.

2.3.2.1 PeerTrust

In PeerTrust, agents rate each other in terms of the satisfaction received. These ratings are weighted by trust scores of the raters and a global trust score is computed recursively using Equation 2.3.1, where:

- $T(u)$ is the trust score of agent u

Trust Algorithm	Dir. Trust Scr.	Indir. Trust Scr.	Trust Score Rep.	Global or Local	Trust Decision
EigenTrust [10]	Sat. ratings	Trust Scores	Continuous, (0, +1)	Global	Rank-based
PeerTrust [12]	Sat. ratings	Sat. ratings	Continuous, (0, +1)	Global	N/A
ManagingTrust [5]	Sat. ratings	Sat. ratings	Discrete, [-1, +1]	Global	Threshold-based
Abdul-Rahman [18]	Sat. ratings	Sat. ratings	Discrete, [1,4]	Local	N/A
Credence [15]	Votes on files	Sat. ratings	Continuous, [-1,1]	Local	N/A
Advogato [13]	Certifications	Certifications	Discrete, [0,1]	Local	Flow-based
TidalTrust [8]	Trust Scores	Trust Scores	Discrete (1,10)	Local	N/A
Appleseed [14]	Trust Scores	Trust Scores	Continuous, [0,1]	Local	Rank based
Marsh [2]	Sat. ratings	N/A	Continuous, [-1, +1)	Local	Threshold-based

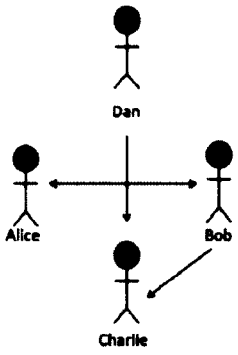
Table 2.1: Classification of trust models surveyed

Note: Dir. = Direct; Indir. = Indirect; Scr. = Score; Sat. = Satisfaction; Rep. = Representation; N/A = Not Applicable

- $I(u)$ is the set of transactions that agent u had with all the agents in the system
- $S(u,i)$ is the satisfaction rating on u for transaction i
- $p(u,i)$ is the agent that provided the rating.

$$T(u) = \sum_{i=1}^{I(u)} S(u,i) * \frac{T(p(u,i))}{\sum_{j=1}^{I(u)} T(p(u,j))} \tag{2.3.1}$$

Consider the example shown in Figure 2.2. Equation 2.3.1 (note that it is recursive) is initialized by allowing all agents to be trusted equally (in this case, $\frac{1}{4}$).



Id	Source	Sink	Satisfaction
0	Dan	Charlie	0.7
1	Dan	Charlie	0.8
2	Dan	Charlie	0.9
3	Alice	Bob	0.8
4	Alice	Bob	0.9
5	Bob	Charlie	0.5
6	Bob	Charlie	0.4
7	Bob	Charlie	0.2
8	Bob	Alice	0.9
9	Bob	Alice	0.8

Figure 2.2: PeerTrust—Agent ratings

Using Equation 2.3.1, T(Alice) is calculated as 0.9 * $\frac{T(Bob)}{T(Bob)+T(Bob)}$ + 0.8 * $\frac{T(Bob)}{T(Bob)+T(Bob)}$ and because T(Bob) is initialized to 0.25, T(Alice) = 0.85. T(Bob) is calculated similarly, but because we have already calculated T(Alice), T(Bob) = 0.8 * 0.85/(0.85+0.85) + 0.7 * 0.85/(0.85+0.85) = 0.75. T(Charlie) is (0.7 + 0.8 + 0.9) * $\frac{T(Dan)}{3*T(Dan)+3*T(Bob)}$ + (0.5 + 0.4 + 0.2) * $\frac{T(Dan)}{3*T(Dan)+3*T(Bob)}$, which is equal to 0.475. Because no agent has interacted with Dan, T(Dan) = 0.25.

We note that in PeerTrust the order of trust score calculation of agents is important. That is, if we were to calculate T(Charlie) first instead of T(Alice) and T(Bob), we would end up with a different trust score for Charlie.

PeerTrust also provides a method for calculating the local trust scores. In it, the satisfaction ratings are weighted by a similarity score that is obtained by comparing the ratings history by the truster and the rating histories by others in a common set of agents. Once the similarity score is obtained, the trust score is calculated using Equation 2.3.2, where $IJS(v,w)$ is the common set of agents on which both v and w provided ratings on, and $I(v,w)$ is the total number of ratings that v provided in w.

$$T(u,w) = \sum_{i=1}^{I(u)} S(u,i) * \frac{Sim(p(u,i),w)}{\sum_{j=1}^{I(u)} Sim(p(u,j),w)}$$

$$where \quad Sim(v,w) = 1 - \sqrt{\frac{\sum_{x \in IJS(v,w)} ((\frac{\sum_{i=1}^{I(x,v)} S(x,i)}{I(x,v)} - \frac{\sum_{i=1}^{I(x,w)} S(x,i)}{I(x,w)})^2}{|IJS(v,w)|}} \qquad (2.3.2)$$

Consider again the example in Figure 2.2. $IJS(Bob, Dan)$ = {Charlie}, $IJS(Alice, Dan)$ = {}, $IJS(Charlie, Dan)$ = {}. $I(Charlie, Dan)$ = 3. $I(Charlie, Bob)$ = 3. Thus $Sim(Bob, Dan)$ = 1 - $\sqrt{}$(((0.5+0.4+0.2)/3)−(0.7+0.8+0.9)/3))^2)/1 = 1- 0.43 = 0.56. In this case, as Bob is the only one who interacted with Alice $T(Alice, Dan)$ = 0.9*0.56+0.8*0.56 = 0.95.

2.3.2.2 EigenTrust

Agents in EigenTrust rate transactions as satisfactory or unsatisfactory [10]. These transaction ratings are used as input, to calculate a local trust score, from which a global trust score is then calculated.

An agent i calculates the normalized local trust score of agent j, as shown in Equation 2.3.3, where $t_{ij} \in \{+1, -1\}$ is the transaction rating, and s_{ij} is the sum of ratings.

$$c_{ij} = \frac{max(s_{ij}, 0)}{\sum_k max(s_{ik}, 0)}$$

$$s_{ij} = \sum_{T_{ij}} tr_{ij} \tag{2.3.3}$$

If an agent has no transactions with other agents (essentially a bootstrapping problem), then C_{ij} will be undefined because the denominator in Equation 2.3.3 will be 0. This is addressed by making use of pre-trusted agents in the system. The pre-trusted agents have a trust score of p_i such that $p_i = \frac{1}{|P|}$ if agent $i \in P$ or 0 otherwise. Thus we can rewrite Equation 2.3.3 as Equation 2.3.8.

$$c_{ij} = \begin{cases} \frac{max(s_{ij},0)}{\sum_j max(s_{ij})}, & \text{if } \sum_j max(s_{ij}, 0) \neq 0 \\ p_j, & \text{otherwise} \end{cases} \tag{2.3.4}$$

Note that we cannot use s_{ij} as the local trust score without normalizing, because malicious agents can arbitrarily assign high local trust values to fellow malicious agents and low local trust values to honest agents. For example, suppose agent i assigns a local trust score of 100 to a malicious agent k. If an honest agent j requests i to provide its local trust value of k, the aggregated trust value in k will be larger, making the system ineffective.

To calculate the global trust score of an agent, the truster queries his friends for their trust scores on the trustee. These local trust scores are aggregated as shown in Equation 2.3.5.

$$t_{ik} = \sum_j c_{ij} c_{jk} \tag{2.3.5}$$

If we let C be the matrix containing c_{ij} elements, $\vec{c_i}$ be the local trust vector for i (each element corresponds to the trust that i has in j), and $\vec{t_i}$ vector containing t_{ik}, then,

$$\vec{t_i} = C^T \vec{c_i} \tag{2.3.6}$$

By asking a friend's friend opinion, Equation 2.3.6 becomes $\vec{t_i} = (C_T)^2 \vec{c_i}$. If an agent keeps asking the opinions of its friends of friends, the whole trust graph can be explored, and Equation 2.3.6 becomes Equation 2.3.7, where n is the number of hops from i.

$$\vec{t} = (C^T)^n \vec{c_i} \tag{2.3.7}$$

The trust scores of the agents converge to a global value irrespective of the trustee. This convergence can be also interpreted using the Markov chain model. If U is the starting state matrix, U_n is the ending state matrix after n steps and P is the transition matrix in a Markov chain model, then $U_n = U.P^n$. So $\vec{c_i}$ can be seen as the start matrix and C as the transition matrix in Equation 2.3.7. For U to converge where all rows are the same vector w and each component in w is positive and the sum of the components is 1, it must be a regular matrix (i.e., some power of the matrix must contain only positive elements).

Pre-trusted agents can also be used for two reasons: for faster convergence of $\vec{t}$ and to break away from malicious collectives. That is, $\vec{t} = (C^T)^n \vec{p}$ converges faster than $\vec{t} = (C^T)^n \vec{c_i}$, and by enforcing each agent to place some trust in a pre-trusted agent while aggregating local trust values, we can break away from malicious collectives. This is shown in Equation 2.3.8, where $a \in \{0, 1\}$, and k represents the iteration.

$$\vec{t}^{(k+1)} = (1 - a) C^T \vec{t}^k + a\vec{p} \tag{2.3.8}$$

Drawing an analogy with the Random Walk theory, if an agent is performing a random walk, in the system it is less likely to get stuck in a group of agents because the probability of "walking to" a pre-trusted agent is not 0 and thereby can break away from a malicious group.

Consider the example in Figure 2.2 and assume the following:

1. A transaction is satisfactory (+1) if its rating ≥ 0.7 and unsatisfactory (-1) otherwise

2. Pre-trusted agents, $P = \{\text{alice, bob, charlie, dan}\}$

We can represent s_{ij} as $\begin{bmatrix} 0 & 2 & 0 & 0 \\ 2 & 0 & -3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 \end{bmatrix}$ where the indices of Alice, Bob, Charlie and Dan are 1, 2, 3 and 4 respectively. After normalizing using Equation 2.3.4, matrix C can be represented as $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0 & 0 & 1 & 0 \end{bmatrix}$. If we arbitrarily set a to be 0.2 and initialize $\vec{t^0} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$ in Equation 2.3.8, then after 100 iterations, we obtain $t^{100} =$

Agent	PeerTrust	EigenTrust
Alice	0.33	0.39
Bob	0.29	0.39
Charlie	0.18	0.14
Dan	0.19	0.08

Table 2.2: Agent ranks—PeerTrust and EigenTrust

$$\begin{bmatrix} 0.39 \\ 0.39 \\ 0.14 \\ 0.08 \end{bmatrix}.$$

Because EigenTrust outputs normalized trust scores (normalized over the sum of all agents), agents are ranked according to their trust scores (unlike PeerTrust). Therefore, an agent is considered trustworthy if it is within a certain rank.

Note that if we also normalize the trust scores obtained using PeerTrust, we obtain a different ranking (see Table 2.2). The ranks of the agents output by PeerTrust and EigenTrust differ. We know that the net satisfactory transactions that agents had with Alice and Bob is 2, and with Charlie and Dan is 0. Thus, intuitively, we expect the trust scores of Alice and Bob to be the same and higher than those of Charlie and Dan. Thus, we conclude that EigenTrust fits our intuition better, whereas PeerTrust does not because Alice is more trustworthy than Bob, as per PeerTrust. This is the type of comparison that a benchmarking tool should support.

2.3.2.3 ManagingTrust

While PeerTrust and EigenTrust use satisfaction ratings of transactions to calculate the trust score of an agent, ManagingTrust [5] uses complaints. If agent p is not satisfied with q following a transaction, then p files a complaint against q. In a system that uses complaints, an agent may in turn complain against the complainant to hide its malicious behaviour. However, if the agent continues its malicious behaviour, then it is likely that other agents will also complain against it and it will eventually be classified as untrustworthy. Using this information, the authors in [5] calculate the global trust score of an agent p using Equation 2.3.9, where P is the set of agents in the system, and $c(p,q)$ is a complaint on q by p.

$$T(p) = |\{c(p,q)|q \in P\}| \times |\{c(q,p)|q \in P\}| \tag{2.3.9}$$

Complaints can be seen as results of unsatisfactory transactions and therefore higher values of $T(p)$ obtained using Equation 2.3.9 indicate less trustworthiness.

A drawback of calculating trust scores in this fashion is that an agent's untrustworthiness increases as the number of complaints that it makes increases, even though if it may have received no complaints. This contradicts our common intuition about trust; the number of bad transactions should not affect the truster's trust score. Furthermore, if $|c(p,q)| \geq |c(q,p)|$, then another agent r would not be able to accurately identify p as the malicious one.

Once the complaints are collected, then the number of complaints received about q and filed by q are normalized (indexed by the arbitrary agent i) as shown in Equations 2.3.11 and 2.3.11, respectively, where $s = \sum_1^w f_i$, w is the number of different witnesses found, $cr_i(q)$ and $cf_i(q)$ are complaints received and filed by q, respectively, and f_i is the number of times agent i is found in the result.

$$cr_i^{norm}(q) = cr_i(q)(1 - \frac{(s - f_i)^s}{s}), i = 1....w \tag{2.3.10}$$

$$cf_i^{norm}(q) = cf_i(q)(1 - \frac{(s - f_i)^s}{s}), i = 1....w \tag{2.3.11}$$

Each time p collects complaints and computes $cr_i^{norm}(q)$ and $cf_i^{norm}(q)$, it also maintains averages of cr_i and cf_i obtained from each arbitrary agent i as cr_p^{avg} and cf_p^{avg} respectively. The decision to trust q is determined as per Equation 2.3.12.

$$trusts(p,q) = \begin{cases} 1, & cr_i^{norm}(q).cf_i^{norm}(q) \leq (0.5 + \frac{4}{\sqrt{cr_p^{avg} cf_p^{avg}}})^2 cr_p^{avg} cf_p^{avg} \\ 0, & \text{otherwise} \end{cases} \tag{2.3.12}$$

For each witness agent, the above boolean function is evaluated. Thus each witness agent might evaluate q differently and p still needs to aggregate these boolean function return values and determine if q is trustworthy. One simple method is to sum the return values of $trusts(p,q)$, and if it is greater than 0 or some other threshold, then consider it to be trustworthy, else untrustworthy. If it is 0, then it is undecided.

Note that ManagingTrust does not use the Friend of a Friend (FOAF) principles like EigenTrust.

2.3.2.4 Advogato

The Advogato algorithm is used in a blogging community called by the same name. The website promotes blogs on open-source discussions and material. The primary uses of the Advagato algorithm are to prevent spammers on the website and to discourage people from posting non-open-source-related material, and this, without using a central authority. However, unlike PeerTrust, EigenTrust, and ManagingTrust, Advogato takes a flow-based approach to determine whether an agent is trustworthy

or not [13]. Trust is seen as a current that flows from the truster to the trustee, and more of this flow means more trust in the trustee. Advogato extends this relation to determine trust in a manner that is resistant to Sybil attacks (see Section 2.4.8). It does so by ensuring that the amount of damage from introducing attacks scales smoothly with the cost of the attack.

Each member can certify another as someone belonging to the opensource community (however, the certifying member should also be certified). The certificates are used to form a certification graph. If V is the set of agent (vertices) and E is the set of edges in the certification graph G, then each edge represents trust between the source agent and the sink agent. That is, for $s, t \in V$:

$$(s,t) \in E \Rightarrow trust(s,t) \tag{2.3.13}$$

Each agent has a capacity assigned to it, which determines the maximum trust flow that can go through it.

$$C : V \rightarrow \mathbb{N} \tag{2.3.14}$$

In Advogato, some agents are considered seeds (inherently trustworthy). The capacity of each agent is calculated relative to the seed and the capacity of the seed itself is set to the total number of agents that need to be trusted/accepted.

The graph is then traversed from the seed. Agents accessed through the edges are said to be one level from the originating seed. The capacity of the agents at a particular level l, is assigned using the formula given below:

$$C(l) = \frac{C(v_{l-1})}{\hat{d}(l-1)} \tag{2.3.15}$$

where:

- l ($\geq$ 1) is the current level (C(v_l) is the capacity of the nodes at level l). Note that the seed's level is 0.

- $\hat{d}$(l) is the average outdegree from level l, which is calculated as the number of outgoing edges from level l divided by the number of nodes in that level.

We now have a graph, $G = (V, E, C)$. The maximum trust flow in this graph is then calculated, using algorithms such as Ford-Fulkerson. To do so, the graph is transformed from single-source multiple-sinks to single source single sink (also known as supersink).

Once the maximum flow through the transformed graph is calculated, an agent is trusted if there is at least a flow of 1 unit from that agent to the supersink.

An important contribution of Advogato is that it is resistant to Sybil attacks (see Section 2.4.8) [19]. This is because the number of bad agents accepted as trustworthy is bounded by $\sum_{v' \in V'}(C_{v'} - 1)$, where each V' is the set of "good" agents with edges to bad agents.

Let us use an example to illustrate Advogato. Consider the initial certification graph given in Figure 2.3. Capacities are assigned to the nodes using Equation 2.3.15 (see Figure 2.4), after which the graph is transformed to have a single source and a single sink and the node capacities are moved to edge capacities. We can do this by splitting a node into 2 and a unit flow for the edge between the second split node (negative) and a dummy node (supersink) if the capacity of the node is greater than 1. The capacity of the edge between the 2 split nodes is the capacity of the node -1. Details of the algorithm can be found in [13].

Figure 2.5 illustrates that because there is a flow from the negative nodes of Alice, Bob, Charlie and Dan to the supersink, they are considered trustworthy.

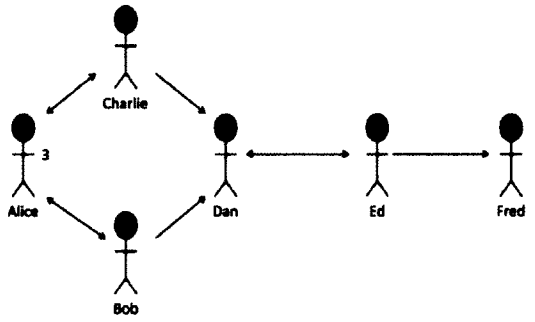


Figure 2.3: Initial graph with *Alice* as the seed and its capacity set to 3

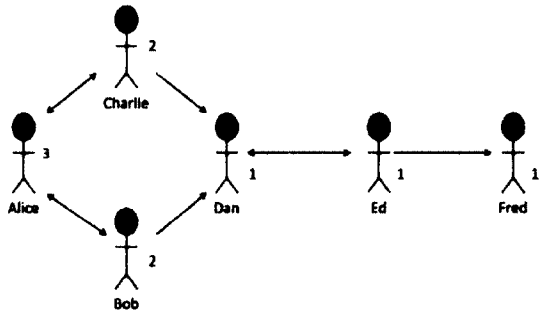


Figure 2.4: Initial graph with capacities assigned to each node

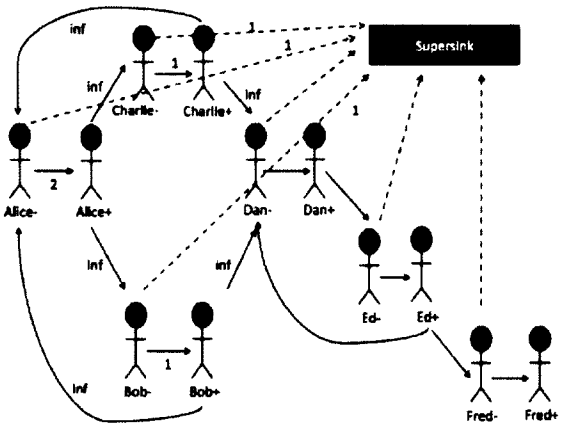


Figure 2.5: Transformed graph: flow from Alice to supersink

2.3.2.5 Appleseed

Like Advogato, Appleseed is also a flow-based algorithm [14]. Instead of determining a boolean trust, it calculates a trust score. Assuming that we are given a directed weighted graph with agents as nodes, edges as trust relationships, and weight of an edge as trustworthiness of the sink, we can determine the amount of trust that flows in the graph. That is, given a trust seed, an energy $in \in \mathbb{R}_0^+$, spreading factor $decay \in [0, 1]$, and convergence threshold T_c, Appleseed returns a trust score of agents from the perspective of the trust seed, as shown in Equation 2.3.16.

$$A \times \mathbb{R}_0^+ \times [0, 1] \times \mathbb{R}^+ \rightarrow (trust : A \rightarrow \mathbb{R}_0^+) \tag{2.3.16}$$

The trust propagation from agent a to agent b is determined using Equation 2.3.17, where the weight of edge (a, b) represents the amount of trust a places in b, and $in(a)$ and $in(b)$ represent the flow of trust into a and b, respectively.

$$in(b) = decay \times \sum_{(a,b)\in E} in(a) \times \frac{weight(a, b)}{\sum_{(a,c)\in E} weight(a, c)} \tag{2.3.17}$$

The trust of an agent b ($trust(b)$) is then updated using Equation 2.3.18, where the decay factor ensures that trust in an agent decreases as the path length from the seed increases.

$$trust(b) := trust(b) + (1 - decay) \times in(b) \tag{2.3.18}$$

Generally trust graphs have loops, which makes Equation 2.3.18 recursive. Thus a termination condition like the one below is required, where $A_i \subseteq A$ is the set of nodes that were discovered until step i and $trust_i(x)$ is the current trust scores for all $x \in A_i$:

$$\forall x \in A_i : trust_i(x) - trust_{i-1}(x) \leq T_c \tag{2.3.19}$$

After Equation 2.3.18 terminates, the trust scores of agents are ranked. Since this set is ranked from the perspective of the seed, Appleseed is a local trust algorithm.

As our survey shows, the trust models vary in terms of their input, output, and the methods they use. In the next section, we investigate various attacks against social trust models.

2.4 Attacks

The performance of trust models greatly depends on their ability to detect attacks by malicious agents. In addition, it is also important that the number of honest agents inaccurately identified as untrustworthy (false positives) is kept to a minimum, to prevent denial-of-service attacks, for instance.

The malicious agents' goal may vary from promoting a virus in a P2P file-sharing system to fooling users on eBay into buying a cheap quality product. Fake online reviews are also common, and at times, they have serious real-world implications.

While the method used by malicious agents is dependent on the target application itself, there are attack strategies that are common to many trust models. In this section, we introduce several of these attack strategies, based on [17, 20].

2.4.1 Self-Promoting

As we saw in Section 2.3.2, recommendations of agents play a pivotal role in social trust models. An agent may exploit the trust model by always recommending another agent that may also be under the attacker's control. The attacker may also adopt a strategy in which one agent behaves honestly with agents but highly recommends

another agent under its control, which caries out the malicious activities. Once the honest agents determine that the agent is malicious, the attacker introduces another agent. Trust models that use only positive recommendations about agents are prone to this attack [17]. To defend against this type of attack, the trust models must have a mechanism to detect the subtle difference between behaving honestly in direct interactions and behaving maliciously when recommending other agents.

2.4.2 White-Washing

An agent carrying white-washing attacks [17] cheats the system by behaving honestly to gain trust and then behaving dishonestly. Once the trust begins to decline, the malicious agent behaves honestly again and repeats this to avoid getting labelled as untrustworthy. Trust models need to be able to quickly detect such behaviour, to prevent this type of attack. Trust models that take into account the dynamic property of trust ("hard to gain, easy to lose") are more efficient in defending against such attacks.

Note that there seems to be no standard definition of this attack. For instance, [21] describes a white-washing attack as one where "an attacker rids himself of a bad reputation by rejoining the system with a new identity". However, we will use the former description of a white-washing attack.

2.4.3 Slandering

This type of attack occurs when an attacker falsely provides negative ratings about an agent that may be behaving honestly, in order to bring down the victim's trustworthiness [17]. This type of attack is hard to detect when only one agent attacks another. If an attacker forms a collusive group to target one agent, trust models may be able to detect that negative ratings about the victim agent come from a single

group of agents.

2.4.4 Bootstrapping

Depending on the trust assigned to newcomers, an attacker may exploit the system by simply creating multiple agents and quit as soon as the malicious behavior is detected by other agents. This type of attack is also called the Initial Window attack in [20].

2.4.5 Value Imbalance

If the trust model supports interactions that are of different value, to agents, then an attacker may behave honestly in low-cost interactions and occasionally cheat in a high-cost interaction. This type of attack, called the value imbalance attack [20], is particularly effective in e-commerce systems like eBay. To detect this attack, trust models must take into account the value of the interaction in trust updates.

2.4.6 Reputation Lag

The reputation lag attack takes advantage of the time between a transaction and the time a feedback is provided for that transaction, and allows an attacker to cheat the system during this time interval [20].

2.4.7 Proliferation

If there are multiple agents with the same reputation, then each agent has equal probability of being picked for a future transaction. An attacker may increase the probability of picking one of its agents in the system by introducing multiple agents. This is called the proliferation attack [20].

2.4.8 Sybil

The number of fake identities (malicious agents) or *sybils* that can be introduced by an attacker is dependent on the time and computational resources available to him [19]. It is practically impossible to prevent this type of attack in a multi-agent society without a central registration authority. Thus, at best, social trust models can only offer damage control by limiting the number of malicious agents accepted as trustworthy. This property of trust models is termed "attack resistance" by Levien in [22].

Often the Sybil attack is used in conjunction with the above-mentioned attacks to achieve a goal. For instance, the attacker may introduce Sybils to form a collusive group to mount a self-promoting attack or a slandering attack.

Thus, to evaluate trust models, one must implement the attack stategies mentioned in this section. The next chapter provides details about existing testbeds to evaluate social trust models.

2.5 Testbeds

We investigated a model for a testbed [23] and two testbed implementations, namely, Agent and Reputation Trust (ART) and Trust and Reputation Experimentation and Evaluation Testbed (TREET) [20], which are used to evaluate trust algorithms. This section provides details of our investigation.

2.5.1 Guha

Although Guha's work on rating systems [23] is not a testbed implementation, it is worth investigating as a model for a testbed. In this work, Guha introduces a model where given a set of objects O to be rated, a set of agents A, a set of possible values

of ratings of objects D, and a set of values for ratings of agents by agents T where $T \in \{positive, negative\}$, then the object rating function R and agent rating function W are given as follows:

$$R : A \times O \rightarrow D$$
$$W : A \times A \rightarrow T \tag{2.5.1}$$

By combining R and W, we get a directed graph with objects and agents as nodes, and arcs between agents, and arcs from agents to objects.

One disadvantage of modelling W as $(A \times A \rightarrow T)$ is that existing trust models that require sets of feedback values $(A \times A \rightarrow \mathbb{R}^*)$, such as PeerTrust (global) [12], EigenTrust [10] and ManagingTrust [5], cannot be accommodated.

Moreover, as per the author, if $W(A_i, A_j)$ is positive, then A_i trusts A_j. By assuming this, it excludes trust models such as AppleSeed [14] and TidalTrust [6] as they require $(A \times A \rightarrow \mathbb{R})$ as input.

2.5.2 ART

ART provides an open-source message-driven simulation engine for implementing and comparing performances of reputation systems. ART uses art paiting sales as the domain.

Each client has to sell paintings belonging to a particular era. To determine their market values, clients refer to agents for appraisals for a fee. Because each agent is an expert only in specific era, it may not be able to provide appraisals for paintings of other eras and therefore refers to other agents for a fee. After such interactions, agents record their experiences, calculate their reputation scores, and use them to choose the most trustworthy agents for future interactions. The goal of each agent is to finish the simulation with the highest bank balance, and, intuitively, the winning

agent's trust mechanism knows the right agents to trust for recommendations.

The ART testbed provides a protocol that each agent must implement. The protocol specifies the possible messages that agents can send to each other. The messages are delivered by the simulation engine, which loops over each agent at every time interval. The engine is also responsible for keeping track of the bank balance of the agents, and assigning new clients to agents. All results are collected and stored in a database and displayed on a graphical user interface (GUI) at runtime.

To evaluate the ART testbed, we implemented EigenTrust [10]. We present our observations below.

2.5.2.1 Implementing EigenTrust in ART

The problem statement defined in EigenTrust [10] is to reduce the number of inauthentic downloads by a peer in a P2P file-sharing community. We can say that the goal of EigenTrust is to increase the overall well-being of the community. However, the goal in ART is to increase the personal bank balance of an agent. That is, to increase the personal well-being of an agent. We therefore have two potentially conflicting goals. In order to integrate EigenTrust in ART, the concepts between the two need to be linked.

In EigenTrust, each agent maintains a matrix to store the local trust scores of other agents in the system. We can extrapolate these local trust scores to reputation values in ART, and appraisals in ART can be thought of as the act of downloading a given file from the network in the EigenTrust. However, the global trust score of an agent in EigenTrust has no equivalent in ART.

In Section 2.3.2.2, we presented the centralized version of the EigenTrust algorithm, but it cannot be used in ART as the simulation framework does not allow for any centralized solution. The authors of the EigenTrust algorithm also propose a distributed version to calculate the global trust scores. We encountered several

difficulties while implementing the distributed version of the EigenTrust algorithm, some of which are summarized below:

1. The distributed version of EigenTrust requires each agent to share its c_{ij} matrix and this implies that all agents must implement EigenTrust. This limits the ability to evaluate the competition between EigenTrust agents and non-EigenTrust agents.

2. The ART source code had to be changed to allow agents to calculate global trust scores.

3. Without major changes in ART's architecture, it is not possible to implement a Sybil attack, as agents cannot be introduced on the fly.

More generally, ART is suited for trust algorithms that have the following characteristics:

- The algorithm calculates direct and indrect trust scores without using FOAF principles. Algorithms such as EigenTrust which calculates indirect scores using FOAF principles, cannot be evaluated using ART.

- The algorithm must have a mechanism to aggregate past experiences into a trust score. This implies that ART cannot be used for evaluating algorithms such as Appleseed, and Advogato.

We conclude that the scope of ART is too narrow to allow the evaluation of many different trust algorithms.

2.5.3 TREET

TREET models a general marketplace scenario where there are buyers, sellers, and 1,000 different products with varying prices, such that there are more inexpensive

items than expensive ones. The sale price of the products is fixed, to avoid the influence of market competition. The cost of producing an item is 75% of the selling price, and the seller incurs this cost. To lower this cost and increase profit, a seller can cheat by not shipping the item. Each product also has a utility value of 110% of the selling price, which encourages buyers to purchase.

Agents join or exit after 100 simulation days or after a day with a probability of 0.05, but to keep the number of buyers and sellers constant, for each departing agent, another agent is introduced. At initialization, each seller is assigned a random number of products to sell. Buyers evaluate the offers from each seller and pick a seller. Sellers are informed of the accepted offers and are paid. Fourteen days after a sale, the buyer knows whether he has been cheated or not, depending on whether he receives the purchased item. The buyer then provides feedback based on his experience of the transaction. The feedback is in turn used to choose sellers for future transactions.

TREET [20] evaluates the performance of various reputation systems under Reputation Lag attack, Proliferation attack, and Value Imbalance attack using the following metrics:

1. cheater sales over honest sales ratio

2. cheater profit over honest profit ratio

As described in Section 2.4.7, multiple seller accounts are needed to orchestrate a Proliferation Attack, but authors in [20] do not consider attacks such as White-Washing and Self-Promoting, which require creating multiple buyer accounts.

TREET addresses many of ART's limitation in a marketplace scenario. To name a few [24], TREET supports both centralized and decentralized trust algorithms, allows collusion attacks to be implemented, and does not put a restriction on trust score representation. However, like ART, the evaluation metrics in TREET are tightly

coupled to the marketplace domain. It is unclear how ART or TREET can be used to evaluate trust models used in other systems, such as P2P file-sharing networks, online product review websites and others that use trust. To our knowledge, there is no testbed that provides generic evaluation metrics and that is independent of the application domain.

2.6 Summary

Trust is a tool used in decision-making process and it can be computed. There are many models based on social trust that attempt to aid agents to make rational decisions. However, these models vary in terms of their input and output requirements. This makes evaluations against a common set of attacks difficult. Testbeds such as ART and TREET can be used to evaluate some of the trust models, but their metrics are too ad hoc.

In the next chapter, we formulate the problem to be solved.

Chapter 3

A Model for a Testbed for Trust Models

3.1 Problem Description

In Chapter 2, we saw that there are many trust models in the literature that address the problem of computing trust in agent systems but they differ in the type of rating used (satisfaction ratings [10] [12], certifications [13] and trust ratings [6] [14]), in whether they calculate global or local trust and, finally, in how the rating is used to decide whether to trust an agent. This makes it challenging to compare and evaluate them against basic trust properties and also against attack scenarios such as self-promoting [17], white-washing [17], slandering [17], and introducing Sybils [19]. To analyze the performance of trust models, we need a testbed that is generic enough to accommodate as many reputation systems as possible. Our requirements are:

1. A model that provides an abstraction layer for developers to incorporate existing and new reputation systems that match the input and output of the model.

2. An evaluation framework to measure and compare the performance of trust models against trust properties and attacks independently of the application domain.

In the next section, we introduce an abstract model for reputation systems. This model will be the foundation of our testbed.

3.2 Model

Our model for a testbed is essentially based on the following concepts:

- Trust is a process that occurs in stages, and many existing trust models fit in various stages of the trust process (Section 3.3).
- Both trust models and their evaluations are viewed and represented in the testbed as parts of workflows using Petri net (Section 3.4).

In stage 1 of the trust process, the feedback provided by agents on other agents is represented as a feedback history graph. In stage 2, a reputation graph is produced, where the weight of an arc denotes the reputation of the target agent. In the final stage, a trust graph is produced, where the existence of an arc implies trust in the target agent.

In the rest of this section, we define the aforementioned graphs in stages.

3.2.1 Stage 1—Obtain Feedback History Graph

We first define a *feedback*, $f(a,b) \in \mathbb{R}$ as an assessment made by agent a, of an action or group of actions performed by agent b. The list of n feedbacks by a on b is called a feedback history, represented as follows:

$$H : A \times A \rightarrow \mathbb{R}^n$$
$$H(a,b) \mapsto \{f_1(a,b), f_2(a,b), ..., f_n(a,b)\} \tag{3.2.1}$$

The feedback $f(a,b)$ indicates the satisfaction received by a from b's action. For

example, in a file-sharing network, the feedback by a downloader may indicate the satisfaction received from downloading a file from an uploader in terms of a value in $\mathbb{R}$. As we saw in Table 2.1, existing trust models use different ranges of values for feedback, and letting the feedback value be in $\mathbb{R}$ allows us to include these reputation systems in our testbed.

The feedback histories for all agents in A are represented in a Feedback History Graph (FHG). It is formally defined as a directed and labelled graph, $FHG = (A, E)$, where A is the set of agents, E is the set of labelled arcs (a, b), and when $H(a, b) \neq \emptyset$, the label is $H(a, b)$.

Each feedback is identified by a global identifier to allow ordering. Note that we have not included time associated with each feedback, but our model can be easily expanded to accommodate it. For example, $f(a, b, \tau) \in [0, 1]$, where τ is the time at which the feedback was provided.

Once the feedback history graph is obtained, the next step is to produce a reputation graph.

3.2.2 Stage 2—Obtain Reputation Graph

A Reputation Graph (RG), $RG = (A, E')$, is a directed and weighted graph, where the weight on an arc $r(a, b) \in E' \mapsto \mathbb{R}$ represents the trustworthiness of b from a's perspective. The edges are added via transitive closure of edges in E. That is: if $(a, b) \in E$ and $(b, c) \in E \Rightarrow (a, b)$, (b, c), and $(a, c) \in E'$ (the labelling of the edges, however, depends on the particular trust algorithm).

Note that while a path (a, b) in E' indicates the transitive closure relation, trust algorithms may also exhibit the reflexive property by adding arcs that loop to indicate that the truster trusts itself to a certain degree for a particular task [10, 12, 14].

The existing literature categorizes trust algorithms into two groups: global and local [5, 14] (also see Chapter 2). We can use both groups of algorithms in our model

by assigning appropriate weights of incoming arcs of an agent. Global algorithms assign a single trust score to each agent. Therefore, if a global trust algorithm is used, then the weights of the incoming arcs of an agent should the same, as shown in Figure 3.1(b). There is no such property for local trust algorithms.

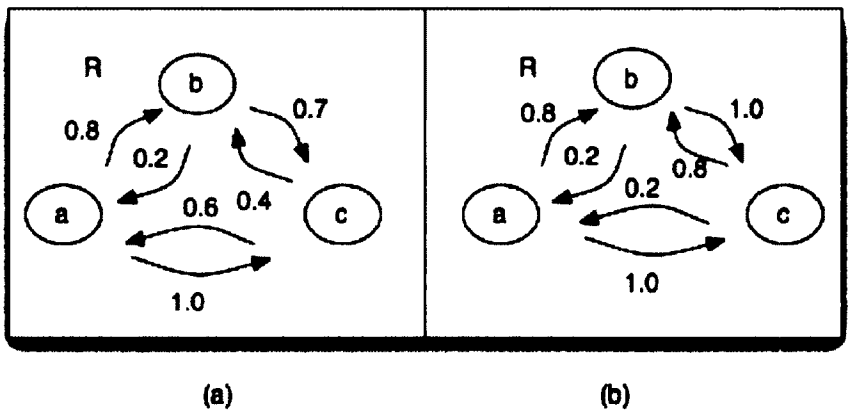


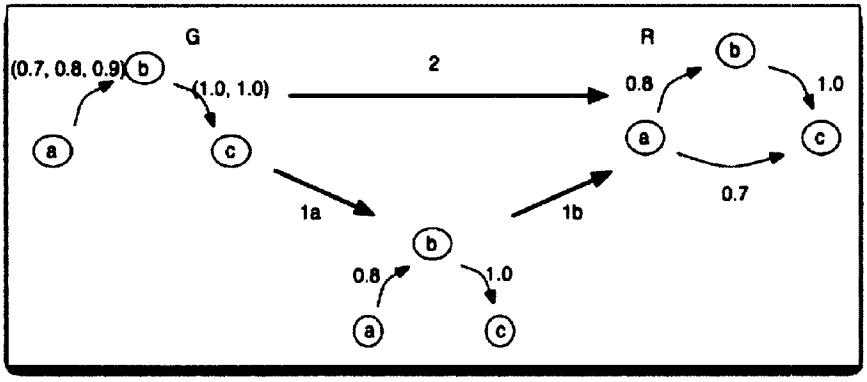
Figure 3.1: Examples of reputation graphs output respectively by a local and global algorithm

We can also classify trust algorithms based on how the reputation graph is produced. One method is to first calculate one-to-one trust scores of agents using direct feedbacks and then use them to calculate the trustworthiness of agents previously unknown to the truster (for e.g. see EigenTrust algorithm in 2.3.2.2). This is shown as 1a and 1b in Figure 3.2. The other method (#2 in Figure 3.2) skips the intermediate graph in the aforementioned method and produces a reputation graph (for e.g. see PeerTrust algorithm in 2.3.2.1).

3.2.3 Stage 3—Obtain Trust Graph

The graph obtained in stage 2 contains information about the trustworthiness of agents. But to use this information to make a decision about a transaction in the future, agents must convert trustworthiness to boolean trust, which can also be expressed as a graph. We refer to this directed graph as the Trust Graph (TG)

Figure 3.2: Two methods to obtain a reputation graph



$TG = (A, F)$, where an edge $f \in F$ represents a trust relationship between the agents (vertices).

3.3 Classifying and Chaining Algorithms

By revisiting the trust models in Chapter 2, and refactoring them according to the stages presented in the above sections, we start to see a new classification scheme. Let us take EigenTrust, PeerTrust, and Appleseed as examples and describe them using our model. EigenTrust takes $FHG : A \times A \mapsto \{0,1\}^*$ as input and outputs $RG : A \times A \mapsto [0,1]$. PeerTrust, on the other hand, takes $FHG : A \times A \mapsto [0,1]^*$ as input and outputs $RG : A \times A \mapsto [0,1]$. Meanwhile, Appleseed requires $RG : A \times A \mapsto [0,1]$ as input and outputs another $RG' : A \times A \mapsto [0,1]$. It is also possible for an algorithm to skip some stages. For example, according to our mode, ManagingTrust [5] skips stage 2 and does not output a reputation graph.

A classification scheme for trust algorithms using our model is provided in Table 3.1[1].

[1]Note that in this table, we have included "0" in $A \times A \mapsto \{0,1\}$ for a trust graph for clarity but in reality, it indicates that there is no edge.

Trust Algorithm		Stage 2	Stage 3
EigenTrust	precondition	FHG: $A \times A \mapsto \{-1, 1\}^*$	-
	postcondition	RG: $A \times A \mapsto [0, 1]$, fully connected, global, relative rep. scores	-
PeerTrust	precondition	FHG: $A \times A \mapsto [0, 1]^*$	-
	postcondition	RG: $A \times A \mapsto [0, 1]$, fully connected, global, absolute rep. scores	-
Appleseed	precondition	RG: $A \times A \mapsto [0, 1]$	-
	postcondition	RG: $A \times A \mapsto \mathbb{R}_0^+$, partially connected, local, relative rep. scores	-
ManagingTrust	precondition	FHG: $A \times A \mapsto \{-1, 1\}^*$	-
	postcondition	-	TG: $A \times A \mapsto \{0, 1\}$, fully connected, global
Advogato	precondition	-	RG: $A \times A \mapsto \{0, 1\}$
	postcondition	-	TG: $A \times A \mapsto \{0, 1\}$, partially connected, local
Ranking	precondition	- -	RG: $A \times A \mapsto [0, 1]$, relative rep. scores
	postcondition	- -	TG: $A \times A \mapsto \{0, 1\}$, partially connected
Thresholding	precondition	-	RG: $A \times A \mapsto [0, 1]$, absolute rep. scores
	postcondition	-	TG: $A \times A \mapsto \{0, 1\}$

Table 3.1: A new classification for trust models.

Note that although these three algorithms output a reputation graph with continuous reputation values between 0 and 1, the semantics of these values are different. EigenTrust outputs relative (among agents) global reputation scores, PeerTrust outputs an absolute global reputation score, and Appleseed produces relative local reputation scores. In other words, EigenTrust and Appleseed are ranking algorithms (global and local, respectively), whereas PeerTrust is not.

3.4 Describing Experiments Using Petri Nets

Since different trust algorithms have been shown to belong to different stages of a workflow, it makes sense to represent an experiment as a workflow. Furthermore, if the trust algorithms are generalized as algorithms that take a graph and outputs another, as shown in Section 3.3, then it is possible to evaluate them by examining the input and the output graphs for certain criteria. This suggests that an algorithm to evaluate a trust model can also be included in the workflow in a similar fashion. Thus, we need to address the following:

1. How to specify the input and output of algorithms.
2. How to state the execution of the algorithms using a simple enough workflow language.

We propose using Petri net as a solution.

3.4.1 Background on Petri Nets

A Petri net is a model for describing processes that may run sequentially or concurrently if their pre-conditions have been met. It is a bipartite graph in which nodes represent places (conditions) and transitions (actions). An arc from a place to a transition indicates the pre-conditions for that transition. An arc from a transition to a

place indicates the post-condition for that transition.

If we denote P as the set of places, and T as the set of transitions, then a Petri net $N = (P \cup T, (P \times T) \cup (T \times P))$ [25]. Given an element $x \in P \cup T$, then ${}^\bullet x := \{y \in P \cup T | (y, x) \in F\}$ denotes the input elements of x, and $x^\bullet := \{y \in P \cup T | (x, y) \in F\}$ denotes the output elements of x.

A token on a place p indicates that the conditions associated with p are satisfied and a transition t can be triggered only if there are sufficient tokens in ${}^\bullet t$. After t occurs, one token are removed from each place in ${}^\bullet t$ and added to $t^\bullet$. This allows for an exact description of how the algorithms described in Table 3.1 can be chained and executed.

3.4.2 Input and Output Parameters of Algorithms

We represent an algorithm as transitions, input places as input parameters, and output places as output parameters of the algorithm. The places can be parameter types such as graphs, boolean values, numerical values, agents, etc. And a token corresponds to a particular instance of that parameter. The signature of the algorithm is defined by the structure of a Petri net. Table 3.2 provides examples of algorithms viewed as functions[1] and their corresponding Petri net structures. Note that each place is marked with an identifier. This allows us to map the input and output places of a transition to input and output parameters of the algorithm.

3.4.3 Tokens as Events

A token in a place indicates that an event associated with that place has occured, i.e., a new instance of the corresponding parameter type has become available and therefore the place is ready to processed. Our testbed allows the experimenter to

[1] In this table, G can be FHG, RG or TG.

specify how a token is associated with a change in a place. For example, a token may be put in an FHG place after n number of feedback is added to the graph. In another example, a token may be placed in an RG place to indicate that the graph has been updated.

3.4.4 Justification for Petri Nets

We note that a simpler linear specification would have been sufficient to describe reputation systems and evaluations that only take the output graphs of the algorithms. For example, a UNIX pipe-based notation such as the following could have been used:

- FHG > EigenTrust | Ranking > TG

However, some of the evaluations may require one or more graphs, and such requirements cannot be met using a linear model. Thus, a directed graph-like approach was needed and Petri nets satisfied these requirements because they allows us to describe various evaluation scenarios (see Table 3.2). Conversely, richer workflow description languages were deemed too complex for our purpose.

Note that although authors in [26] also use Petri Nets to model experiments to evaluate trust algorithms, they denote agents as places whereas we denote input and output of trust algorithms as places. The lack of a good explanation of their model makes it hard to compare our use of Petri Net with theirs. Moreover, it is not clear how existing algorithms can fit in their testbed.

3.4.5 Examples

The Petri net to describe PeerTrust is shown in Figure 3.3. A token is placed in *FHG* after it has been populated with feedbacks. This indicates that the transition, PeerTrust, is ready to be fired. When it is fired, reputation scores between agents

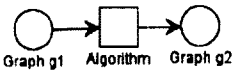
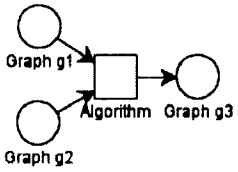
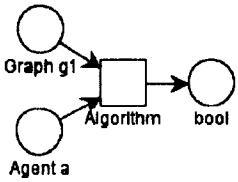
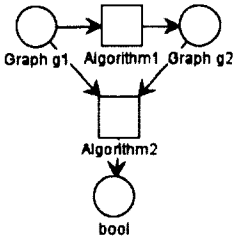
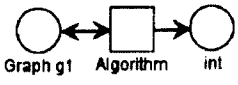
Function	Petri Net
`Algorithm:` $G \rightarrow G$	
`Algorithm:` $G \times G \rightarrow G$	
`Algorithm:` $G \times A \rightarrow \{0,1\}$	
`Algorithm1:` $G \rightarrow G$ `Algorithm2:` $G \times$ `Algorithm1`$(G) \rightarrow \{0,1\}$	
`Algorithm:` $G \rightarrow G \times \mathbb{N}$	

Table 3.2: Algorithms and corresponding Petri nets

are calculated using FHG as input, and RG is updated with reputation edges. The transition then removes a token from FHG, and puts a token to RG indicating that it is finished.

Figure 3.3: Workflow for PeerTrust

Besides using a Petri net for modelling simple trust algorithms, we can also use it for describing evaluations. Suppose an experimenter wants to compare PeerTrust and EigenTrust. The inputs and outputs of these algorithms are semantically different. So to match the input, we could use a Discretization algorithm (DA) that takes $FHG0 : A \times A \mapsto [0,1]^*$ as input and outputs $FHG1 : A \times A \mapsto \{-1,1\}^*$ (e.g., if $f(a,b)$ in $FHG0$ is greater than some threshold, then $f'(a,b)$ in $FHG1$ is 1, otherwise -1). Using a Normalization algorithm (NA), the output of PeerTrust is normalized such that it is semantically equivalent to that of EigenTrust. It takes $RG2 : A \times A \mapsto [0,1]$ as input, and outputs another reputation graph $RG3 : A \times A \mapsto [0,1]$, where the reputation scores are relative to one another. This is achieved using Equation 3.4.1, where C is the set of agents reached by the outgoing edges of a.

$$r'(a,b) = \frac{r(a,b)}{\sum_c^C r(a,c)} \tag{3.4.1}$$

Once the pre-conditions and post-conditions of EigenTrust and PeerTrust are matched, Spearman's rank correlation coefficient [27] is computed. This metric $\in [-1,1]$ specifies the degree to which the ranking order of the outputs match, where 1 means a perfect match and -1 means no match. Thus, it takes two reputation graphs and outputs a value in $[-1, 1]$.

The choice of discretization and normalization methods are important and they may introduce a bias when comparing two algorithms. However for the purpose of

demonstrating our model, we will assume that our choices are good enough.

The Petri Net and its order of execution are shown from Figures 3.4 to 3.9.

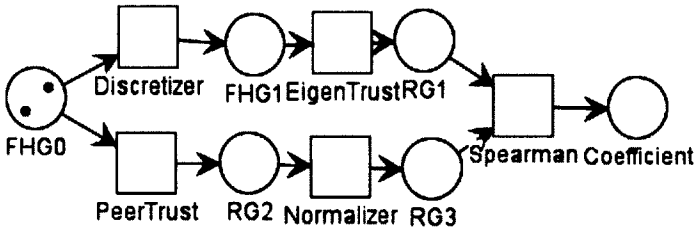


Figure 3.4: EigenTrust and PeerTrust comparison—Initial Petri net

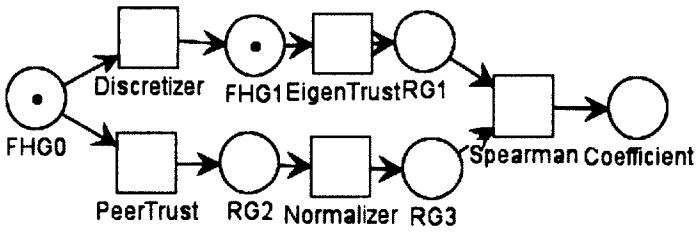


Figure 3.5: EigenTrust and PeerTrust comparison—After running Discretizer

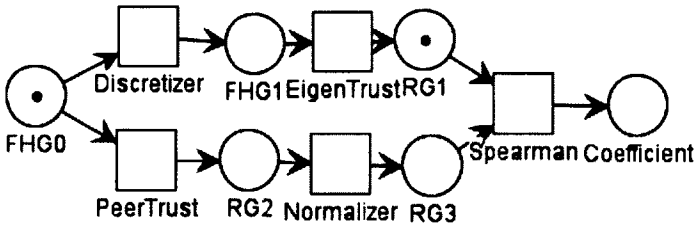


Figure 3.6: EigenTrust and PeerTrust comparison—After running EigenTrust

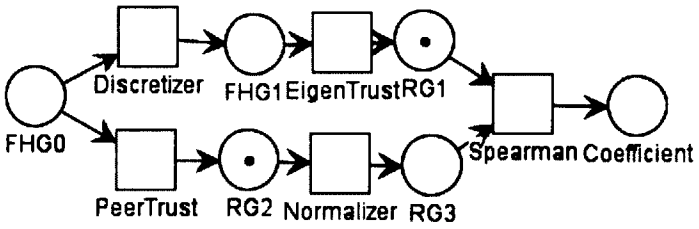


Figure 3.7: EigenTrust and PeerTrust comparison—After running PeerTrust

In the current implementation of our testbed, we assume that the pre-condition checks as per Table 3.1 are enforced by the algorithms rather than the testbed. However, this can be modified easily in the future.

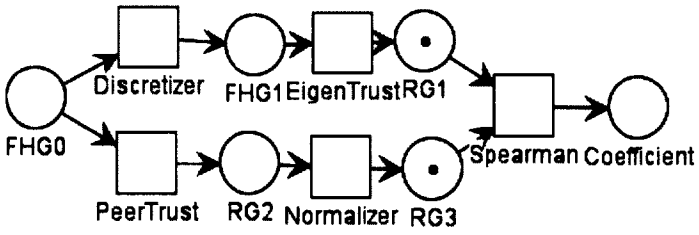


Figure 3.8: EigenTrust and PeerTrust comparison—After running Normalizer

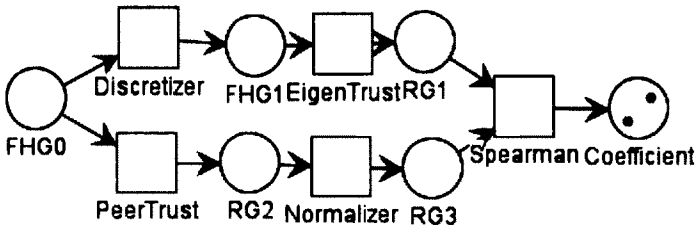


Figure 3.9: EigenTrust and PeerTrust comparison—After running Spearman coefficient calculation

3.5 Summary

In this chapter, we showed that trust can be examined as a process that occurs in stages and that many existing trust algorithms can be categorized according to the stages they fit to. We then presented a Petri net-based solution to describe both the trust algorithms and their evaluations.

3.5.1 Comparison

Comparing our model with Guha's work [23] (see Section 2.5.1), we note that the trust graph W in [23] corresponds to the trust graph in our model, but the rating of objects does not fit in our model. However, the fundamental difference between our model and Guha's is that Guha's work tries to solve a related but different problem. It tries to complete graph R using W (i.e., given W, find $R(A_i, O_k)$ if it is unknown) and rank N items in O according to A_i. This model can be used for recommendation systems such as Credence [15]. On the other hand, our model compares trust algorithms that complete graph W, given $(A \times A \rightarrow \mathbb{R}^*)$ or $(A \times A \rightarrow \mathbb{R})$.

In the next chapter, we experiment and analyze trust algorithms using the model described in this chapter.

Chapter 4

Implementation and Evaluation

We built and designed a prototype based on the model proposed in Chapter 3. Section 4.1 provides implementation details on this testbed. We conducted several experiments and analyzed their results on three trust algorithms, namely EigenTrust, PeerTrust, and Appleseed [1]. The experiments are grouped into two categories: vulnerability assessments (Section 4.2) and trust properties assessments (Section 4.3). Vulnerability assessments aim at testing trust algorithms and their behaviour under attack scenarios. In Section 2.2, we stated that trust models must adhere to certain properties of trust. We cover these types of tests in the trust properties assessments section.

Table 4.1 summarizes the various tests we carried out.

4.1 Implementation

A prototype was designed and built in Java to test the model described in Chapter 3.

The two main components of the testbed are graphs and algorithms (see Figure 4.1). A *Graph* can be a feedback history graph, reputation graph, trust graph, or a

[1] Although we implemented ManagingTrust, we did not run any experiments against it due to time constraints.

Table 4.1: Overview of the experiments

Category	Test	Algorithms tested		
		EigenTrust	PeerTrust	Appleseed
Vulnerability assessments	A priori trust	✓	✓	
	Bootstraping	✓	✓	
	Slandering	✓	✓	✓
	Sybil			✓
Trust properties assessments	Dynamic property		✓	
	Trust propagation	✓		✓

Petri net. These graphs follow our model as described in Section 3.2. The input and output of the algorithms are defined by the structure of the Petri Net (see Section 3.4.2).

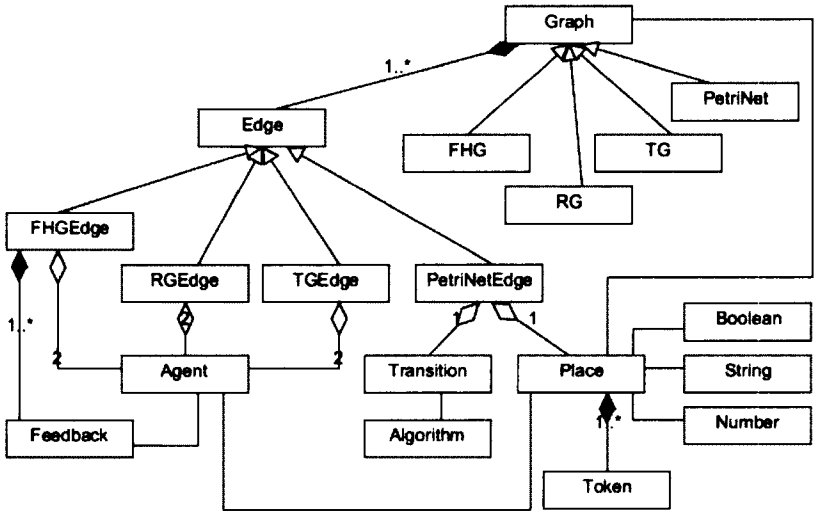


Figure 4.1: Class diagram for the testbed

The traversal of the Petri net describes the execution of the test. It begins by putting tokens on places, which are seen as events indicating that the corresponding places are ready to be processed. The transitions that are reachable from the places are then requested to fire. If there is a token on every input place, then the transitions may be fired. They in turn invoke the wrapped algorithms, passing on both input places as inputs to the algorithms. The algorithms execute, update the output places and exit. The transitions then put a token on every output places, indicating that they are ready to be processed.

The graphs were built using JGraphT library [28], which is a popular graph library for Java. The graphs can be populated either programatically or by providing a file as per the Attribute-Relation File Format (ARFF) [29]. For instance, a feedback history graph is populated with a file containing a list of relations, each containing 3 attributes: source agent identifier, sink agent identifier and the feedback value. An example of a ARFF file for populating a feedback history graph is given in Listing 4.1:

Listing 4.1: An example ARFF file for populating a feedback history graph

```
@relation feedback
@attribute assessorID string
@attribute assesseeID string
@attribute feedbackValue numeric
@data
0,1,0.6
0,1,0.7
0,1,0.2
```

A reputation graph is populated using a file containing a list of relations each containing 3 attributes; source agent identifier, sink agent identifier, and the reputation

(for e.g., see Listing 4.2).

Listing 4.2: An example ARFF file for populating a reputation graph

```
@attribute srcAgent  string
@attribute sinkAgent string
@attribute reputation numeric
@data
0,1,1
1,2,1
1,3,1
```

Finally, the input for a trust graph contains a list of trust relations, each containing 2 attributes; source and sink agent identifiers.

In this prototype, the Petri net is created and executed programmatically using our API. However, this can be easily enhanced so that a user interface or a file-based input is used instead. Suppose we have a simple Petri Net as shown in Figure 3.3 to evaluate PeerTrust. The code given in Listing 4.3 is used to construct this Petri Net in our testbed. When $t1$ is fired, it invokes the underlying algorithm and if the input and output are reputation graphs (this is the case for PeerTrust), it invokes `algorithm.calculateScore(Agent source, Agent sink)`. The output reputation graph is then populated with the values from this method.

Listing 4.3: How to create a simple petri net in our testbed

```
FHG fhg = new FHG();
Place p1 = new Place(fhg);
RG rg = new RG();
Place p2 = new Place(rg);
PeerTrust pt = new PeerTrust();
```

```
Transition t = new Transition(pt);
Workflow wf = new Workflow();
wf.addEdge(p1, t);
wf.addEdge(t, p2);
fhg.populate(arffFile);
p1.placeToken();
t1.fire();
```

The testbed's source code is open and is available on Google Code (`http://code.google.com/p/repsystestbed/`).

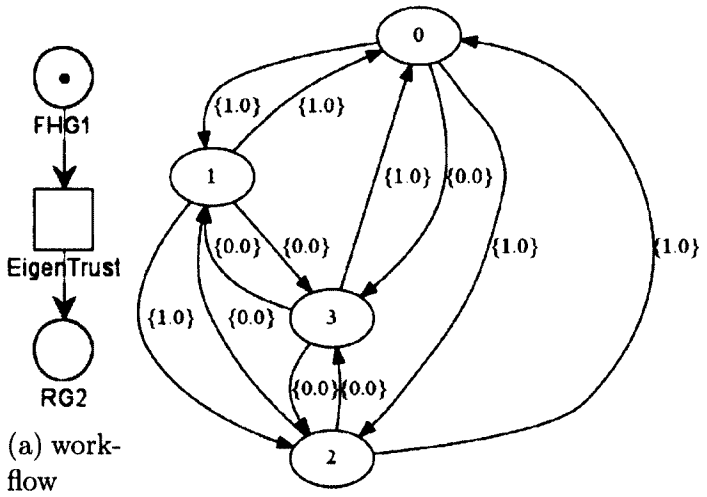
4.2 Vulnerability Assessments

The security of trust algorithms is measured by their resistance to attacks. In this section, we subject them to attacks and assess their vulnerabilities.

4.2.1 A Priori Trust

A priori trust is required in some trust algorithms such as PeerTrust and EigenTrust (see sections 2.3.2.1 and 2.3.2.2) but this exposes these algorithms to attacks targeting the pre-trusted agents. In this section, we analyze how PeerTrust and EigenTrust are affected by the choice of the pre-trusted agents.

Setup for EigenTrust One of the requirements for EigenTrust is that the initial normalized matrix containing the relative reputation scores, c_{ij}, must be irreducible and regular [10]. To explain this requirement, suppose we were given FHG1 in Figure 4.2(a). We can use a simple Petri net as shown in Figure 4.2(b), to model and run the experiment.



(a) work-flow

(b) FHG1

Figure 4.2: Setup for EigenTrust

EigenTrust Results If there are no pre-trusted agents, EigenTrust outputs RG2, as shown in Figure 4.3. Note that the sum of the weights along the outgoing edges of an agent is not equal to 1, and therefore the results are invalid. The rest of this section explains how we reached these invalid results.

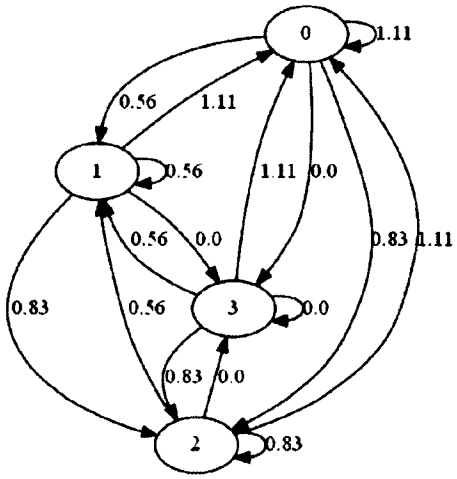


Figure 4.3: EigenTrust with no pre-trusted agents

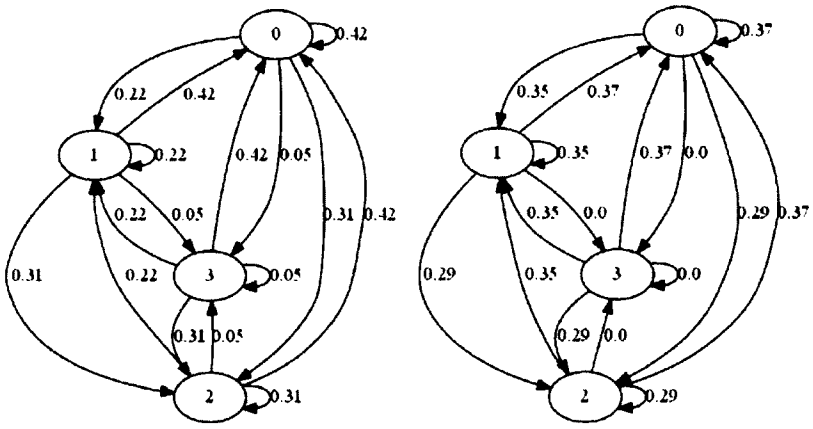
According to [10], the feedback history can be summarized by s_{ij}

as $\begin{bmatrix} 0.0 & 1.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$. After normalizing this matrix, we obtain c_{ij}= $\begin{bmatrix} 0.0 & 0.5 & 0.5 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$.

As noted earlier (see Section 2.3.2.2), the above matrix is not a regular matrix (i.e some power of the matrix must contain only positive elements) and thus it cannot converge.

For a matrix to be irreducible, there must be a path from an agent to every agent in the graph. The paths from agent 0 are 0→1, 0→2, 0→1→2, but there is no path from agent 0 to agent 3. In a way, agent 0 is stuck in a group with agent 1 and 2. To mitigate this problem, the authors in [10] propose a solution: placing a certain amount of trust on at least one agent that is outside the collective group, such that there is a chance for agent 0 to get out of the loop and reach agent 3. In short, pre-trusted agents are often necessary for EigenTrust to produce meaningful and correct results.

But how do we select the pre-trusted agents? The authors of EigenTrust assume that the initial peers in the system are typically trustworthy and thus can be considered pre-trusted. What if we consider all agents to be pre-trusted? We obtain a valid reputation graph (RG2 in Figure 4.4(a)), which agrees with our intuition because we expect the best rated agents to be the most trustworthy. If we randomly select agent



(a) RG2—all agents pretrusted (b) RG2—agent 1 pretrusted

Figure 4.4: (a) EigenTrust with all agents pre-trusted, (b) EigenTrust with agent 1 as the only pre-trusted agent

1 to be the pre-trusted agent, EigenTrust outputs RG2, given in Figure 4.4(b), and as expected, agent 0 is the most trustworthy. Thus, the choice of pre-trusted agent is critical to EigenTrust's performance and this exposes EigenTrust to targeted attacks on pre-trusted agents.

This aspect of EigenTrust has been researched and alternatives have been proposed in [30] and [31]. [30] proposes an extension of EigenTrust where globally pre-trusted agents are not required. Instead, each agent randomly chooses a trusted agent and assigns it as the "representative node" whose function is to break away from malicious collectives. [31] shows that by attacking the pre-trusted agents in EigenTrust, attackers can easily achieve a high rank in the network. To make EigenTrust resilient to this attack, they propose a version of EigenTrust where, instead of producing global reputation scores and global pre-trusted agents, the reputation scores are local and the pre-trusted agents are personalized. However, this makes their algorithm a local trust algorithm.

Setup and Results for PeerTrust PeerTrust [12] also requires a priori trust in agents for it to function. However, unlike EigenTrust, all agents must be pre-trusted.

For example, consider a simple feedback history graph where agent 0 has rated 1 with a value 1.0 (positive feedback). In order to perform Equation 2.3.1, we need to know agent 0's trust score, and because it has not received any feedback, its trust score must be initialized to some value. The authors of PeerTrust suggest setting it to $\frac{1}{|A|}$ where $|A|$ is the total number of agents. Thus, PeerTrust is also vulnerable to targeted attacks on pre-trusted agents.

4.2.2 Bootstrapping-based Attacks

This type of attacks is aimed at exploiting algorithm-specific bootstrapping techniques.

4.2.2.1 Negative feedback versus no feedback

Setup In this experiment, we wanted to investigate whether the reputation score of an agent with negative feedbacks differs from one with no feedbacks. For example, consider FHG0, given in Figure 4.5(b), where agent 3 has received only negative feedbacks (because feedback value < 0.7), whereas agent 0 has received no feedback. Ideally, a trust algorithm should be able to differentiate between an agent with no feedbacks and an agent with negative feedbacks. For example, one may expect that the reputation of an agent that received negative feedback be less than that of one with no feedback, so that newcomers are not discouraged to join the system.

Results Figures 4.5(e) and 4.5(f) show the input and output of EigenTrust [10] and PeerTrust [12]. The Spearman coefficient of RG1 and RG3 is 0.83, which suggests that EigenTrust's ranking of agents is not the same as PeerTrust's. If we order the agents according to their relative global reputation scores in RG1 and RG3, we obtain Table 4.2. We observe the following:

1. As expected, Agent 3 is ranked lowest because it received only negative feedbacks. However, EigenTrust also ranks agent 0 lowest, while PeerTrust ranks it at second lowest.

2. According to PeerTrust, agents 1 and 2 are of the same rank because they both received equal number of positive feedbacks. However, according to EigenTrust agent 2 is the most trustworthy. We explain the observations below.

	Agent 0	Agent 1	Agent 2	Agent 3
EigenTrust	3	2	1	3
PeerTrust	2	1	1	3

Table 4.2: Negative feedback versus no feedback - ranks

As noted in Section 2.3.2.2, EigenTrust has a requirement for matrix c_{ij} to converge, and in order for that to take place, the elements in the matrix can only contain positive elements. If the sum of the feedbacks is less than 0, EigenTrust assigns a initial reputation score of 0 for the assessee, as per Equation 2.3.3. Thus, in a system implementing EigenTrust as the trust algorithm, an agent cannot differentiate between a newcomer and a malicious agent that received all negative feedbacks. PeerTrust, on the other hand, does not have this disadvantage.

According to EigenTrust, agent 2's reputation score is higher than agent 1's even though both agent 1 and 2 received the same amount of positive feedbacks. This is because if an agent a has direct interaction with agent b and b has direct interaction with agent c, then $r(a, c)$ is dependent on both $r(a, b)$ and $r(b, c)$. In our case, agent 2's recommender is agent 1 and agent 1's recommender is agent 0. Because $r(*, 1)$ is greater than $r(*, 0)$ where * represents any agent, EigenTrust accurately outputs that agent 2's rank is higher than that of agent 1's.

When calculating the reputation score of agent 1 using PeerTrust (see Equation 2.3.1), agent 0's feedbacks on agent 1 are weighted by 0.25. This is because the

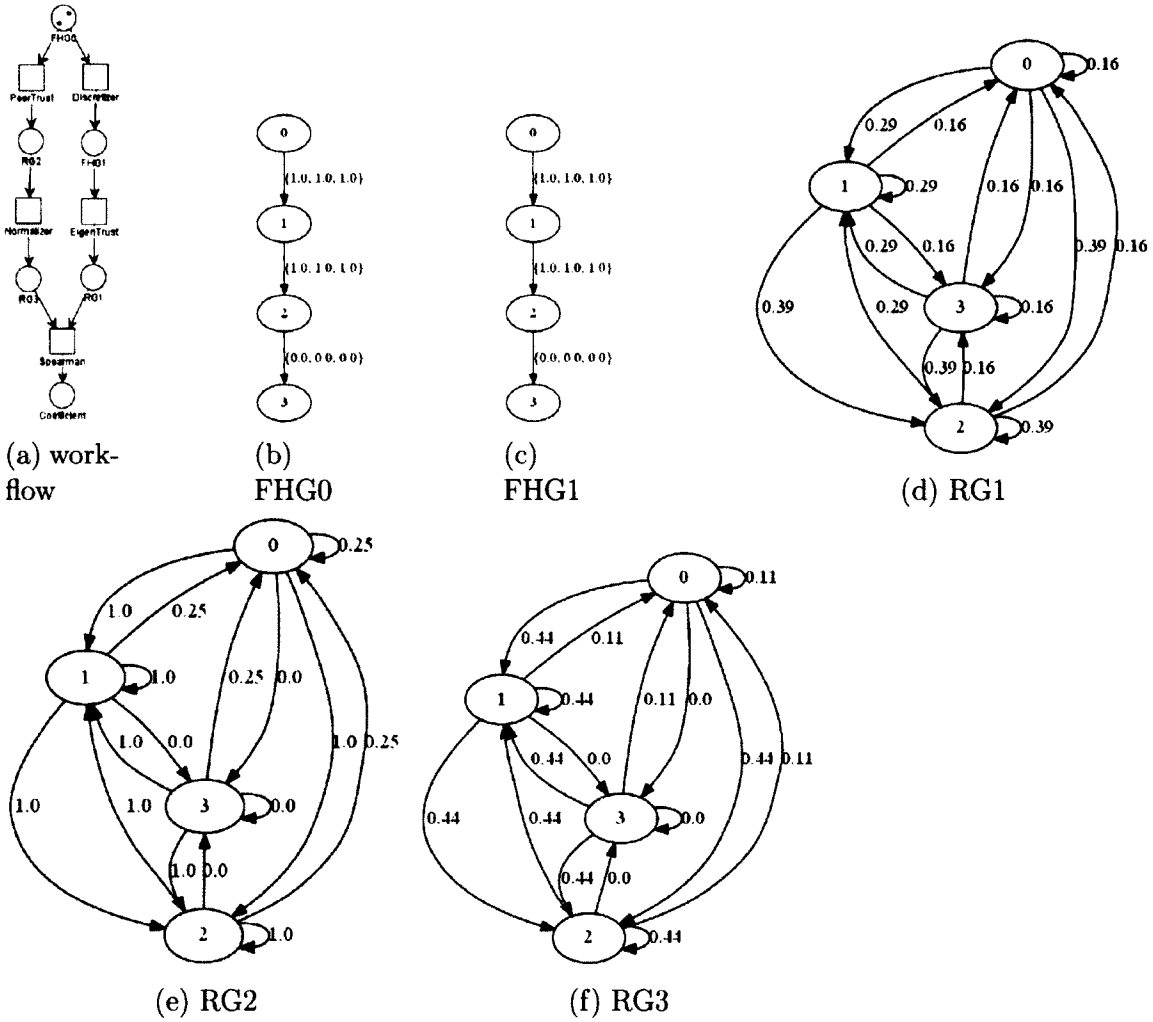
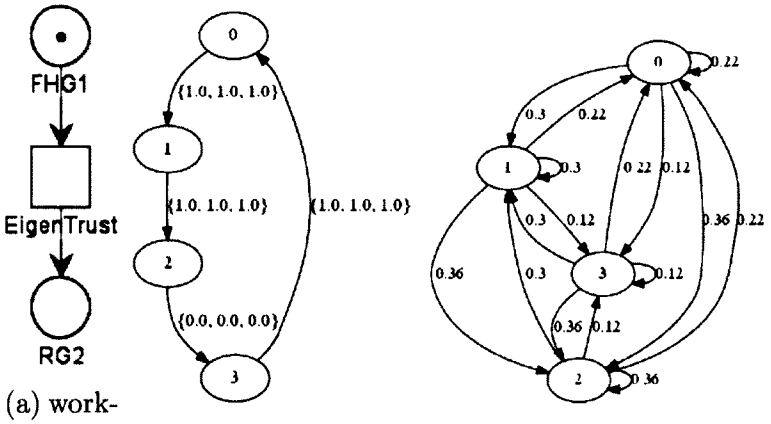


Figure 4.5: Negative feedback versus no feedback—Spearman Coefficient

reputation scores of all agents are initialized to $\frac{1}{4}$. Thus, we obtain an absolute reputation score $(*, 1) = 1.0$. Likewise, $r(*, 2) = 1.0$. However $r(*, 3) = 0$ due to the fact that agent 3 received all 0-valued feedback.

Thus, we conclude that EigenTrust is vulnerable to attacks where a malicious agent that received only negative feedbacks may be confused for a newcomer, but PeerTrust is not sensitive to those same attacks.



(a) work-flow (b) FHG1 (c) RG2

Figure 4.6: Self-promoting attack—EigenTrust

4.2.2.2 Self-promoting Attack

Setup Suppose the least trustworthy agent rates a newcomer (i.e., one that received no feedback) highly. If the newcomer becomes more trustworthy than before the rating, then this can be exploited maliciously, paving way to collusive self-promoting attacks. If the newcomer becomes less trustworthy than before, then the newcomer is vulnerable to a slandering attack. In this case, we may prefer the newcomer's trustworthiness to remain unchanged. Consider FHG1 in Figure 4.6 which is based on FHG0 in Figure 4.5, where agent 3 rates agent 0 positively.

Results EigenTrust's ranking of agents is $\{2, 1, 0, 3\}$, and comparing this with the agent ranks in Table 4.2, we see that agent 0's reputation relative to agent 3 increased. Note that it is possible that agent 3 genuinely rated agent 0 positively in which case, EigenTrust output is accurate but in case this is a collusive self-promoting attack, then it can fool EigenTrust. On the other hand, this experiment breaks PeerTrust, due to a division-by-0 problem in Equation 2.1. Indeed, when calculating agent 0's reputation score, the feedbacks provided by agent 3 and its reputation score must be taken into account. In this case, agent 3's reputation score is 0, due to the fact that it received only 0-valued feedbacks. Thus this breaks Equation 2.1.

4.2.3 Slandering Attack

4.2.3.1 Scenario 1

Setup When the least trustworthy agent (agent 3) slanders a newcomer (agent 0) by providing negative feedbacks, the reputation algorithm may resist by not decreasing agent 0's reputation. The input for this experiment are FHG0 and FHG1 as shown in Figure 4.7.

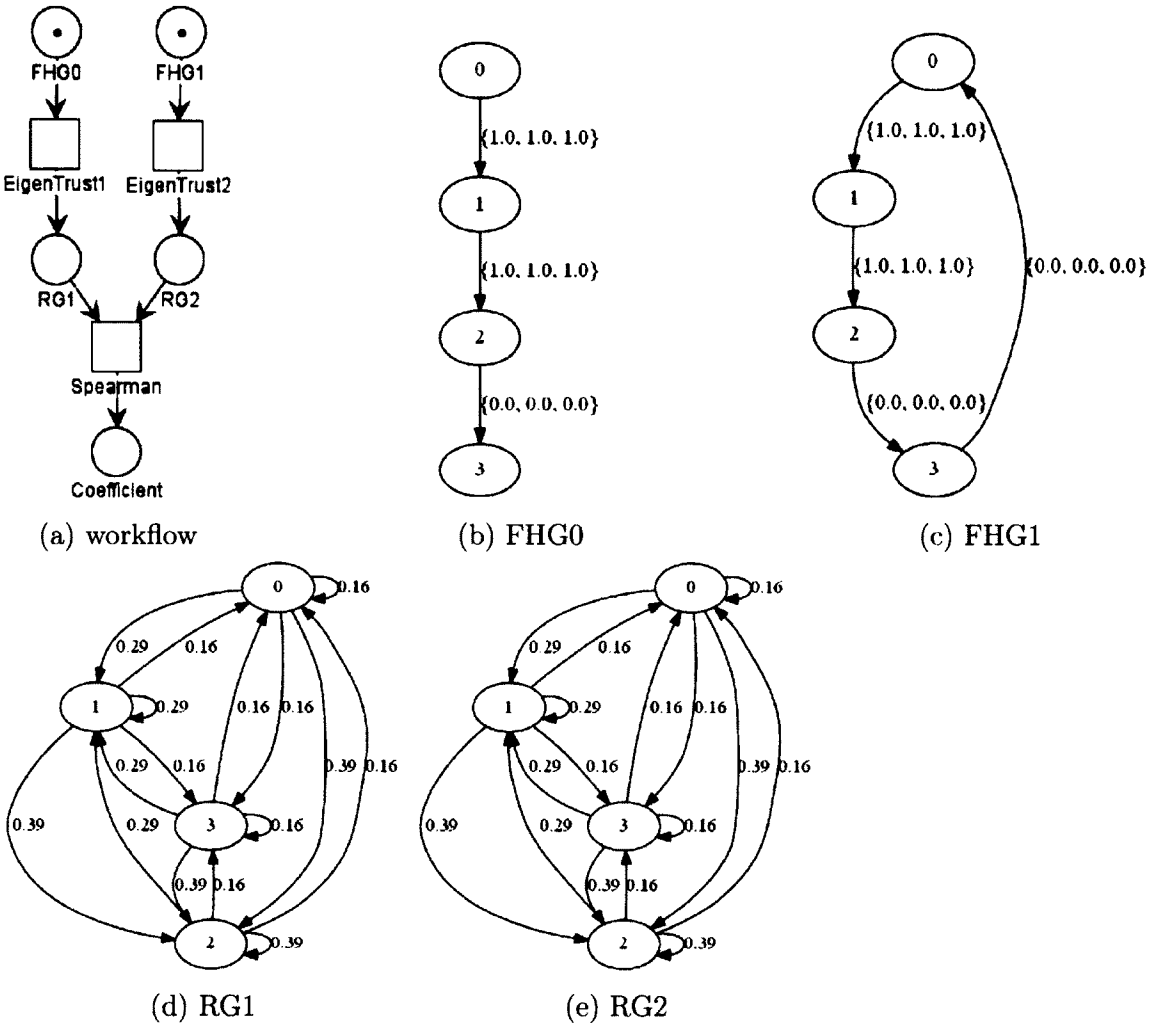
Results Comparing RG1 in Figure 4.7 and RG2 in Figure 4.7, we obtain a Spearman coefficient of 1 (no change) and observe that agent 0's global rank did not change. If agent 3 had indeed provided a dishonest negative feedback, then we can say that EigenTrust output was accurate. However, it is possible that agent 3 had provided a genuine negative feedback, in which case, we would have expected agent 0's rank to decrease. Thus, we can only say that EigenTrust is insensitive to negative feedbacks and therefore resists such slandering of a newcomer. On the other hand, PeerTrust produced invalid results (same reason as in Section 4.2.2.2).

4.2.3.2 Scenario 2

Setup What happens if agent 3 rates agent 2 negatively in spite, as shown in FHG0 in Figure 4.8 and FHG0 in Figure 4.9? One can make the following observations:

- Agent 3 rated agent 2 negatively to cover its malicious acts (a slandering attack).
- Agent 2 cheated agent 3 and thus obtained negative feedbacks but acted honestly with agent 1 in order to keep its reputation high (a white-washing attack).

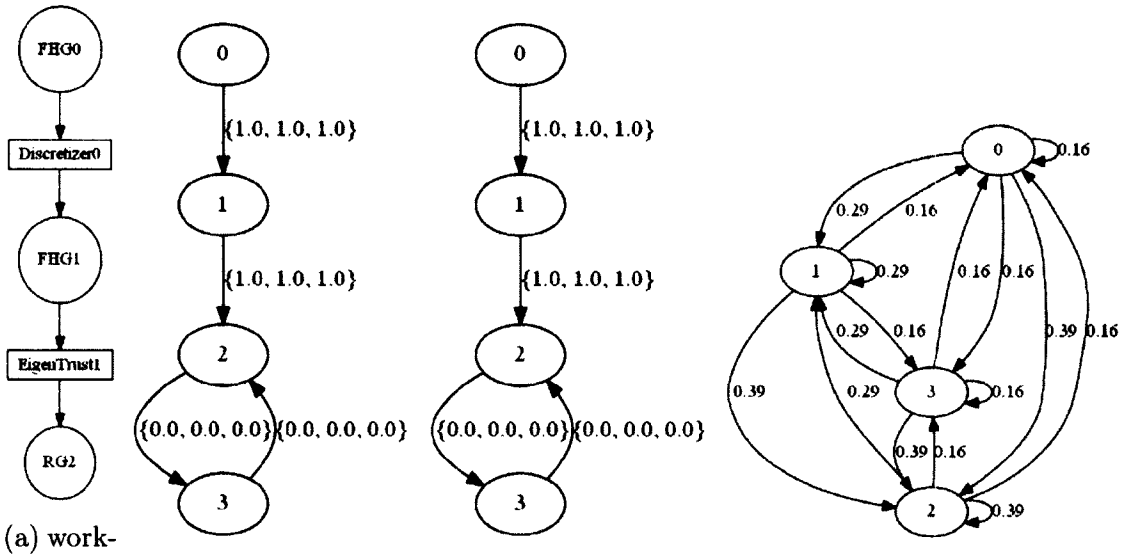
Having received more positive feedbacks than agent 3, if agent 2's reputation is not affected, then the algorithm is said to be resistant to a slandering attack but it is vulnerable to white-washing attack.



(a) workflow (b) FHG0 (c) FHG1

(d) RG1 (e) RG2

Figure 4.7: Slandering attack scenario 1—EigenTrust



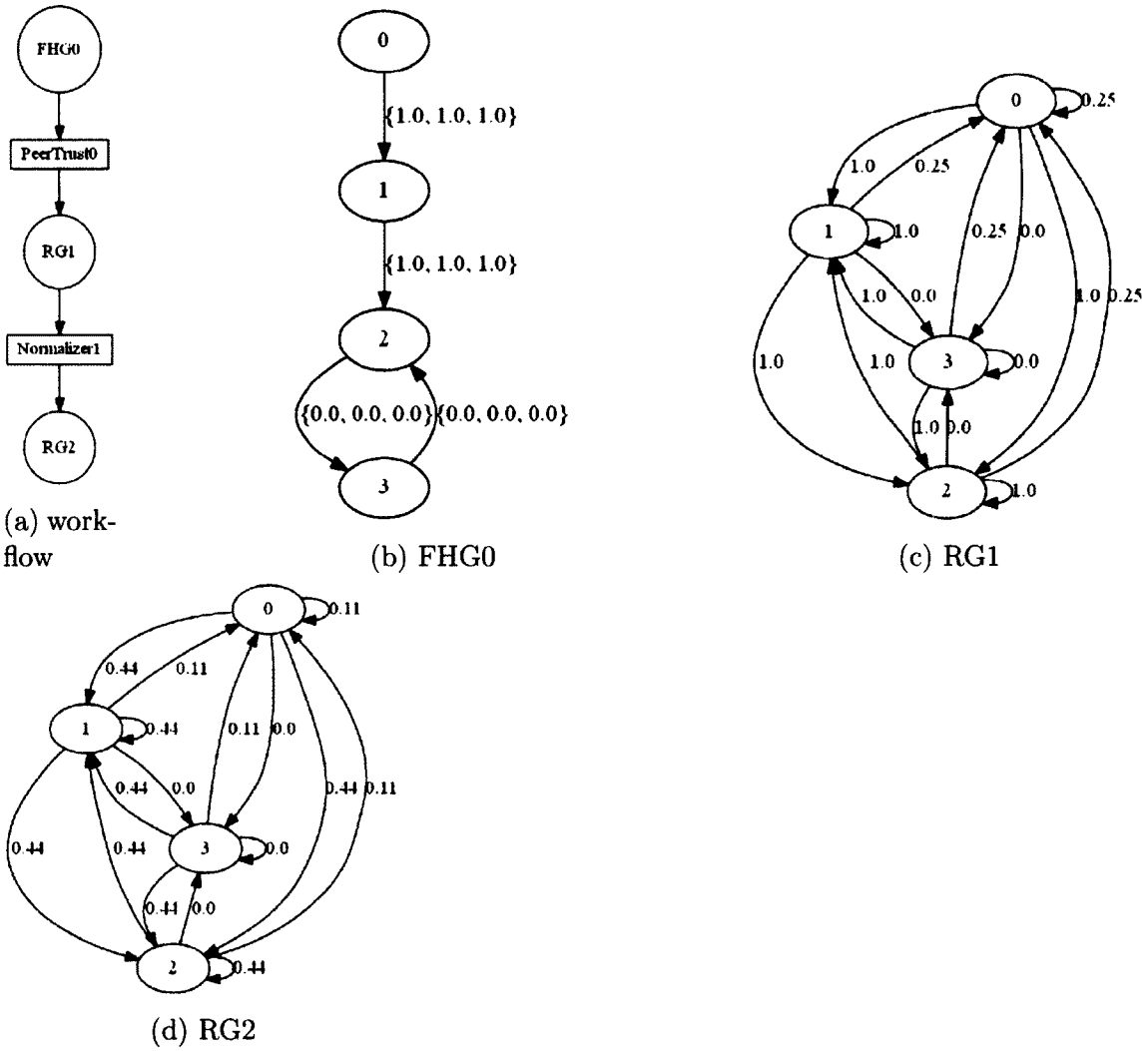
(a) work-flow (b) FHG0 (c) FHG1 (d) RG2

Figure 4.8: Slandering attack scenario 2 - EigenTrust

Results RG2 in Figure 4.8 and RG2 in Figure 4.9 show that both EigenTrust and PeerTrust are resistant to this kind of slandering attack but as mentioned earlier they are vulnerable to white-washing attacks (because the negative feedbacks by agent 3 did not reduce agent 2's reputation).

4.2.4 Slandering + Sybil Attack for Local Trust Algorithms

Setup In this type of attack, a malicious agent introduces a number of Sybil whose purpose is to slander a victim in the system. Suppose we are given the reputation graph shown in Figure 4.10. Let agent 0 be a malicious agent that slanders agent 1. Assuming this agent has unlimited resources, it may introduce an unlimited number of Sybils to collectively slander agent 1. In such a scenario, it is useful to study how $r(0,1)$ changes with respect to other agents. If $r(0,1)$ changes such that it is less than $r(0,2)$, then we can conclude that the slandering attack was succesful and that the greater of Sybils required, the more the algorithm is resistant to a Sybil attack.



(a) work-flow

(b) FHG0

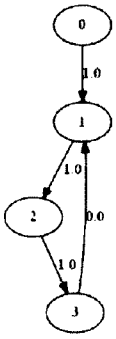
(c) RG1

(d) RG2

Figure 4.9: Slandering attack scenario 2—PeerTrust

However, as explained in Section 4.2.3.2, one can interpret this scenario as a white-washing attack. Thus, if an algorithm is resistant to a slandering attack, then it may be vulnerable to white-washing attacks.

Figure 4.10: Slandering attack using a reputation graph



To measure Appleseed's attack resistance, we used the Petri net shown in Figure 4.11. After populating RG1 with the graph in Figure 4.10, a token is placed in RG1 and Appleseed is executed. After its execution, a token is placed in RG2 (see Figure 4.12), which indicates that we are ready to run the evaluation algorithm (evalss) that verifies whether the victim's reputation is less than that of an agent in attacker's possession (in this case, we wish to check if $r(0,1) < r(0,2)$). If it is false, the result is updated and a Sybil agent x is created. Edges $(2, x, 1.0)$ and $(x, 1, 0)$ are also added to RG1 and a token is placed in RG1, indicating that Appleseed is ready to run again. If it is true, the result is updated and a token is added to RG1 every time RG1 is updated with a slander edge. The number of slander edges required for the attack to be successful is equal to the number of tokens in RG1 minus 1. Figure 4.13 shows the state of the Petri net after adding 10 slander edges to RG (see Figure 4.14). Note that at any given point in time, the evaluation result (attack result) can be known by checking the value in place "result".

Results Appleseed resists this type of attack well. Table 4.3 summarizes the values of $r(0,1)$ and $r(0,2)$ in RG2 after adding 1, 10, 50 and 100 Sybils and slander edges to

Figure 4.11: Sybil attack—Initial Petri net

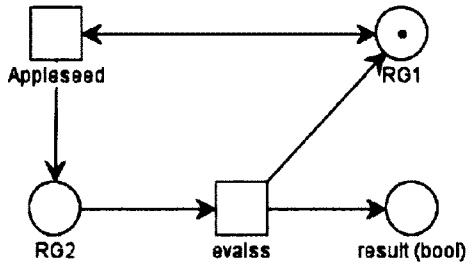


Figure 4.12: Sybil attack—After executing Appleseed

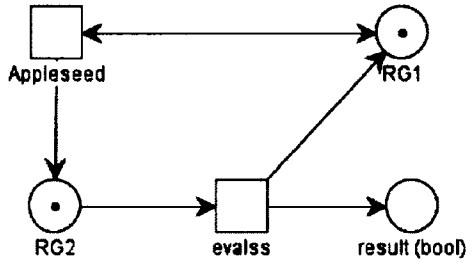
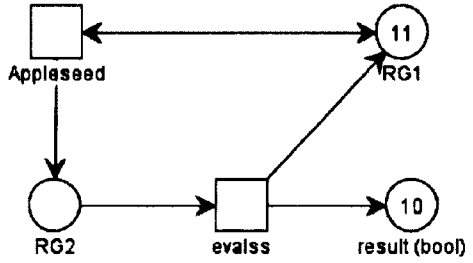


Figure 4.13: Sybil attack—After executing evalss 10 times



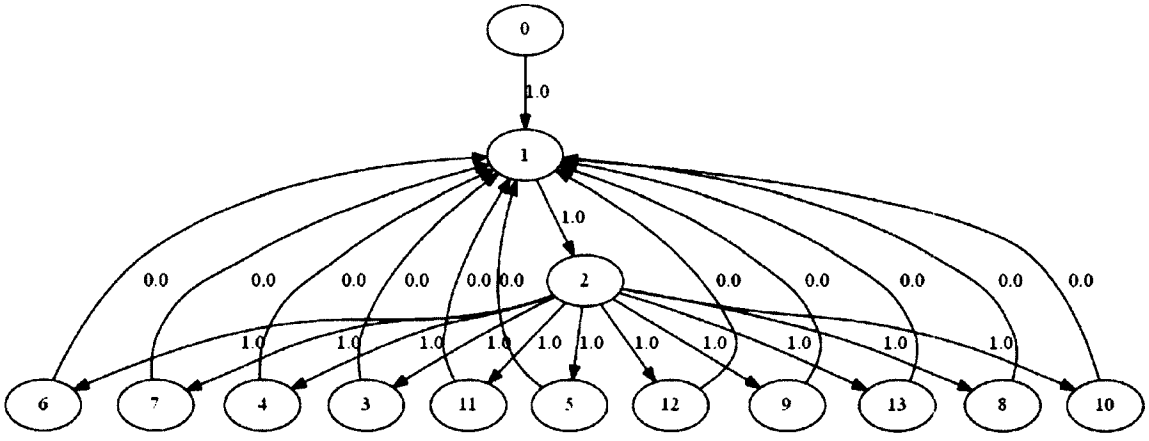


Figure 4.14: RG1 after adding 10 sybils and slander edges

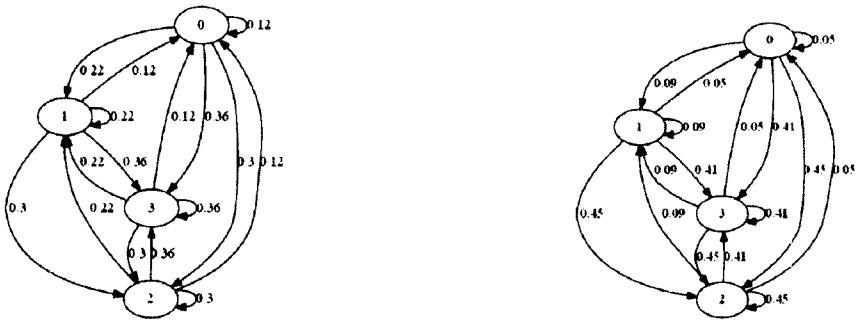
No. of slandering edges added	$r(0,1)$	$r(0,2)$
1	0.36	0.15
10	0.34	0.15
50	0.34	0.14
100	0.34	0.14

Table 4.3: Reputation scores by Appleseed

RG1. We observe that $r(0,1) > r(0,2)$. This confirms the attack resistance property of Appleseed described in [14].

Limitations We modified EigenTrust such that it takes a reputation graph and outputs another reputation graph using Equation 2.3.7. When we ran the same experiment with a single slander edge, our modified version of EigenTrust output the reputation graph as shown in Figure 4.10(a). Since $r(0,1) < r(0,2)$, one can wrongly come to a conclusion that the slandering attack was successful. We explain below why that is a wrong conclusion.

We know that EigenTrust does not differentiate between an agent that received negative feedbacks and no feedback (see Section 4.2.2.1). To ensure that we didn't observe $r(0,1) < r(0,2)$ due to this problem, we added a reputation edge $(3,2,1.0)$ to RG1. In this case, EigenTrust output a reputation graph as shown in Figure 4.15(b). As expected, globally, agent 1 is indeed less trustworthy than agent 2 and 3. This problem manifests itself in all global reputation algorithms because of their "global" view of the system. Therefore, sybil attack resistance evaluation as described above cannot be applied for global reputation algorithms.



(a) Output when RG1 is the input (b) Output after adding edge (3,2,1.0) to RG1

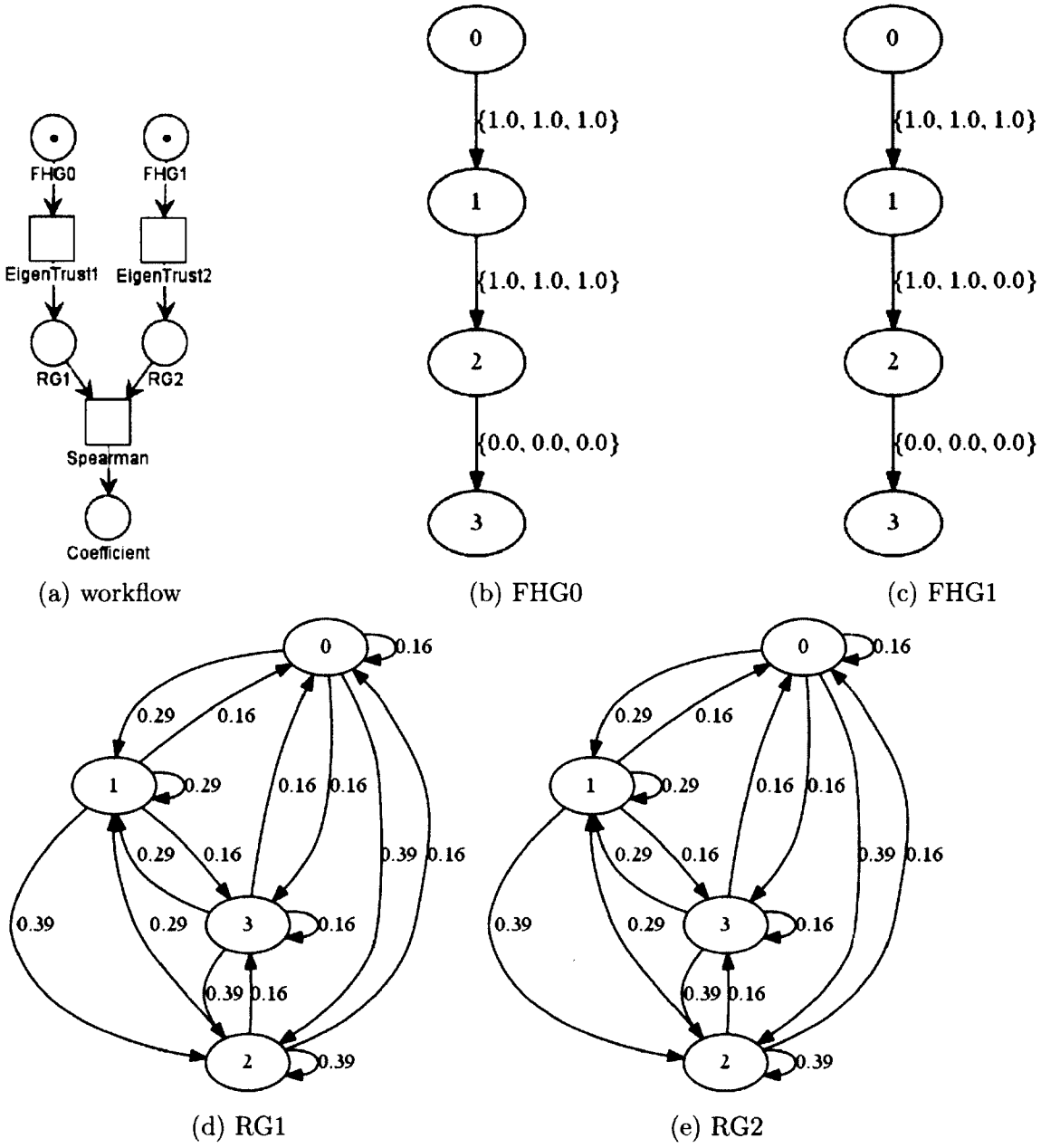
Figure 4.15: Modified version of EigenTrust's output—Sybil attack

4.2.5 Normalization-based Attack

Setup In this experiment, we investigate how reputation is affected by the number of positive feedbacks versus the negative feedbacks. Suppose we were given FHG0 in Figure 4.16(b) and FHG1 in Figure 4.16(c). Since there are positive feedbacks on agent 2 in FHG0 than FHG1, it is reasonable to expect agent 2's reputation in RG1 to be lower than in RG2.

Results Comparing RG1 and RG2 in Figure 4.16, we note that they are identical (Spearman's coefficient = 1.0), which implies that EigenTrust reports no change in agent 2's rank. On the other hand, we noted that agent 2's rank has changed from 1 to 2 according to PeerTrust (Spearman's coefficient = 0.83).

Let us explain why agent 2's rank did not change as per EigenTrust. Using Equation 2.3.3, we obtain $s_{ij} = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ and after normalization, $c_{ij} =$



(a) workflow (b) FHG0 (c) FHG1

(d) RG1 (e) RG2

Figure 4.16: Normalization-based attack—EigenTrust

$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

That is, agent 1 trusts agent 2 fully. Due to this normalization, we have lost a critical piece of information: number of positive feedbacks versus number of negative feedbacks. Thus, if an agent has interacted with a malicious agent only, then the malicious agent can get the victim to trust him fully, as long as the number of positive feedbacks that the malicious agent received is greater than the number of negative feedbacks. This problem is acknowledged by EigenTrust's authors [10]. On the other hand, PeerTrust does not suffer from this problem, because it does not attempt to perform a sum of postive and negative feedbacks in this fashion.

4.3 Trust Properties Assessments

In this section, trust algorithms are evaluted for their adherence to trust properties, namely transitivity and the dynamic evolution of trust, as noted in Section 2.2.

4.3.1 Transitivity

Setup As explained in Section 2.2.2, trust is not transitive in the strict sense. That is, if a trusts b and b trusts c, it is necessarily true that a trusts c. Moreover, trustworthiness decreases as the length of the trust path increases.

We subjected Appleseed and the modifed version of EigenTrust mentioned in Section 4.2.4 to simple tests to ensure the basic transitivity rules of trust are upheld. According to our model, Appleseed expects a reputation graph as input and outputs

a local reputation graph. On the other hand, the modified version of EigenTrust requires a local reputation graph as input and outputs a global reputation graph. Even though the output reputation graphs are of different type, the same assertions can be applied to test the transitive rules, as they do not depend on whether the reputation scores are local or global.

Results Table 4.4 presents the test cases, expected results and actual results. Both Appleseed and EigenTrust passed our transitive tests.

Limitations The authors of [16] suggest that when inferring trust between agent a and b using indirect trust relationships, trust algorithms must ensure only valid paths are used. A valid path is one where the last edge in the path from a to b is a direct trust relationship. Suppose we are given $r(a,b) = 1$, $r(b,c) = 1$, $r(c,d) = 1$ and we wish to infer r(a,d). The paths from a to d are $\{(a,b),(b,c),(c,d)\}$, $\{(a,b),(b,d)\}$ and $\{(a,c),(c,d)\}$ but according to [16], the path $\{(a,b),(b,d)\}$ is not valid. Because we do not distinguish between a direct edge and an indirect edge in reputation graphs and trust graphs, such tests cannot be verified using our model.

4.3.2 Dynamic Evolution of Global Trust

In Section 2.2.1, we noted that trust evolves such that a negative feedback decreases trust and a positive feedback increases trust. Furthermore, trust is lost at a rate higher than it is gained. Adherence to this property prevents white-washing atttacks where attackers cheat periodically while maintaining a high reputation.

We propose the following method to evaluate this property.

Let $f_i(*,b)$ be the i'th feedback on b in feedback history $H(*,b)$, which is the list of feedbacks on b by all agents in the system and $r_i(*,b)$ be the b's global reputation score following i'th feedback.

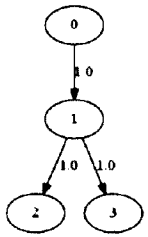
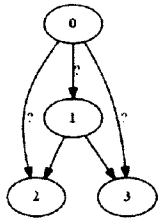
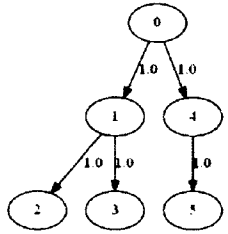
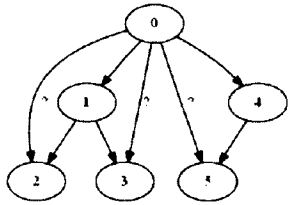
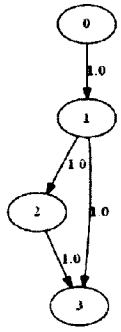
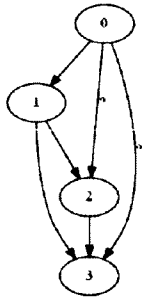
Expt	Input RG	Output RG	Expectation	Results	
				Modified EigenTrust	Appleseed
1			$r(0,1) > r(0,2)$ $r(0,2) == r(0,3)$	true	true
2			$r(0,5) > r(0,2)$ $r(0,5) > r(0,3)$	true	true
3			$r(0,3) > r(0,2)$	true	true

Table 4.4: Transitivity tests and results

Assuming both reputation scores and feedback values are on the same scale and $r(*, b)$ is a function of the feedback history $H(*, b)$, then trust must evolve according to conditions 4.3.1, 4.3.2 and 4.3.3, where:

- $\delta f = f_i(*, b) - r_{i-1}(*, b)$
- $\delta r = r_i(*, b) - r_{i-1}(*, b)$
- $\frac{\delta r}{\delta f}^+$ and $\frac{\delta r}{\delta f}^-$ are the changes in global reputation score as a result of a positive feedback change and a negative feedback change respectively

$$\delta f > 0 \Rightarrow \delta r > 0 \wedge \delta f > \delta r \tag{4.3.1}$$

$$|\frac{\delta r}{\delta f}^+| < |\frac{\delta r}{\delta f}^-| \tag{4.3.2}$$

$$\delta f = 0 \Rightarrow \delta r = 0 \tag{4.3.3}$$

That is:

1. Condition 4.3.1 verifies that if there is a positive change in feedback, then the change in reputation is also positive, but the change in reputation is less than the change in feedback.
2. Condition 4.3.2 verifies that if the magnitude of the rate of change in reputation due to positive feedbacks is less than the magnitude of the rate of change in reputation due to negative feedbacks.
3. Condition 4.3.3 verifies that if there is no change in the feedback, then there is no change in reputation, either.

Note that we cannot use $\frac{f_i(*,b)-r_{i-1}(*,b)}{r_i(*,b)-r_{i-1}(*,b)} \geq 0$ because it is undefined if $r_i(*, b) - r_{i-1}(*, b) = 0$.

Setup and results We set up two scenarios to investigate the behaviour of PeerTrust. In the first scenario, there are only two agents a and b and the feedback history $H(a,b)$ is handcrafted such that a is typically satisfied with b, except once where $f(a,b)$ is 0. In this scenario, we determine whether trust loss is greater than trust gain. In the second scenario, we introduce a third agent c and we investigate the impact of $r(c,a)$ on $r(a,b)$ and whether the above conditions are still held in this scenario.

4.3.2.1 Scenario 1: Evolution due to direct interactions only

Consider the feedback histories between two agents a and b in Table 4.5, which shows agent a was dissatisfied with b only once. Even though one can argue that b mounted a white-washing attack in which it behaved honestly in order to cheat once in a while, it is not certain, because it is also possible that a is concealing a slandering attack, where it unfairly rated b. Therefore, information such as the intention of an agent cannot be extracted from the feedback history alone. However, we can use the above conditions to characterize a trust algorithms's behaviour as to whether reputation evolves as it should.

Assuming that all agents are pre-trusted equally (i.e., $r_0(*,b) = 0.5$), Table 4.5 and Figure 4.17 show the evolution of $r_0(*,b)$. When $f_3(a,b)$ (a negative feedback) occurred, $r(*,b)$ decreased but it increased at a slower rate when $f_4(a,b)$ (a positive feedback) occured. Thus, PeerTrust respects conditions 4.3.1, 4.3.2, and 4.3.3.

In a separate experiment, we also noted that if there were two conscutive negative feedbacks on b, then $r(*,b)$ decreased to 0.5. This suggests that an attacker can know exactly how many positive feedbacks are required in order to restore his lost reputation after cheating. Therefore, PeerTrust is not resistant to a white washing attack but as expected, it is resistant to a slandering attack.

For an evaluation algorithm (evaldp) to evaluate this dynamic property of trust,

i	Assessor	Assessee	f	$r(*, b)$	δf	δr
1	a	b	1.0	1.0	0.5	0.5
2	a	b	1.0	1.0	0	0
3	a	b	0	0.66	−1.0	−0.33
4	a	b	1.0	0.75	0.33	0.08
5	a	b	1.0	0.8	0.25	0.05

Table 4.5: Evolution due to direct interactions only—Scenario 1

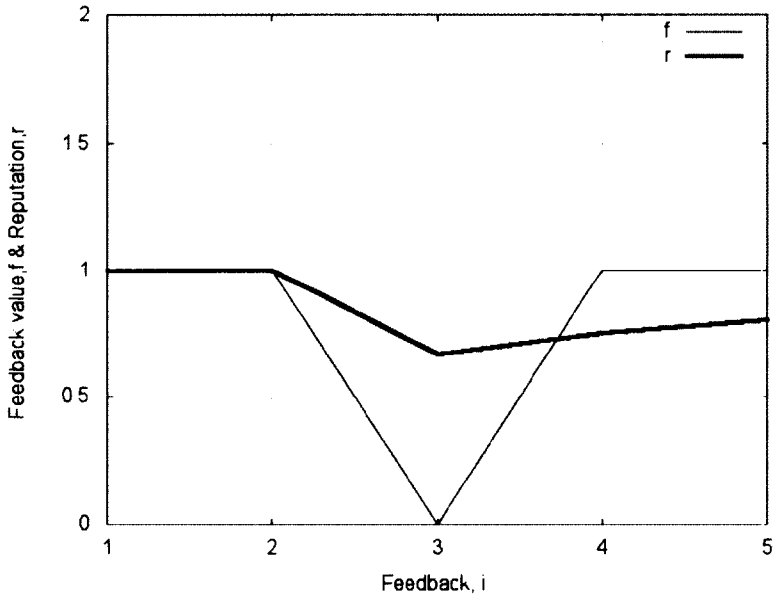


Figure 4.17: Evolution of reputation w.r.t feedback—Scenario 1

it needs the following as input:

- Agent to be evaluated
- Initial feedback history graph FHG0
- Reputation graph RG1 that corresponds to FHG0
- Another feedback history graph FHG1 with the subsequent feedback added
- Reputation graph RG2 that corresponds FHG1.

Figures 4.18 to 4.20 show the Petri Net used for this evaluation. Two instances of PeerTrust are created to process two different instances of FHG, where FHG1 is FHG0 plus the new feedback. evaldp then compares the RGs output for the two FHGs, and outputs a boolean result that is true if the conditions 4.3.1, 4.3.2, and 4.3.3 were met, or returns false otherwise.

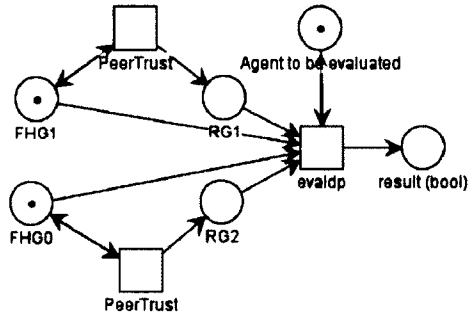


Figure 4.18: Dynamic evolution of trust—Initial Petri Net

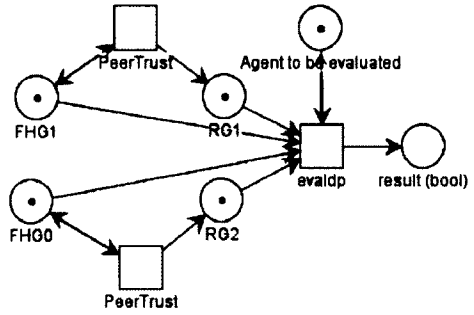


Figure 4.19: Dynamic evolution of trust—After executing PeerTrust

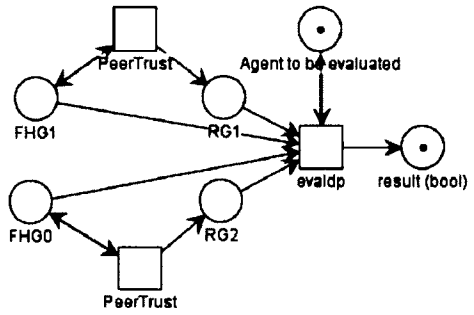


Figure 4.20: Dynamic evolution of trust—After executing evaldp

i	Assessor	Assessee	f	$r(*, b)$
1	c	a	1.0	0
2	a	b	0.9	0.9
3	a	b	0.9	0.9
4	c	a	0	0.9
5	a	b	1.0	0.93

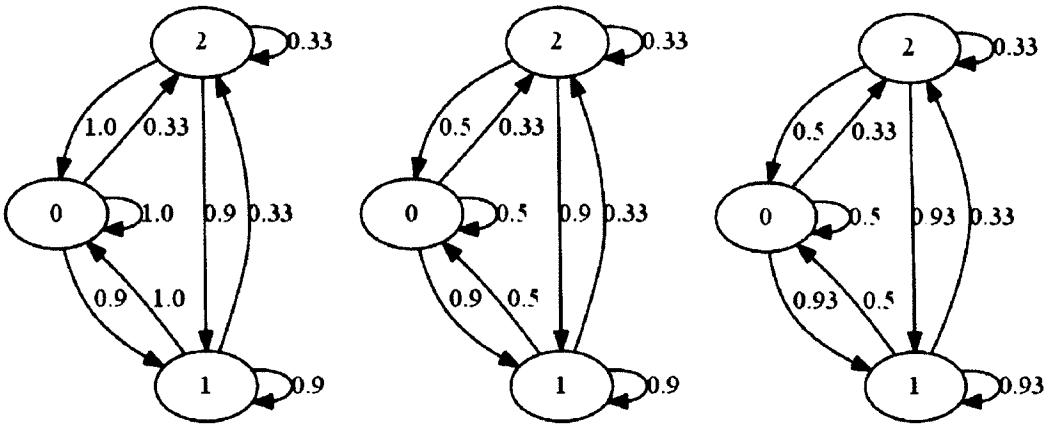
Table 4.6: Evolution due to direct and indirect interactions

4.3.2.2 Scenario 2: Evolution due to direct and indirect interactions

In this experiment, we use the feedback history provided in Table 4.6. Initially, agent c provided a positive feedback to a (f_1), but later provides a negative feedback (f_4). Because b's reputation score is dependent on a's reputation score, a negative change in a's reputation should affect b's reputation. However, results show that it is unaffected. We explain the reason below.

If we compare the reputation graphs shown in Figures 4.21(a) and 4.21(b), we note that $r(*, a)$ (denoted by an edge to node 0) changed from 1 to 0.5. However, this change did not affect $r(*, b)$ (denoted by an edge to node 1) after adding f_5. This is because of the normalization technique used by PeerTrust to calculate b's reputation as shown below (see also Equation 2.3.1):

$$
\begin{aligned}
I(b) &= \{f_2, f_3, f_5\} \\
T(b) &= \sum_{i=1}^{I(b)} S(b,i) * \frac{T(p(b,i))}{\sum_{j=1}^{I(b)} T(p(b,j))} \\
T(b) &= 0.9 * \frac{0.5}{3 * 0.5} + 0.9 * \frac{0.5}{3 * 0.5} + 1.0 * \frac{0.5}{3 * 0.5} \\
T(b) &= 0.93
\end{aligned}
\tag{4.3.4}
$$



(a) RG after f_3 (b) RG after f_4 (c) RG after f_5

Figure 4.21: Changes in reputaion graphs output by PeerTrust

Thus, PeerTrust did not propagate trust as one would have expected (i.e., if c does not trust a, it should not trust b either) and this can be easily exploited by attackers. For example, if agents a and b both are part of a collusive group, then agent c cannot recognize this and therefore cannot break away from a collusive group of attackers.

We conclude that PeerTrust also suffers from attacks due to normalization techniques.

Limitations

1. By definition, the global trust of an agent reflects the impact of feedbacks by all other agents in the system. To verify the conditions presented at the begining of this section, we only need to know the order of feedbacks for an agent, regardless of the assessor. However, verifying the evolution of local trust scores according to the above conditions is subjective. Suppose we obtained Table 4.7 from a local trust algorithm. If we only consider direct experience between a and b, then we observe that $r(a, b)$ increased from 0.1 to 0.3 despite a negative feedback (f_6), but this increase may have been contributed by f_5 and the fact that $r(a, c) = 0.9$. That is, $r(a, b)$ is an aggregated score obtained through direct and indirect

i	Assessor	Assessee	f	$r(a,b)$	$r(a,c)$	$r(c,b)$
1	a	c	0.9	-	0.9	-
2	a	b	0.6	0.6	0.9	-
3	a	b	0.7	0.65	0.9	-
4	a	b	0.2	0.1	0.9	-
5	c	b	0.9	-	0.9	0.4
6	a	b	0	0.3	0.9	0.4

Table 4.7: Local reputation example

experiences and, depending on the algorithm, different weights may be given to direct versus indirect experience [32].

4.4 Summary

In this chapter, we provided details of our prototype and evaluated trust algorithms for vulnerabilites to attacks and adherence to trust properties. Table 4.8 summarizes our results, where we categorize the algorithms we investigated according to their vulnerabilities to attacks. Because lack of adherence to trust properties can also result in attacks, we have included them in Table 4.8. To our knowledge, the following vulnerabilities are new and add to the contributions of this thesis:

- PeerTrust's vulnerability to white-washing attacks
- Vulnerabilities due to bootstrapping techniques in EigenTrust
- PeerTrust's vulnerability to normalization-based attacks
- Division-by-zero problem in PeerTrust

The experiments show the flexibility of our testbed. We show that both trust algorithms and their evaluations can fit into a single model. Thus, we conclude that

Attack	EigenTrust	PeerTrust	Appleseed
A priori trust	Yes	Yes	-
Negative vs. no feedback	Yes	No	-
Self-promoting	Yes	Yes	-
Slandering	No	No	-
Sybil	-	-	No
Normalization-based	Yes	Yes	-
Dynamic evolution based (white-washing)	-	Yes	-
Transitive property-based	No	-	No

Table 4.8: Trust algorithms and their vulnerabilities

our model for a testbed satisfies the requirements outlined in Chapter 3.1.

The next chapter highlights the contributions of this work, limitations of our model, and describes future work.

Chapter 5

Conclusion

5.1 Solution to the Research Problem

Our work showed that the state-of-art trust models vary considerably (see Table 2.1), but many of them can be fit into a single model (Section 3.3). This addresses the first research problem we stated: "A model that provides an abstraction layer for developers to incorporate existing and new reputation systems." In addition, we showed that the same model is able to evaluate reputation systems against application-independent attacks.

5.2 Contributions

The following key contributions were presented in this thesis:

- **A view of trust as a workflow, with graphs as input and output:** This view is what allowed us to accommodate the high variation between trust models.
- **A model for a testbed:** Our Petri net-based approach to model the trust process allows us to provide a executable model of the workflow above, and to

integrate it with various evaluation schemes, as shown in Section 3.4.

- **Testbed prototype:** We have implemented a prototype for a testbed based on our model (Section 4.1) and used it for evaluating some of the reputation systems we investigated in this thesis. The wide variety of evaluations shows the flexibility of both our model and its implementation.

- **Evaluation results:** Our contributions include finding new and confirming known vulnerabilities in the reputations we evaluated (Sections 4.2 and 4.3).

Derived Publications

- Chandrasekaran, P., Esfandiari, B.: A Model For a Testbed For Evaluating Reputation Systems. In Proceedings of the 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 296-303, 2011.

5.3 Limitations and Future Work

We have identified the following limitations in our work:

- **Limitations in the model**

  1. Since the feedback history graph limits itself to agent-to-agent transaction ratings, recommender systems such as Credence [15] that use agent-to-object ratings cannot be included in the testbed.
  2. The inferred trust values (discrete or continuous) must be derived either through direct agent-to-agent ratings or from direct trust values.
  3. The model must derive trust from a single context (i.e., an algorithm that attempts to extrapolate trust in context a for another context b cannot be included in our testbed).

- **Agent behaviour simulation** Our testbed does not have the capability to generate a feedback history graph using simulation of agent behaviour (stage 1 of the workflow). Such a feature would be useful for designing large scale experiments with each agent acting in a different manner.

- **Support for distrust** Distrust indicates how much an agent is not trusted (opposite of trustworthiness) and algorithms such as the distrust-aware version of Appleseed compute both trust and distrust propagation. Because our reputation graphs do not include distrust information, such algorithms cannot be evaluated using our testbed.

- **Separation of direct and indirect trust** As mentioned in Section 4.3.1, transitive rules according to [16] cannot be evaluated using our testbed.

- **Limitation in evaluation metrics** Our method of evaluating reputation systems against Sybil attacks cannot be used for global reputation systems (see Section 4.2.4). Moreover, we note that the evaluation of dynamic evolution of trust according to our interpretation of Marsh's work [2] cannot be used for local reputation algorithms. This is explained in Section 4.3.2.

In addition to addressing the above limitations in our model, future work involves building a user interface for our testbed and performing large-scale experiments. In particular, experiments using large datasets from websites such as Epinions, Advogato, and Facebook can yield interesting results.

List of Acronyms

P2P Peer-to-Peer

PKI Public Key Infrastructure

FOAF Friend of a Friend

ART Agent and Reputation Trust

TREET Trust and Reputation Experimentation and Evaluation Testbed

PGP Pretty Good Privacy

CA Certificate Authority

FHG Feedback History Graph

RG Reputation Graph

TG Trust Graph

NA Normalization algorithm

DA Discretization algorithm

ARFF Attribute-Relation File Format

List of References

[1] C. Castelfranchi, R. Falcone, and E. Lorini, "A non-reductionist approach to trust," in *Computing with Social Trust*, ser. HumanComputer Interaction Series, J. Golbeck, Ed. Springer London, 2009, pp. 45–72. [Online]. Available: http://dx.doi.org/10.1007/978-1-84800-356-9_3

[2] S. P. Marsh, "Formalising trust as a computational concept," Ph.D. dissertation, University of Stirling, Apr 1994.

[3] D. Gambetta, *Trust.* Oxford, 1990.

[4] Y. Wang and J. Vassileva, "Trust and reputation model in peer-to-peer networks," in *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, ser. P2P '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 150–. [Online]. Available: http://dl.acm.org/citation.cfm?id=942805.943810

[5] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," in *CIKM '01 Proceedings of the tenth international conference on information and knowledge management.* ACM, 2001, pp. 310–317.

[6] J. Golbeck, "Computing and applying trust in web-based social networks," Ph.D. dissertation, University Of Maryland, 2005.

[7] A. Jøsang, "Trust and reputation systems," in *Foundations of Security Analysis and Design (FOSAD) IV.* Springer, 2007, pp. 209–245.

[8] J. A. Golbeck, "Computing and applying trust in web-based social networks," Ph.D. dissertation, University of Maryland at College Park, 2005.

[9] A. Josang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support System*, vol. 43, pp. 618–644, March 2007.

[10] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *WWW '03 Proceedings of the 12th international conference on World Wide Web.* ACM, 2003, pp. 640–651.

[11] P. R. Zimmermann, *The official PGP user's guide.* Cambridge, MA, USA: MIT Press, 1995.

[12] L. Xiong and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, July 2004.

[13] R. Levien, in *Computing with social trust.* Springer, 2009, ch. Attack-resistant Trust Metrics, pp. 121–132.

[14] C.-N. Ziegler, in *Computing with social trust.* Springer, 2009, ch. On propagating interpersonal trust in social networks, pp. 133–166.

[15] K. Walsh and E. G. Sirer, "Experience with an object reputation system for peer-to-peer filesharing," in *NSDI'06 Proceedings of the 3rd conference on Networked Systems Design and Implementation*, 2006.

[16] A. Jøsang and S. Pope, "Semantic constraints for trust transitivity," in *Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling - Volume 43*, ser. APCCM '05. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2005, pp. 59–68. [Online]. Available: http://dl.acm.org/citation.cfm?id=1082276.1082284

[17] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Computing Surveys*, vol. 42, pp. 1:1–1:31, December 2009.

[18] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," in *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6.* Washington, DC, USA: IEEE Computer Society, 2000, p. 6007.

[19] J. R. Douceur, "The sybil attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01 Revised Papers from the First International Workshop on Peer-to-Peer Systems. London, UK: Springer-Verlag, 2002, pp. 251–260. [Online]. Available: http://portal.acm.org/citation.cfm?id=646334.687813

[20] R. Kerr and R. Cohen, "Smart cheaters do prosper: defeating trust and reputation systems," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, ser. AAMAS '09. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 993–1000. [Online]. Available: http://portal.acm.org/citation.cfm?id=1558109.1558151

[21] E. Carrara and G. Hogben, "Enisa position paper no. 2 reputation-based systems: a security analysis," 2007, [Accessed 31-Aug-2012]. [Online]. Available: http://www.enisa.europa.eu/activities/identity-and-trust/privacy-and-trust/reputation-systems/reputation-based-systems-a-security-analysis

[22] R. Levien and A. Aiken, "Attack-resistant trust metrics for public key certification," in *7th USENIX Security Symposium*, 1998, pp. 229–242.

[23] R. Guha, "Open rating systems," in *Proceedings of the 1st workshop on Friends of a Friend, Social Networking and the Semantic Web*, 2004, [Accessed 31-Aug-2012]. [Online]. Available: http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/open_rating_systems/wot.pdf

[24] R. Kerr and R. Cohen, "Treet: the trust and reputation experimentation and evaluation testbed," *Electronic Commerce Research*, vol. 10, pp. 271–290, December 2010.

[25] C. Girault and R. Valk, *Petri Nets for Systems Engineering*, 1st ed. Springer, 2003.

[26] A. Bidgoly and B. Ladani, "Trust modeling and verification using colored petri nets," in *Information Security and Cryptology (ISCISC), 2011 8th International ISC Conference on*, sept. 2011, pp. 1 –8.

[27] "Laerd statistics: Spearman's rank-order correlation," https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php, 2012, [Online; Accessed 20-July-2012].

[28] "Jgrapht - a free java graph library," http://jgrapht.org/, Accessed: 15-July-2012.

[29] "Attribute-relation file format (arff)," http://www.cs.waikato.ac.nz/ml/weka/arff.html", Accessed: 15-July-2012.

[30] R. Jansen, T. Kaminski, F. Korsakov, A. S. Croix, and D. Selifonov, "A priori trust vulnerabilities in eigentrust," University of Minnesota, http://www-users.cs.umn.edu/jansen/papers/fet-csci5271.pdf, Tech. Rep., 2008.

[31] N. Chiluka, N. Andrade, D. Gkorou, and J. A. Pouwelse, "Personalizing eigentrust in the face of communities and centrality attack," in *AINA '12 Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, 2012, pp. 503–510.

[32] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins, "Propagation of trust and distrust," in *Proceedings of the 13th international conference on World Wide Web*, ser. WWW '04. New York, NY, USA: ACM, 2004, pp. 403–412. [Online]. Available: http://doi.acm.org/10.1145/988672.988727