

RESEARCH

Open Access



# Toward a testbed for evaluating computational trust models: experiments and analysis

Partheeban Chandrasekaran and Babak Esfandiari\*

\*Correspondence:  
babak@sce.carleton.ca  
Department of Systems and  
Computer Engineering, Carleton  
University, 1125 Colonel By Drive,  
Ottawa, Ontario K1s5B6, Canada

## Abstract

We propose a generic testbed for evaluating social trust models and we show how existing models can fit our testbed. To showcase the flexibility of our design, we implemented a prototype and evaluated three trust algorithms, namely EigenTrust, PeerTrust and Appleseed, for their vulnerabilities to attacks and compliance to various trust properties. For example, we were able to exhibit discrepancies between EigenTrust and PeerTrust, as well as trade-offs between resistance to slandering attacks versus self-promotion.

**Keywords:** Trust testbed; Reputation; Multi-agent systems

## Introduction

### Motivation

With the growth of online community-based systems such as peer-to-peer file-sharing applications, e-commerce and social networking websites, there is an increasing need to provide computational trust mechanisms to determine which users or agents are honest and which ones are malicious. Many models calculate trust by relying on analyzing a history of interactions. The calculations can range from the simple averaging of ratings on eBay to flow-based scores in the Advogato website. 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 vulnerabilities against certain types of attacks. Thus, there is still a need for a generic testbed to evaluate and compare computational trust models.

### Overview of our solution and contributions

In this paper, we present a model and a testbed for evaluating a family of trust algorithms that rely on past transactions between agents. Trust assessment is viewed as a process consisting of a succession of graph transformations, where the agents form the vertices of the graph. The meaning of the edges depends on the transformation stage,

and can refer to the presence of transactions between the two agents or the existence of a trust relationship between them. Our first contribution is to show that with this view, existing reputation systems can be adopted under a single model, but they work at different stages of the trust assessment workflow. This allows us to present a new classification scheme for a number of trust models based on where they fit in the assessment workflow. The second contribution of our work is that this workflow can be described formally, and by doing this, we show that it is possible to model a variety of attacks and evaluation schemes. Finally, out of the larger number of systems we classified, we selected three reputation systems, namely EigenTrust [1], PeerTrust [2] and Appleaseed [3], to exemplify the range and variety of reputation systems that our testbed can accommodate. We evaluated these three systems in our testbed against simple attacks and we validated their compliance to basic trust properties. In particular, we were able to exhibit differences in the way EigenTrust and PeerTrust rank the agents, we observed the subtle interplay between slandering and self-promoting attacks (higher sensitivity to one attack can lead to lower sensitivity to the other), and we verified that trust weakens along a friend-of-a-friend chain and that it is more easily lost than gained (as it should be).

### **Organization**

This article is organized as follows: section 'Background and literature review' provides background and state of the art on trust models, attacks against them, and existing testbeds for evaluation. Section 'Problem description and model' formulates the research problem of this article and proposes our model for a testbed. Section 'Classifying and chaining algorithms' shows how some of existing trust algorithms can fit our model, and how one can combine or compare them using our model and testbed. Section 'Results and discussion' describes the implementation details of our testbed prototype and presents evaluation results of three different trust algorithms, namely EigenTrust, PeerTrust, and Appleaseed. Section 'Conclusions' concludes this article and summarizes the contributions and limitations of our work.

## **Background and literature review**

### **Social trust models**

Trust management systems aid agents in establishing and assessing mutual trust. However, the actual mechanisms used in these systems vary. For example, public key infrastructures [4] rely on certificates whereas reputation-based trust management systems are based on experiences of earlier direct and indirect interactions [5].

In this paper we will focus on social trust models based on reputation. 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.

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 it would in certificate authority-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.

#### ***Nature of input***

Various inputs are used by social trust algorithms to measure the trustworthiness of agents. In EigenTrust [1], PeerTrust [2], TRAVOS [6] and Beta Reputation System (BRS) [7], 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 Aberer and Despotovic's system [5]<sup>1</sup>, agents may file complaints (can be seen as dissatisfaction) about each other after a transaction. In Advogato [8], whose goal is to discourage spam on its blogging website, users explicitly certify each other as belonging to a particular level in the community. Trust algorithms may also directly use trust scores among agents to compute an aggregated trustworthiness of agents, as in TidalTrust [9] and Appleseed [3]. In the specific context of P2P file-sharing, Credence [10] 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.

#### ***Direct vs. indirect trust***

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 gossiping to poll agents with whom the truster has had interactions in the past.

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 [11]. As mentioned earlier, reputation systems use different inputs (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 and TRAVOS use satisfaction ratings to calculate direct trust scores, which they then use 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 Aberer use simple averaging of ratings, TRAVOS and BRS use the beta probability density function, and Appleseed uses the Spreading Activation model.

#### ***Global vs. local trust***

The trust algorithm may output a *global* trust score or a *local* trust score [3, 12]. 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 and thus each truster may trust an agent differently. In our survey, we found PeerTrust, EigenTrust, and Aberer to be global trust algorithms whereas TRAVOS, BRS, Credence, Advogato, TidalTrust, Appleseed, Marsh [13] and Abdul-Rahman [14] are local trust algorithms.

**To trust or not to trust**

Once the trust score is calculated, it can be used to decide whether to trust the agent. It can be as simple as comparing the trust score against a threshold: if the trust score is above a certain threshold, then the agent is trusted. Marsh [13], and Aberer [5] use thresholding techniques. If the trust algorithm outputs normalized trust scores of agents as in EigenTrust, 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, 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, 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. Does the trust algorithm use only direct experience or does it also rely on third party recommendations?
3. Is the trust score of an agent global or local?
4. How does one decide whether to trust an agent?

Given the above discussion, and to assess the scope of our testbed, we propose to model, evaluate and compare three algorithms from fairly different families. The next sections provide detailed descriptions of the trust models we selected and that we implemented in our testbed. The details are given to help understand the output of our experiments, but readers familiar with EigenTrust, PeerTrust and/or AppleSeed may skip those respective sections.

**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 Eq. 2.1, where:

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

$$T(u) = \sum_{i=1}^{I(u)} S(u, i) \times \frac{T(p(u, i))}{\sum_{j=1}^{I(u)} T(p(u, j))} \quad (2.1)$$

PeerTrust also provides a method for calculating local trust scores. In both local and global trust score computations, the trust score is compared against a threshold to decide whether to trust or not.

**EigenTrust**

Agents in EigenTrust rate transactions as satisfactory or unsatisfactory [1]. These transaction ratings are used as input, to calculate a local direct 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 Eq. 2.2, where  $t_{ij} \in \{+1, -1\}$  is the transaction rating, and  $s_{ij}$  is the sum of ratings.

$$\begin{aligned} s_{ij} &= \sum_{T_{ij}} tr_{ij} \\ c_{ij} &= \frac{\max(s_{ij}, 0)}{\sum_k \max(s_{ik}, 0)} \end{aligned} \quad (2.2)$$

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.

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 Eq. 2.3.

$$t_{ik} = \sum_j c_{ij} c_{jk} \quad (2.3)$$

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  $\vec{t}_i$  the vector containing  $t_{ik}$ , then,

$$\vec{t}_i = C^T \vec{c}_i \quad (2.4)$$

By asking a friend's friend's opinion, Eq. 2.4 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 Eq. 2.4 becomes Eq. 2.5, where  $n$  is the number of hops from  $i$ .

$$\vec{t}_i = (C^T)^n \vec{c}_i \quad (2.5)$$

The trust scores of the agents converge to a global value irrespective of the trustee.

Because EigenTrust outputs global 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.

### **Appleseed**

Appleseed is a flow-based algorithm [3]. Assuming that we are given a directed weighted graph with agents as nodes, edges as trust relationships, and the 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.

The trust propagation from agent  $a$  to agent  $b$  is determined using Eq. 2.6, 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.

$$in(b) = decay \times \sum_{(a,b) \in E} in(a) \times \frac{weight(a,b)}{\sum_{(a,c) \in E} weight(a,c)} \quad (2.6)$$

The trust of an agent  $b$  ( $trust(b)$ ) is then updated using Eq. 2.7, where the decay factor ensures that trust in an agent decreases as the path length from the seed increases.

$$trust(b) := trust(b) + (1 - decay) \times in(b) \quad (2.7)$$

Generally, trust graphs have loops, which makes Eq. 2.7 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  $x \in A_i$ :

$$\forall x \in A_i : trust_i(x) - trust_{i-1}(x) \leq T_c \quad (2.8)$$

After Eq. 2.7 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 brief survey shows, the trust models vary in terms of their input, output, and the methods they use. To evaluate and compare them, testbeds are needed. In the next section we take a look at existing testbeds.

### Testbeds

We investigated two testbed models, namely Guha's [15] and Macau [16], and two testbed implementations, namely ART [17] and TREET [18], which are used to evaluate trust algorithms. This section provides details of our investigation.

#### Guha

Guha [15] proposes a model to capture document recommendation systems, where trust and reputation play an important role. The model relies on a graph of agents where the edges can be weighted based on their mutual ratings, and a rating function for documents by agents. Guha then discusses how trust can be calculated based on those ratings, and evaluates a few case studies of real systems that can be accommodated by the model.

Guha's model can capture trust systems that take a set of documents and their ratings as input (such as Credence [10]), but it cannot accommodate systems where the only input consists of direct feedbacks between agents, such as in PeerTrust (global) [2] or EigenTrust [1]. Also, the rating of documents is itself an output of Guha's model, and that is often not the purpose or output of many more general-purpose trust models.

In short, document recommendation systems can be viewed as a specialization or subclass of more general trust systems, and Guha's model is suitable for that subclass.

#### Macau

Hazard and Singh's Macau [16] is a model for evaluating reputation systems. The authors distinguish two roles for any agent: a *rater* that evaluates a *target*. Transactions are viewed as a *favor* provided by the target to the rater. The target's reputation, local to each rater-target pairing, is updated after each transaction and depends on the previous reputation value. The target's payoff in giving a favor is also dependent on its current reputation but also on its belief of the likelihood that the rater will in turn return the favor in the future.

Based on the above definitions, the authors define a set of desirable properties for a reputation system:

- **Monotonicity:** given two different targets  $a$  and  $b$ , the computed reputation of  $a$  should be higher than that of  $b$  if the predicted payoff of a transaction with  $a$  is higher than with  $b$ .
- **Unambiguity and convergence:** the reputation should converge over time to a single fixpoint, regardless of its initial value.
- **Accuracy:** this convergence should happen quickly, thus minimizing the total reputation estimation errors in the meantime.

Macau thus captures an important stage in trust assessment, i.e. the update of one-to-one trustworthiness based on past transactions. It has been used to evaluate, in terms of their compliance to the properties defined above, algorithms such as TRAVOS [6] and the Beta Reputation System (BRS) [7] that model positive and negative experiences as random variables following a beta probability distribution. The comparison of trust models relying on the beta distribution and their resilience to various attacks has also recently been explored in [19].

### **ART**

The Agent Reputation and Trust testbed (ART) [17] provides an open-source message-driven simulation engine for implementing and comparing the performance of reputation systems. ART uses art painting 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 a specific era, it may not be able to provide appraisals for paintings from 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.

ART is best suited for evaluating trust calculation schemes from a first person point of view. It is not meant as a platform for testing trust management as a service provided by the system. For example, to evaluate EigenTrust in ART, one would either need to considerably modify ART itself (for the centralized version of EigenTrust) or to require cooperation from the participating agents and an additional dedicated distributed infrastructure (for the distributed version). Furthermore, as also pointed out in [16] and [20], the comparison of the performance of different agents is not necessarily based on their correct ability to assess the reputation of other agents, but rather based on how well they model and exploit the problem domain.

### **TREET**

The Trust and Reputation Experimentation and Evaluation Testbed (TREET) [18] 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, an agent is introduced for each departing agent. 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 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

Multiple seller accounts are needed to orchestrate a Proliferation Attack, but TREET does 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 [21], 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.

### Summary

Trust is a tool used in the decision-making process and it can be computed. There are many models based on social trust that attempt to aid agents in making rational decisions. However, these models vary in terms of their input and output requirements. This makes evaluations against a common set of attacks difficult.

### Problem description and model

Our goal is to have a testbed that is generic enough to accommodate as many trust management systems and models as possible. Our requirements are:

1. A model that provides an abstraction layer for developers to incorporate existing and new 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 this section, we introduce an abstract model for trust management systems. This model will be the foundation of our testbed. Our model is essentially based on the following stages:

1. In stage 1 of the trust assessment process, the feedback provided by agents on other agents is represented as a *feedback history graph*.



2. In stage 2, a *reputation graph* is produced, where the weight of an arc denotes the reputation of the target agent. “Reputation” here follows [14], as “an expectation about an individual’s behavior based on information about or observations of its past behavior”. It is viewed as an estimation of trustworthiness based on a combination of direct and indirect feedback.
3. In the final stage, a *trust graph* is produced, where the existence of an arc implies trust in the target agent. We take “trust” here to mean the “belief by agent A that agent B is trustworthy” [2, 22], and so it is boolean and subjective in our model.

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

### 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  $b$ , where  $a$  and  $b$  belong to the set  $A$  of all the agents in the system. The list of  $n$  feedbacks by  $a$  on  $b$ ,  $FHG(a, b)$ , is called a *feedback history*, represented as follows:

$$FHG(a, b) \mapsto (f_1(a, b), f_2(a, b), \dots, f_n(a, b)) \quad (3.1)$$

The feedback  $f_i(a, b)$  indicates the  $i$ th 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  $\mathbb{R}$ . 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.

If  $A$  is the set of agents,  $E$  is the set of labelled arcs  $(a, b)$ , and the label is  $FHG(a, b)$  when  $FHG(a, b) \neq \emptyset$ , then the feedback histories for all agents in  $A$  are represented in a directed and labelled graph called Feedback History Graph (FHG)<sup>2</sup>,  $FHG = (A, E)$ :

$$FHG : A \times A \rightarrow \mathbb{R}^{N^*} \quad (3.2)$$

Note that we have not included timestamps associated with each feedback (which would be useful for, among other things, running our testbed as a discrete event simulator), but our model can be expanded to accommodate it.

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

### Stage 2—obtain reputation graph

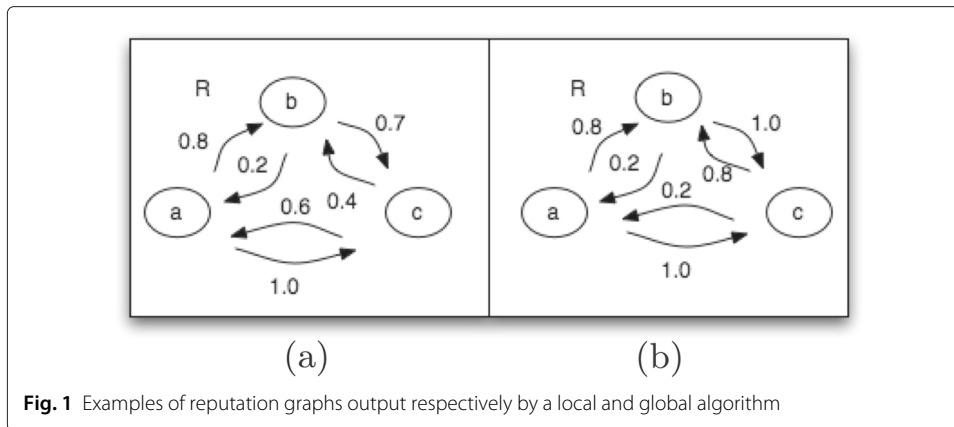
A Reputation Graph (RG),  $RG = (A, E')$ , is a directed and weighted graph, where the weight on an arc,  $RG(a, b)$ , is the trustworthiness of  $b$  from  $a$ ’s perspective:

$$RG : A \times A \rightarrow \mathbb{R} \quad (3.3)$$

The edges are added by computing second and  $n$ th-hand trust 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 value of the weight of the edges, however, depends on the particular trust algorithm).

Reputation algorithms may also exhibit the reflexive property by adding looping arcs to indicate that the truster trusts itself to a certain degree for a particular task [1–3].

The existing literature categorizes reputation algorithms into two groups: *local* and *global* (Figs. 1(a) and (b), respectively) [3, 5]. Global algorithms assign a single reputation score to each agent. Therefore, if a global algorithm is used, then the weights of the



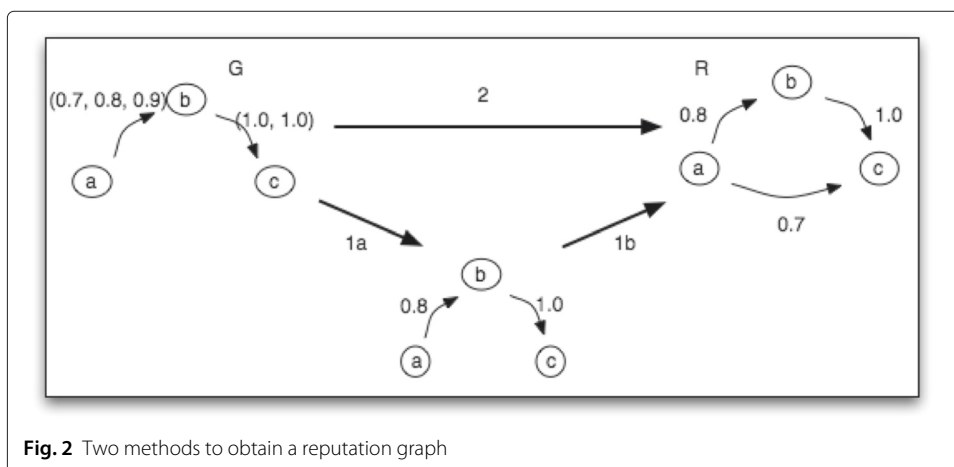
incoming arcs of an agent should be the same, as shown in Fig. 1(b) (although for clarity's sake we will often present the graph simply as a ranking of agents in the rest of this article). There is no such property for local algorithms.

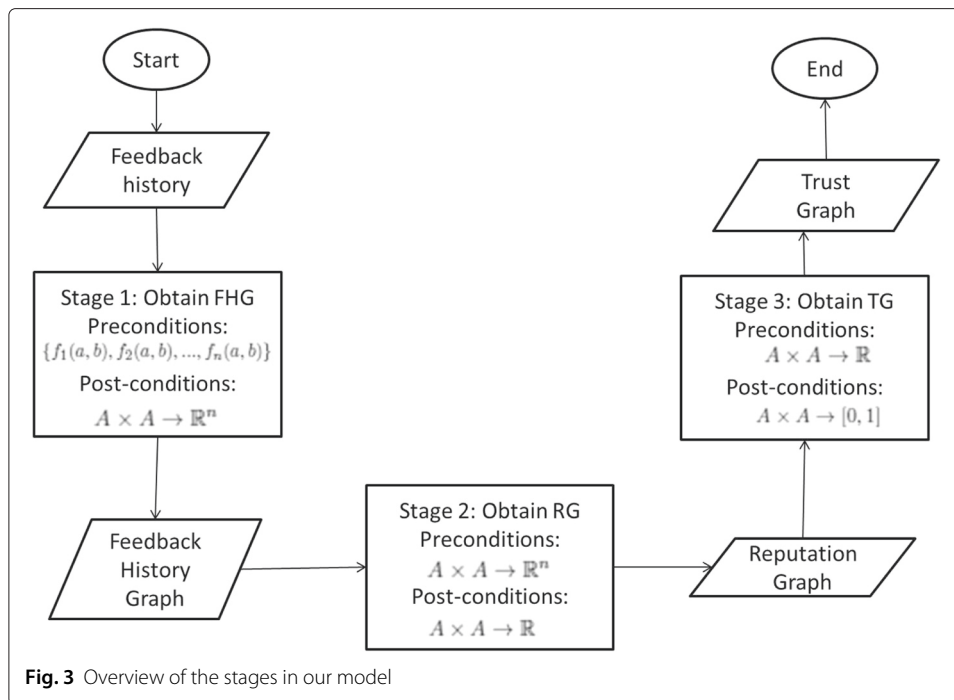
Reputation algorithms may also differ in how the graphs is produced. One method is to first calculate one-to-one scores of agents using direct feedbacks and then use them to calculate the trustworthiness of agents previously unknown to the truster (e.g., EigenTrust). This is shown as 1a and 1b in Fig. 2. The other method (#2 in Fig. 2) skips the intermediate graph in the aforementioned method and produces a reputation graph (e.g., PeerTrust).

**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 (see [23] for an example), which can also be expressed as a graph. We refer to this directed graph as the Trust Graph (TG)  $TG = (A, F)$ , where a directed edge  $ab \in F$  represents agent  $a$  trusting agent  $b$ .

To summarize our model, we can represent the stages as part of a workflow as illustrated in Fig. 3.



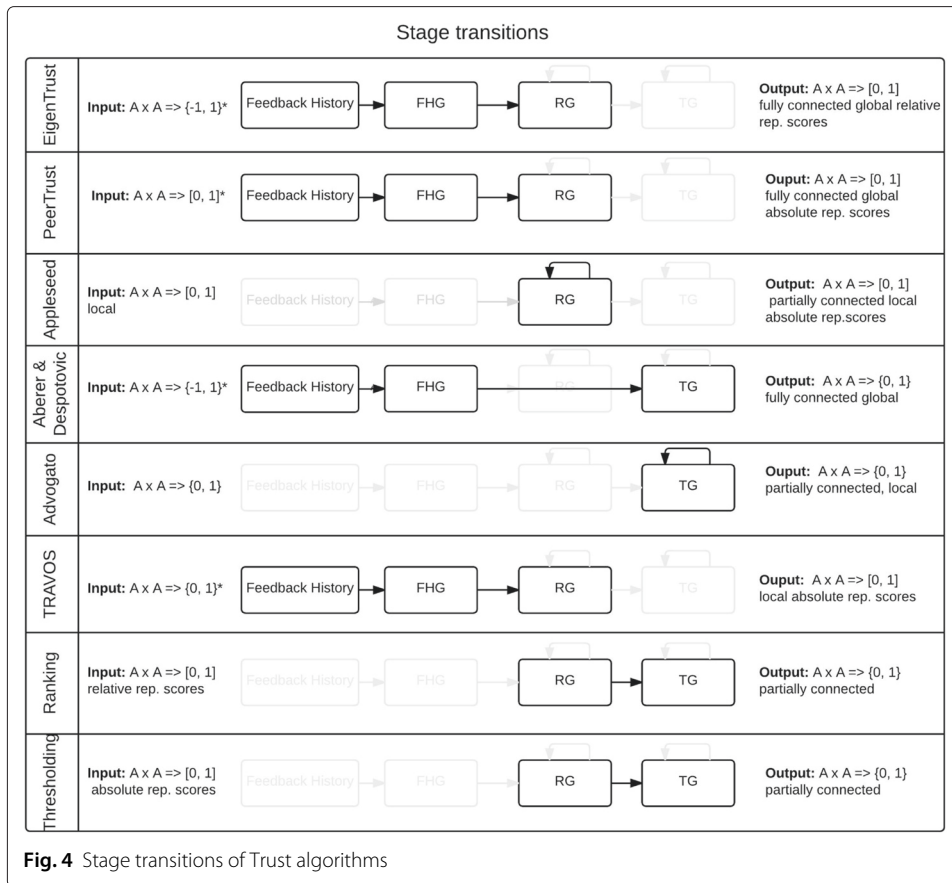


In the next section, we see at what stages in our model do various algorithms fit, and describe criteria for chaining different algorithms.

### Classifying and chaining algorithms

By refactoring the trust models 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 an *FHG* with edge labels in  $\{0, 1\}^*$  as input and outputs an *RG* with edge labels in  $[0, 1]$ . PeerTrust, on the other hand, takes an *FHG* with edge labels in  $[0, 1]^*$  as input and outputs an *RG* with edge labels in  $[0, 1]$ . Meanwhile, Appleseed requires an *RG* with edge labels in  $[0, 1]$  as input and outputs another *RG'* in the same codomain. It is also possible for an algorithm to skip some stages. For example, according to our model, Aberer [5] skips stage 2 and does not output a reputation graph. One can also represent simple mechanisms to generate a trust graph by applying a threshold on reputation values (as output for example by EigenTrust), or by selecting the top *k* agents. This stage transitions of algorithms are depicted<sup>3</sup> in Fig. 4. In addition to the existing classification criteria in the state of the art, trust algorithms can now be classified according to their stage transitions (i.e., from one stage to another as well as transitioning within a stage) as shown in Table 1.

It is important to 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.



As we can see, each step of the trust assessment process can be viewed as a graph transformation function, and we can use this functional view to easily describe evaluation mechanisms as well. Suppose an experimenter wants to compare PeerTrust and EigenTrust. The inputs and outputs of these algorithms are semantically different. To match the input, we can use a function that discretizes continuous feedback values  $f(a, b)$  in  $[0, 1]$  to  $\{-1, 1\}$ , using some threshold  $t$ :

**Table 1** A classification for trust models

Trust Algorithm	Stage Transitions	Input	Global or Local	Absolute or Relative Reputation Scores
EigenTrust	0 → 2	satisfaction ratings	global	relative
PeerTrust	0 → 2	satisfaction ratings	global	absolute
AppleSeed	2 → 2	reputation scores	local	absolute
Aberer & Despotovic	0 → 3	complaints	global	N/A
Advogato	3 → 3	certificates	local	N/A
TRAVOS	0 → 2	satisfaction ratings	local	absolute
Ranking	2 → 3	reputation scores	N/A	relative
Thresholding	2 → 3	reputation scores	N/A	absolute

$$\begin{aligned}
 & \text{discretize}_t: [0, 1] \rightarrow \{-1, 1\} \\
 & x \mapsto \text{discretize}_t(x) = \begin{cases} -1 & \text{if } x \leq t \\ 1 & \text{if } x > t \end{cases} \quad (4.1)
 \end{aligned}$$

To lighten the notations, in what follows we will adopt a default threshold of 0.5 and drop  $t$ . We will also, in an abuse of notation, actually use  $\text{discretize}: FHG \rightarrow FHG$ , which applies the function defined in 4.1 to every feedback of every edge of the graph.

Let us turn to the output now. Recall that the output of EigenTrust is a relative global score while PeerTrust's is also global but absolute. To make the outputs more directly comparable, we can use a normalization function on PeerTrust's output, ensuring that the sum of outgoing weights for each agent is 1. It takes a reputation graph  $RG2$  with edge labels in  $[0, 1]$  as input, and outputs another reputation graph  $RG3$  with edge labels in  $[0, 1]$ , but where the reputation scores are relative to one another. This is achieved using Eq. 4.2, where  $N(a)$  is the set of agents adjacent to  $a$  via outgoing edges.

$$\begin{aligned}
 & \text{normalize}: A \times A \rightarrow [0, 1] \\
 & (a, b) \mapsto \text{normalize}(a, b) = \frac{RG(a, b)}{\sum_{c \in N(a)} RG(a, c)} \quad (4.2)
 \end{aligned}$$

Again, in an abuse of notation we will also refer in what follows to  $\text{normalize}$  as the function in  $RG \rightarrow RG$  that applies 4.2 to every edge of the graph. Note again that normalization isn't strictly speaking a necessary step if all one is concerned about is the ranking of the agent, and not the values attached to each agent, especially since the semantics of these values are still ultimately attached to the algorithm that is computing them. Normalization does not affect the ranking. Either way, Spearman's rank correlation coefficient [24] can finally be used to compare the rankings output by global trust functions such as EigenTrust and PeerTrust. 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:  $\text{spearman}: RG \times RG \rightarrow [-1, 1]$ .

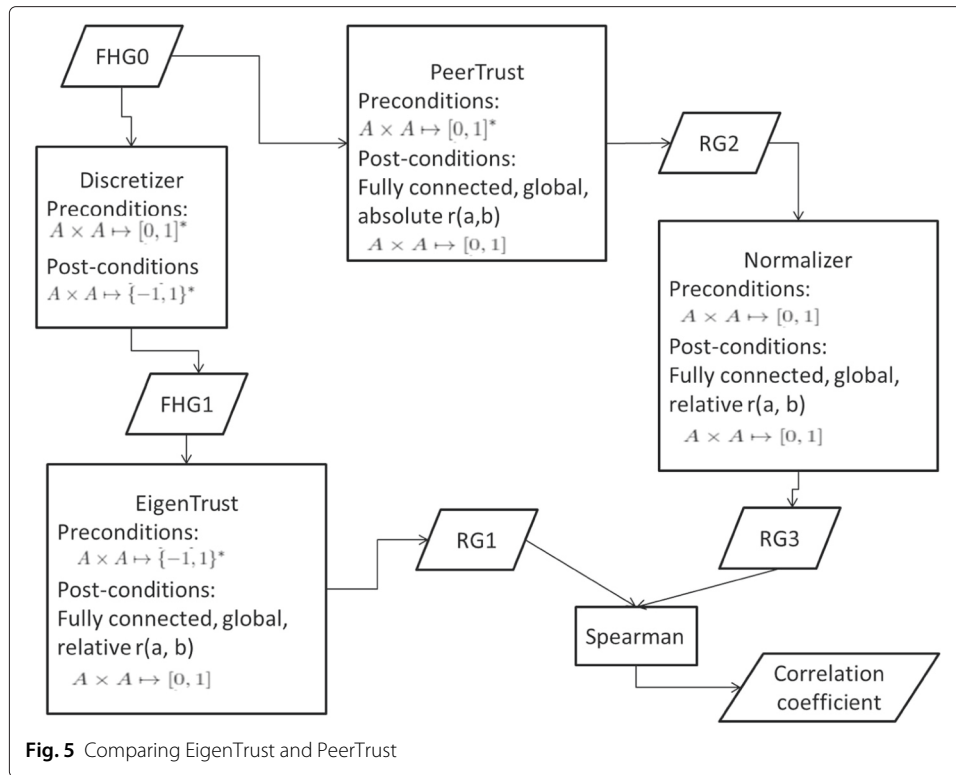
The choice of discretization and normalization methods are important and they may introduce a bias when comparing two algorithms. For the purpose of demonstrating our model, we will assume that our choices are good enough.

By combining these algorithms, we can represent the workflow as shown in Fig. 5.

To more clearly capture trust management algorithms as a process consisting of graph transformation operations, and to be able to express them in our experiments as such, we introduce named functions that make these transformations explicit. Here is an illustration of the convention we will follow: the function that captures EigenTrust's transformation from an  $FHG$  to an  $RG$  will be named  $f2r_{\text{eigentrust}}$ , where "f" stands for  $FHG$  and "r" for  $RG$ . Hence, in functional programming terms, the experiment comparing EigenTrust and PeerTrust using a given feedback history graph  $FHG0$  can be specified very simply as follows:

$$\text{spearman}(f2r_{\text{eigentrust}}(f2f_{\text{discretize}}(FHG0)), r2r_{\text{normalize}}(f2r_{\text{peertrust}}(FHG0)))$$

In the current implementation of our testbed, we assume that the pre-condition checks as per Table 1 are enforced by the algorithms rather than the testbed. However, this can be modified easily in the future.



In the next section, we present simple experiments that we ran to analyze trust algorithms using the model described in this section.

**Results and discussion**

Using the prototype testbed that we implemented following the model presented in the previous section, we conducted several experiments and analyzed their results on three trust algorithms, namely EigenTrust, PeerTrust, and Appleseed. The experiments that we present in this section are grouped into two categories: vulnerability assessments and trust properties assessments. But first we give a general description of the implementation.

**Implementation**

A prototype was designed and built in Java to test the model described in Section ‘Problem description and model’. The two main components of the testbed are graphs and algorithms. A *Graph* can be a feedback history graph, a reputation graph or a trust graph. These graphs follow our model as described in the previous section.

The graphs can be populated either programatically or by providing a file following the Attribute-Relation File Format (ARFF) used in the Weka machine learning toolkit (ARFF was chosen for its simplicity, but other formats could obviously be accommodated as well in the future if needed). 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 1:

**Listing 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
```

---

In our prototype, even though the algorithms are implemented using object-orientation, they are ultimately presented to the experimenter as functions that take a graph as input and return a graph as the output, to fully conform to the model we have presented, and to make the experiments simple and intuitive to express. Other utility functions, such as evaluation, discretization and comparison, are also provided. See Listing 2 for an example.

**Listing 2** How to create a simple experiment in our testbed

---

```
import static trust.Algorithms.*;
FHG fhg = new FHG("ex.arff");
eigenTrust(discretize(fhg)); //the experiment!
```

---

The testbed's source code is open and is available on Google Code<sup>4</sup>. To ensure that our algorithm implementations match the authors' intentions and our own understanding, we also provide a set of unit tests taken from examples found in the literature as well as our own additional scenarios.

### Vulnerability assessments

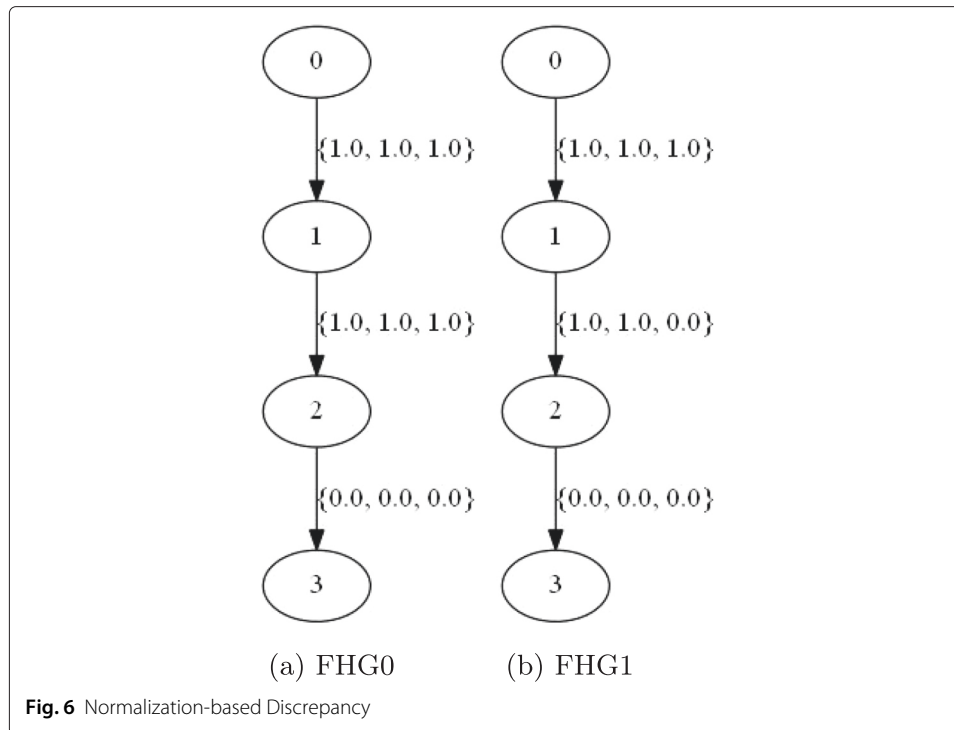
The security of trust algorithms is measured by their resistance to attacks. In this section, we subject them to attacks and assess their vulnerabilities. We will purposely try to come up with the simplest attack scenarios that can exhibit the properties we are looking for. Thirunarayan et al. [19] follows a similar approach but restricted to trust algorithms relying on the beta probability distribution.

#### Normalization-based attack

**Setup** In this first experiment, we want to exhibit whether two different trust algorithms (in this case, EigenTrust vs PeerTrust) may output different rankings given the same input. For this we investigate how reputation is affected by the number of good feedbacks versus the bad feedbacks. Suppose we were given FHG0 in Fig. 6(a) and FHG1 in Fig. 6(b). Since there are more good 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.

We normalize the output of PeerTrust simply to make the scale of the reputation numbers similar to EigenTrust (we do not claim that the semantics are the same). And as EigenTrust requires feedbacks to be either positive (+1) or negative (-1), we first need to convert the 1.0 values in the FHGs to +1 and the 0.0 values to -1, hence the use of the discretizer function in the experiment specifications below:

$$\begin{aligned}
 & spearman(f2r_{eigenTrust}(f2f_{discretize}(FHG0)), f2r_{eigenTrust}(f2f_{discretize}(FHG1))) \\
 & spearman(f2r_{peerTrust}(FHG0), f2r_{peerTrust}(FHG1)) \\
 & spearman(f2r_{eigenTrust}(f2f_{discretize}(FHG0)), f2r_{peerTrust}(FHG0)) \\
 & spearman(f2r_{eigenTrust}(f2f_{discretize}(FHG1)), f2r_{peerTrust}(FHG1))
 \end{aligned}$$



**Results** Comparing the rankings in Table 2, we note that EigenTrust reports no change in agent 2’s rank, whereas in PeerTrust its rank has changed from 1 to 2 (Spearman’s coefficient = 0.83).

Why did agent 2’s rank not change in the EigenTrust experiment? During the initial calculation of its *normalized local trust values* (used for the calculation of the reputation graph) for a given agent, EigenTrust essentially calculates the difference between positive and negative feedbacks for each neighbour and then it normalizes it over all its neighbours. In our case, agent 1 only has agent 2 as its neighbour and so the normalization process leads to a loss of information, namely the number of positive 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 [1]. But PeerTrust does not suffer from this problem, because it does not attempt to perform a sum of positive and negative feedbacks in this fashion.

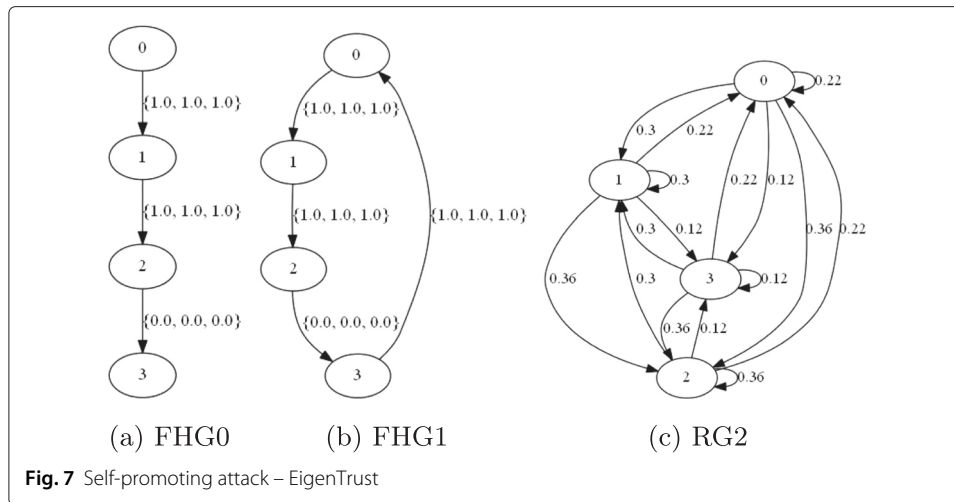
**Self-promoting attack**

**Setup** Suppose the least trustworthy agent rates a newcomer (i.e., one that had received no feedback yet) highly. If the newcomer becomes more trustworthy than before the

**Table 2** Rankings before and after slandering in retort (reputations in brackets)

	Agent 0	Agent 1	Agent 2	Agent 3
EigenTrust(FHG0)	3 (0.16)	2 (0.29)	1 (0.39)	3 (0.16)
EigenTrust(FHG1)	3 (0.16)	2 (0.29)	1 (0.39)	3 (0.16)
PeerTrust(FHG0)	3 (0.11)	1 (0.44)	1 (0.44)	4 (0.00)
PeerTrust(FHG1)	3 (0.13)	1 (0.52)	2 (0.35)	4 (0.00)





rating, then this can be exploited maliciously, paving the way to collusive self-promoting attacks [12]. Conversely, if the newcomer becomes less trustworthy than before, then the newcomer is vulnerable to a slandering attack. Consider FHG1 in Fig. 7(b) which is based on FHG0 in Fig. 7(a), after agent 3 rates agent 0 highly. The experiment set-up is therefore as follows:

$$spearman (f2r_{eigenTrust} (f2f_{discretize}(FHG0)), f2r_{eigenTrust} (f2f_{discretize}(FHG1)))$$

and the output can be seen in Fig. 7(c).

**Results** Looking at Table 3, we see that agent 0’s reputation relative to agent 3 increased. Note that it is possible that agent 3 genuinely rated agent 0 highly, in which case EigenTrust’s output is a good assessment, otherwise it might have missed a case of collusive self-promoting attack. It is also worth noting that this experiment breaks PeerTrust, due to a division-by-0 problem in an equation. 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. The division by agent 3’s reputation is the problem here.

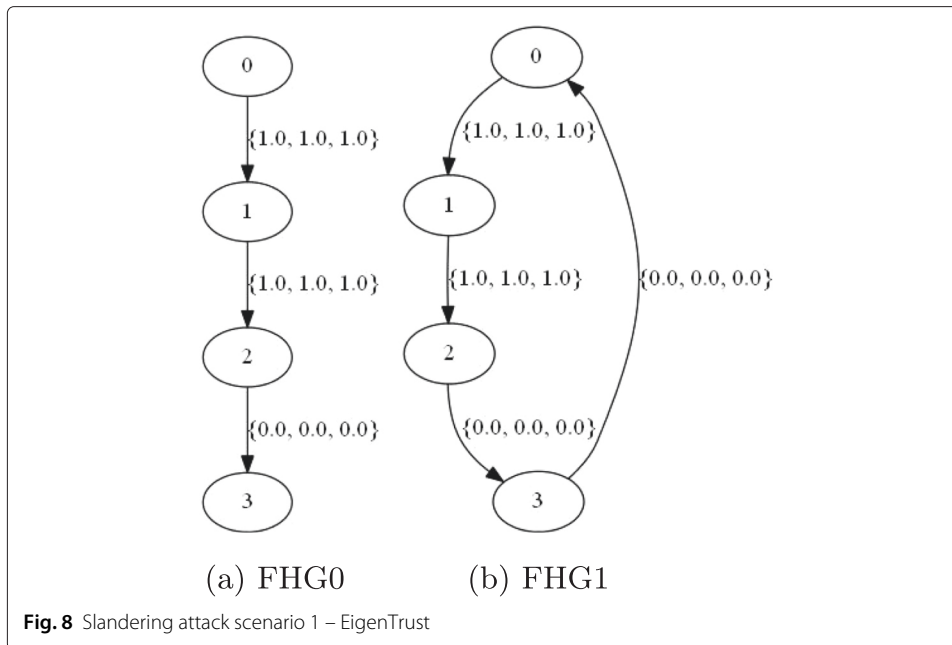
**Slandering attack, scenario 1**

**Setup** When the least trustworthy agent (agent 3) gives a newcomer (agent 0) a very low rating, it can either be a genuine rating or a slandering attempt [12]. The reputation algorithm may resist slandering by not decreasing agent 0’s reputation, but the risk or tradeoff would be the possibility of ignoring a genuine assessment. The inputs for this experiment are FHG0 and FHG1 as shown in Fig. 8, and the experiment is again therefore specified as follows:

$$spearman (f2r_{eigenTrust} (f2f_{discretize}(FHG0)), f2r_{eigenTrust} (f2f_{discretize}(FHG1)))$$

**Table 3** Ranking before and after low feedback to newcomer (reputations in brackets)

	Agent 0	Agent 1	Agent 2	Agent 3
EigenTrust(FHG0)	3 (0.16)	2 (0.29)	1 (0.39)	3 (0.16)
EigenTrust(FHG1)	3 (0.22)	2 (0.3)	1 (0.36)	4 (0.12)



**Results** Comparing the rankings in Table 4, we observe that agent 0’s reputation and global rank has not changed. If agent 3 has indeed provided a dishonest negative feedback, then we can say that EigenTrust output is a correct assessment. However, it is also possible that agent 3 is the one being slandered (by agent 2) and has provided a genuine negative feedback, in which case, we would expect agent 0’s rank to decrease. Thus, we can only say that EigenTrust is insensitive to bad feedbacks and therefore resists the slandering of a newcomer. Note that again PeerTrust would produce invalid results for the same reason as previously.

**Slandering attack, scenario 2**

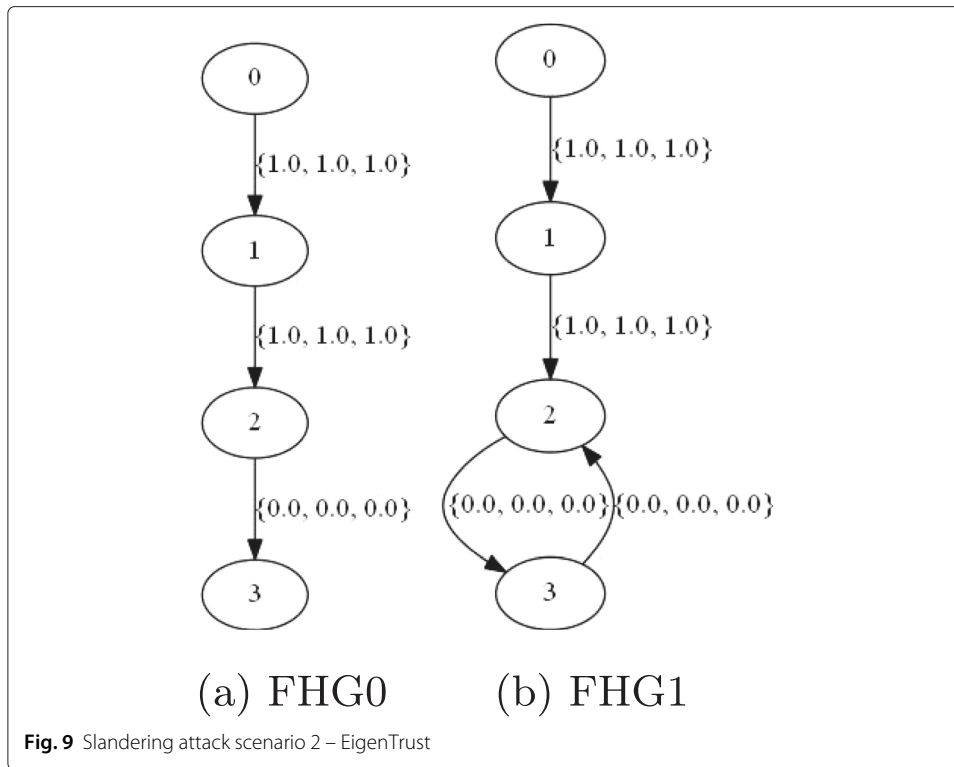
**Setup** Now going back to the initial situation (Fig. 9(a)), what happens if, as shown in FHG1 in Fig. 9(b), agent 3 decides to direct its bad feedbacks to agent 2 instead of agent 0? Looking purely at the feedbacks, one can make the following guesses:

- 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 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

**Table 4** Ranking before and after high feedback to newcomer (reputations in brackets)

	Agent 0	Agent 1	Agent 2	Agent 3
EigenTrust(FHG0)	3 (0.16)	2 (0.29)	1 (0.39)	3 (0.16)
EigenTrust(FHG1)	3 (0.16)	2 (0.29)	1 (0.39)	3 (0.16)



to a white-washing attack. Both EigenTrust and PeerTrust can be used to run this experiment, which is specified as follows:

```

spearman (f2r_eigentrust (f2f_discretize (FHG0)), f2r_eigentrust (f2f_discretize (FHG1)))
spearman (f2r_peertrust (FHG0), f2r_peertrust (FHG1))
spearman (f2r_eigentrust (f2f_discretize (FHG1)), r2r_normalize (f2r_peertrust (FHG1)))
    
```

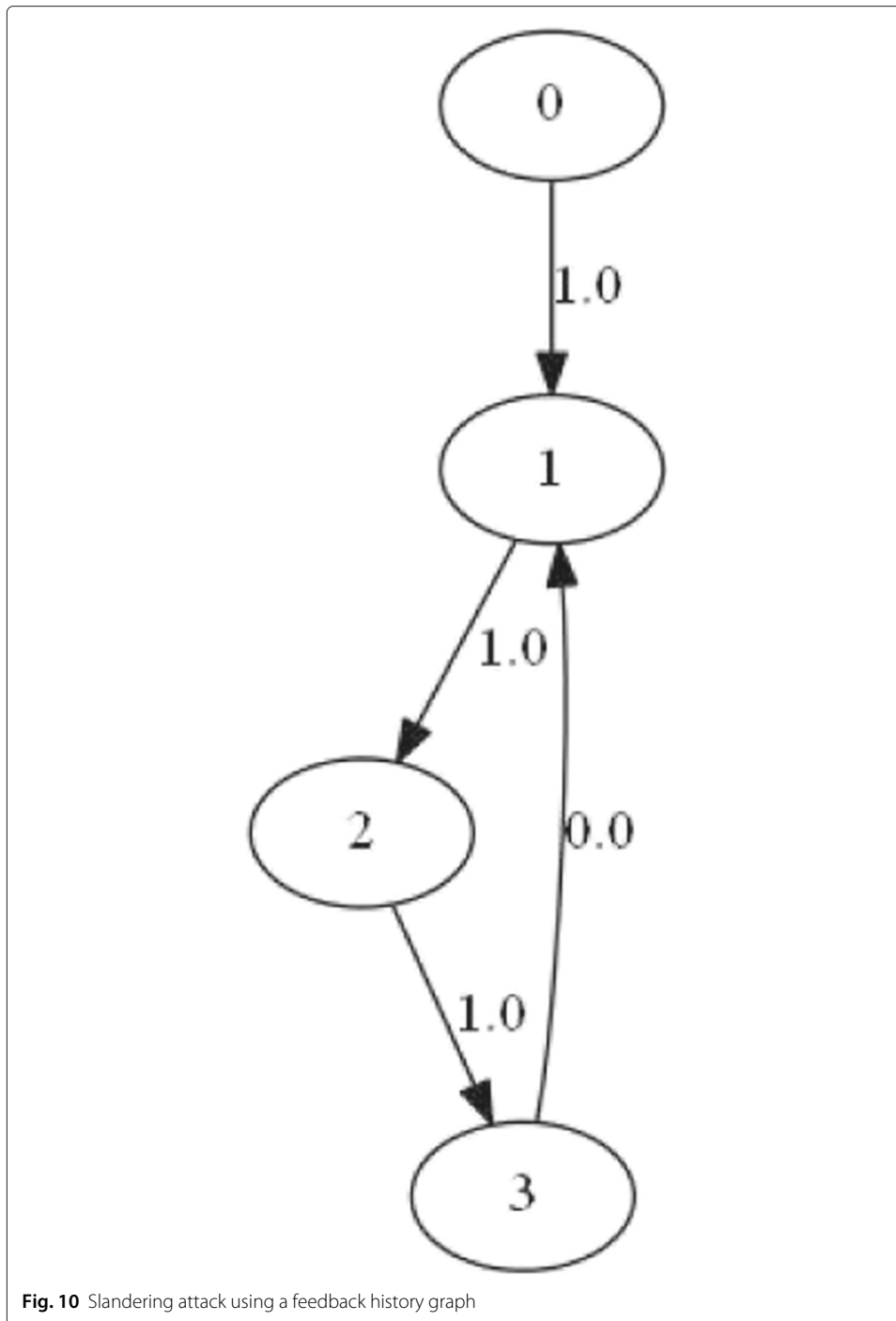
**Results** The rankings in Table 5 show that both EigenTrust and PeerTrust are resistant to this kind of slandering attack, but, as mentioned earlier, they are vulnerable to white-washing attacks (because the negative feedbacks by agent 3 did not lower agent 2’s ranking).

**Slandering + Sybil attack for local trust algorithms**

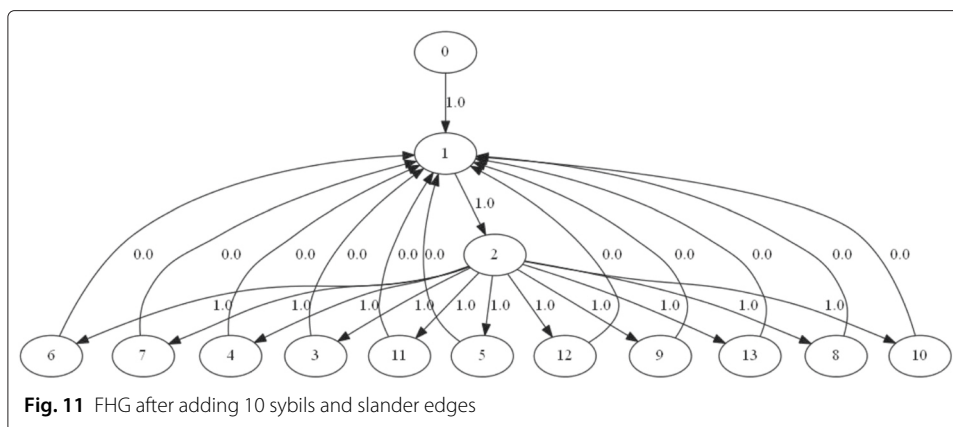
**Setup** We now turn to finding out how many malicious agents it takes to slander effectively. In a Sybil attack, a malicious agent introduces a number of *Sybil*s (i.e, accomplices or pseudonyms), whose purpose is to slander a victim in the system [12]. Suppose we are given the reputation graph shown in Fig. 10. Let agent 0 be a malicious agent that slanders agent 1. Assuming this agent has unlimited resources (e.g., an unlimited and

**Table 5** Rankings before and after slandering in retort (reputations in brackets)

	Agent 0	Agent 1	Agent 2	Agent 3
Eigentrust(FHG0)	3 (0.16)	2 (0.29)	1 (0.39)	3 (0.16)
Eigentrust(FHG1)	3 (0.16)	2 (0.29)	1 (0.39)	3 (0.16)
PeerTrust(FHG0)	3 (0.25)	1 (1.00)	1 (1.00)	4 (0.00)
PeerTrust(FHG1)	3 (0.11)	1 (0.44)	1 (0.44)	4 (0.00)



low-cost ability to create pseudonyms), it may introduce an unlimited number of Sybils to collectively slander agent 1. In such a scenario, it is useful to study how  $r(0, 1)$ , i.e. the reputation of Agent 1 from the point of view of Agent 0, 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 successful and we would measure the effectiveness of the attack based on the number of Sybils required. However, as explained in Section ‘Slandering attack, scenario 2’, one can also 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.

To measure Appleseed’s attack resistance with the graph in Fig. 10, we first verify whether the victim’s reputation is less than that of an agent in the 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 the FHG and Appleseed is run again (AppleSeed normally needs a reputation graph as input, but in this case since each FHG edge has a singleton feedback, the value of the feedback can be directly used as reputation). This process continues and Sybils are added for a certain number of iterations or until the attack succeeds (Fig. 11).

**Results** Appleseed resists this type of attack well. Table 6 summarizes the values of  $r(0, 1)$  and  $r(0, 2)$  in RG after adding 1, 10, 50 and 100 Sybils and slander edges to the FHG. We observe that  $r(0, 1) > r(0, 2)$ . This confirms the attack resistance property of Appleseed described in [3]. This is not the case of global trust algorithms such as EigenTrust, since the addition of sybils simply dilutes the reputation values of agents.

**Trust properties assessments**

In this section, trust algorithms are evaluated for their adherence to trust properties, namely: (a) weak transitivity and (b) the fact that trust is more easily lost than gained.

**Weak transitivity**

**Setup** Trustworthiness obtained via the transitive closure of trust paths should decrease as the length of the trust path increases [9, 25]. We subjected Appleseed and EigenTrust to simple tests to verify this basic rule. Even though the input and output graphs are of different types for these two algorithms, the same assertions can be applied to test the

**Table 6** Reputation scores by Appleseed

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

transitive rules, as they do not depend on the semantics of the weight of edges or whether the reputation scores are local or global.

**Results** Table 7 presents the test cases (the "Input FHG/RG" column), expected results (the "Output RG" column represents the transitive closure resulting from the graph transformation, along with question marks on the edges where the weak transitivity property is tested) and actual results. Both Applesed and EigenTrust passed our transitive tests.

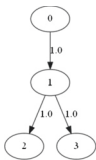
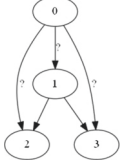
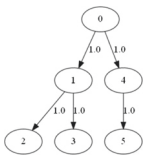
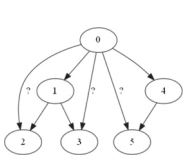

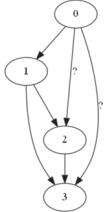
**Dynamic evolution of global trust**

According to Marsh [13], "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" [13]. That is, if cooperation does not ensue, then trust is lost at a rate higher that it was gained. Adherence to this property prevents white-washing attacks 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  $FHG(*, b)$ , which is the list of feedbacks on  $b$  by all agents in the system and  $r_i(b)$  be  $b$ 's global reputation score following  $i$ 'th feedback.

**Table 7** Transitivity tests and results

Exp	Input FHG/RG	Output RG	Expectation	Results	
				EigenTrust	Applesed
1			$r(0, 1) \geq r(0, 2)$ $r(0, 2) = r(0, 3)$	true	true
2			$r(0, 5) \geq r(0, 2)$ $r(0, 5) \geq r(0, 3)$	true	true
3			$r(0, 3) \geq r(0, 2)$	true	true

Assuming both reputation scores and feedback values are on the same scale and  $r(b)$  is a function of the feedback history  $FHG(*, b)$ , then trust must evolve according to conditions 5.1 and 5.2, where we define, for agent  $b$  and for any consecutive “instants”  $i - 1$  and  $i$ :

- $\delta f := f_i(*, b) - r_{i-1}(b)$  (the  $i$ th feedback is compared to the reputation at instant  $i-1$ )
- $\delta r := r_i(b) - r_{i-1}(b)$  where  $r_i(b)$  is obtained after feedback  $f_i(*, b)$
- $\frac{\delta r^+}{\delta f} := \frac{\delta r}{\delta f}$  if  $\frac{\delta r}{\delta f} \geq 0$ , and  $\frac{\delta r^-}{\delta f}$  otherwise

$$0 \leq \frac{\delta r^+}{\delta f} \leq 1 \tag{5.1}$$

$$\left| \frac{\delta r^+}{\delta f} \right| \leq \left| \frac{\delta r^-}{\delta f} \right| \tag{5.2}$$

That is:

1. Condition 5.1 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 5.2 verifies that the magnitude of the rate of change in reputation due to positive feedbacks is always less than the magnitude of the rate of change in reputation due to negative feedbacks.

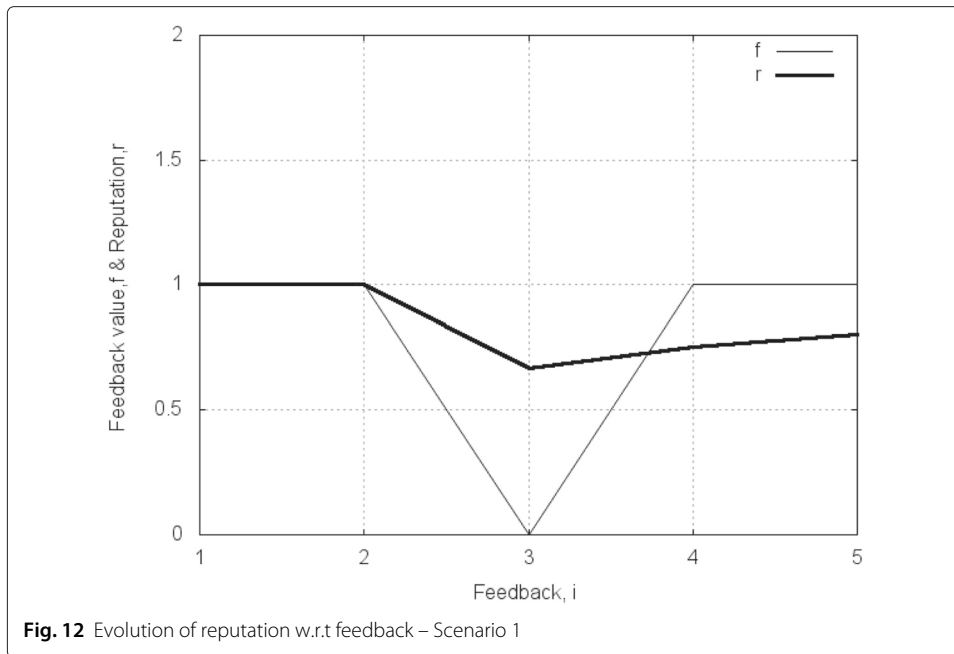
**Setup and results** We set up two scenarios to investigate the behaviour of PeerTrust. In the first scenario, there are only two agents  $a$  and  $b$ , and the feedback history  $FHG(a, b)$  is handcrafted such that  $a$  is typically satisfied with  $b$ , except once, where  $f(a, b)$  is set to 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.

*Scenario 1: evolution due to direct interactions only* Consider the feedback histories between two agents  $a$  and  $b$  in Table 8, which shows agent  $a$  was dissatisfied with  $b$  only once. One could argue that  $b$  mounted a white-washing attack in which it behaved honestly in order to cheat once in a while, or it is also possible that  $a$  is concealing a slandering attack, where it unfairly rated  $b$ . Even though information such as the intention of an agent thus cannot be extracted from the feedback history alone, we can use the above conditions to characterize a trust algorithm’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 8 and Fig. 12 show the evolution of  $r_0(*, b)$ . When  $f_3(a, b)$  (a negative feedback) occurred,  $r(*, b)$

**Table 8** Evolution due to direct interactions only—Scenario 1

$i$	Assessor	Assessee	$f$	$r$	$\delta f$	$\delta r$
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



decreased but it increased at a slower rate when  $f_4(a, b)$  (a positive feedback) occurred. Thus, PeerTrust has respected conditions 5.1 and 5.2.

In a separate experiment, we also noted that if there were two consecutive 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 misbehaving.

*Scenario 2: evolution due to direct and indirect interactions* In this experiment, we use the feedback history provided in Table 9. 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 in what follows.

If we compare the reputation values shown in Table 10, we note that  $r(a)$  changed from 1 to 0.5. However, this change did not affect  $r(b)$  after adding  $f_5$ . This is because of the normalization technique used by PeerTrust to calculate  $b$ 's reputation.

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.

**Table 9** Evolution due to direct and indirect interactions

$i$	Assessor	Assessee	$f$	$r$
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 10** Reputation output by PeerTrust

$i$	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

### 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 beginning 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 11 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 experiences and, depending on the algorithm, different weights may be given to direct versus indirect experience [26].

### Conclusions

We provided details of our prototype and evaluated trust algorithms for vulnerabilities to attacks and adherence to trust properties. We were able to show that our model could indeed accommodate various algorithms, and that the specification of experiments in terms of the model was straightforward and easy to express programmatically. The experiments that we ran allowed us to make the following observations:

- We were able to exhibit discrepancies between the outputs of EigenTrust and PeerTrust due to the way the initial normalized local trust values are calculated in EigenTrust; EigenTrust ignores the excess positive feedbacks once they outnumber negative feedbacks;
- There are tradeoffs between sensitivity to self-promotion and sensitivity to slandering. For example, an algorithm that is overly sensitive to slandering might take too long to incorporate bad feedback, and this allows bad behavior to go unnoticed for a while. In general, the difficulty is that one cannot always detect an attack just by looking at feedback history, since multiple interpretations are always possible.

**Table 11** Local reputation example

$i$	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

Clarification may come with the accumulation of feedback data, but multi-agent systems might not be able to survive to see that accumulation, unless a certain number of pre-trusted nodes can be provided to bootstrap the system.

- We were able to express and verify basic trust properties, such as weak transitivity and the fact that trust is more easily lost than gained. These should be building blocks upon which more complex properties could be expressed.

We have identified the following limitations and future work:

- Since the feedback history graph limits itself to agent-to-agent transaction ratings, recommender systems such as Credence [10] that use agent-to-object ratings cannot be included in the testbed.
- Trust systems that rely on different agent types (advisors, trusters, trustees, etc.) such as in [17] cannot be accommodated in our model.
- Our framework cannot accommodate trust algorithms, such as REGRET [27] or FIRE [28], that use inputs other than feedback history.
- Different agents might have different reputations in different contexts. Our testbed does not explicitly represent this context. One would have to create separate graphs for each context (thus making the assumption that the contexts are independent from one another), and this might be an oversimplification in cases where trust in one context partially influences trust in another context. We note however that in most of the trust algorithms we have classified the notion of context is not explicitly taken into account either.
- Agent behavior simulation: it would be desirable to use agent simulations to automatically generate feedback history graphs (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.

In addition to addressing the above limitations in our model, future work involves implementing more trust algorithms, 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.

## Endnotes

<sup>1</sup>In the remainder of the paper we will simply refer to the trust systems or models that have no specific name by the name of the first author of the paper we are citing, so [5] will be referred to as “Aberer”.

<sup>2</sup>in what follows we will use the graph name and the labelling functions interchangeably to reduce extraneous notations

<sup>3</sup>Note that in this figure, 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.

<sup>4</sup><http://code.google.com/p/repstestbed/>

## Competing interests

The authors declare that they have no competing interests.

**Authors' contributions**

PC did the implementation and ran the experiments, and participated in the formalization, the state of the art work and the write-up. BE participated in the formalization, the state of the art work and the write-up. Both authors read and approved the final manuscript.

**Received: 24 November 2014 Accepted: 3 July 2015**

**Published online: 07 September 2015**

**References**

- Kamvar SD, Schlosser MT, Garcia-Molina H (2003) The eigentrust algorithm for reputation management in p2p networks. In: WWW '03 Proceedings of the 12th International Conference on World Wide Web. ACM, Budapest, Hungary. pp 640–651. <http://doi.acm.org/10.1145/775152.775242>
- Xiong L, Liu L (2004) Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Trans Knowl Data Eng* 16(7):843–857
- Ziegler CN (2005) Chap. On propagating interpersonal trust in social networks. In: *Computing with Social Trust*. Springer, London. pp 133–166
- Zimmermann PR (1995) *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA
- Aberer K, Despotovic Z (2001) Managing trust in a peer-2-peer information system. In: CIKM '01 Proceedings of the Tenth International Conference on Information and Knowledge Management. ACM, Atlanta, GA. pp 310–317
- Teacy WTL, Patel J, Jennings N, Luck M (2006) Trivos: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents Multi-Agent Syst* 12(2):183–198. doi:10.1007/s10458-006-5952-x
- Jøsang A, Ismail R (2002) The beta reputation system. In: *Proceedings of the 15th Bled Electronic Commerce Conference*. pp 41–55
- Levien R (2009) Chap. Attack-resistant Trust Metrics. In: *Computing with Social Trust*. Springer, London. pp 121–132
- Golbeck JA (2005) *Computing and applying trust in web-based social networks*. PhD thesis, University of Maryland at College Park
- Walsh K, Sireg EG (2006) 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. Usenix, Berkeley, USA
- Josang A, Pope S (2005) Semantic constraints for trust transitivity. In: *Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modelling - Volume 43*. APCCM '05, Australian Computer Society, Inc., Darlinghurst, Australia, Australia. pp 59–68. <http://dl.acm.org/citation.cfm?id=1082276.1082284>
- Hoffman K, Zage D, Nita-Rotaru C (2009) A survey of attack and defense techniques for reputation systems. *ACM Comput Surv* 42:1–1131
- Marsh SP (1994) *Formalising trust as a computational concept*. PhD thesis, University of Stirling
- Abdul-Rahman A, Hailes S (2000) Supporting trust in virtual communities. In: HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6. IEEE Computer Society, Washington, DC, USA. p 6007
- Guha R (2004) Open rating systems. In: *Proceedings of the 1st Workshop on Friends of a Friend, Social Networking and the Semantic Web*. FOAF Galway, Galway, Ireland. [http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/open\\_rating\\_systems/wot.pdf](http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/open_rating_systems/wot.pdf). Accessed 31-Aug-2012
- Hazard CJ, Singh MP (September August 3) Macau: A basis for evaluating reputation systems. In: Rossi F (ed). *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, Beijing, China. AAAI Publications, Palo Alto, CA, USA. <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6854>
- Fullam K, Klos TB, Muller G, Sabater J, Schlosser A, Topol Z, Barber KS, Rosenschein JS, Vercouter L, Voss M (Unknown Month July 25) A specification of the agent reputation and trust (ART) testbed: experimentation and competition for trust in agent societies. In: Dignum F, Dignum V, Koenig S, Kraus S, Singh MP, Wooldridge M (eds). *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*. ACM, Utrecht, The Netherlands. pp 512–518. doi:10.1145/1082473.1082551, <http://doi.acm.org/10.1145/1082473.1082551>
- Kerr R, Cohen R (2009) Smart cheaters do prosper: defeating trust and reputation systems. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS '09. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC. pp 993–1000. <http://portal.acm.org/citation.cfm?id=1558109.1558151>
- Thirunarayan K, Anantharam P, Henson C, Sheth A (2014) Comparative trust management with applications: Bayesian approaches emphasis. *Future Generation Comput Syst* 31:182–199
- Yann Krupa JFH, Vercouter L (2009) Extending the Comparison Efficiency of the ART Testbed. In: Paolucci M (ed). *Proceedings of the First International Conference on Reputation: Theory and Technology - ICORE 09*, Gargonza, Italy
- Kerr R, Cohen R (2010) Treet: the trust and reputation experimentation and evaluation testbed. *Electron Commer Res* 10:271–290
- O'Hara K (2012) A general definition of trust. <http://eprints.soton.ac.uk/273193/>
- Ceolin D, Nottamkandath A, Fokkink W (2014) Efficient semi-automated assessment of annotations trustworthiness. *J Trust Manag* 1(1):1–31
- Myers JL, Well AD (2003) *Research Design and Statistical Analysis*. Lawrence Erlbaum Associates, New Jersey
- Josang A Trust and reputation systems. In: Aldini A, Gorrieri R (eds). *Foundations of Security Analysis and Design IV, FOSAD 2006/2007 Tutorial Lectures*. Lecture Notes in Computer Science. Springer, Berlin Vol. 4677. pp 209–245. [http://dx.doi.org/10.1007/978-3-540-74810-6\\_8](http://dx.doi.org/10.1007/978-3-540-74810-6_8)
- Guha R, Kumar R, Raghavan P, Tomkins A (2004) Propagation of trust and distrust. In: *Proceedings of the 13th International Conference on World Wide Web, WWW '04*. ACM, New York, NY, USA. pp 403–412. doi:10.1145/988672.988727, <http://doi.acm.org/10.1145/988672.988727>
- Sabater J, Sierra C (2001) Regret: A reputation model for gregarious societies. In: *Fourth Workshop on Deception Fraud and Trust in Agent Societies Vol. 70*
- Dong-Huynha T, Jennings N, Shadbolt N (2004) Fire: An integrated trust and reputation model for open multi-agent systems. In: *16th European Conference on Artificial Intelligence, Valencia, Spain*. IOS Press, Amsterdam. pp 18–22