COMPARS: Toward An Empirical Approach for Comparing the Resilience of Reputation Systems

Euijin Choo
North Carolina State University
echoo@ncsu.edu

Jianchun Jiang
Institute of Software Chinese Academy of Science
jianchun@nfs.iscas.ac.cn

Ting Yu
North Carolina State University
Qatar Computing Research Institute
tyu@ncsu.edu
tyu@qf.org.qa

ABSTRACT

Reputation is a primary mechanism for trust management in decentralized systems. Many reputation-based trust functions have been proposed in the literature. However, picking the right trust function for a given decentralized system is a non-trivial task. One has to consider and balance a variety of factors, including computation and communication costs, scalability and resilience to manipulations by attackers. Although the former two are relatively easy to evaluate, the evaluation of resilience of trust functions is challenging. Most existing work bases evaluation on static attack models, which is unrealistic as it fails to reflect the adaptive nature of adversaries (who are often real human users rather than simple computing agents).

In this paper, we highlight the importance of the modeling of adaptive attackers when evaluating reputation-based trust functions, and propose an adaptive framework—called COMPARS—for the evaluation of resilience of reputation systems. Given the complexity of reputation systems, it is often difficult, if not impossible, to exactly derive the optimal strategy of an attacker. Therefore, COMPARS takes a practical approach that attempts to capture the reasoning process of an attacker as it decides its next action in a reputation system. Specifically, given a trust function and an attack goal, COMPARS generates an attack tree to estimate the possible outcomes of an attacker's action sequences up to certain points in the future. Through attack trees, COMPARS simulates the optimal attack strategy for a specific reputation function f, which will be used to evaluate the resilience of f. By doing so, COMPARS allows one to conduct a fair and consistent comparison of different reputation functions.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems-Distributed applications; K.4.4 [**Computer and Society**]: Electronic Commerce—*Security*

Keywords

Reputation system; resilience; evaluation framework;trust functions

1. INTRODUCTION

Large-scale decentralized systems, such as peer-to-peer systems, online auction communities and ad hoc mobile networks, often involve transactions between strangers from different security domains with no pre-existing knowledge of each other. Although such systems offer great benefits in terms of service diversity, flexibility and scalability, they also give malicious parties the opportunity to cheat during transactions without being identified or punished [28, 31]. Inspired by social interactions between human beings, reputation mechanisms have emerged as a major technique for trust establishment in decentralized systems. In a reputation system, upon completion of a transaction, the involved parties issue feedback to evaluate one another's service or behavior during the transaction. Before a new transaction starts, one may first assess a party's trustworthiness based on the feedback on its previous transactions. This process can be viewed as the application of a *trust function*, which conceptually takes as input the feedback for a party's past transactions (or that of other parties, when necessary), and outputs a trust value to indicate its trustworthiness.

Many trust functions have been proposed in the literature (e.g., [1, 3, 12, 13, 29, 30, 33]). However, when building a decentralized system, it is non-trivial to pick the right trust function. One has to consider and balance a variety of factors, including computation and communication costs, scalability and resilience to manipulation. While many other factors are relatively straightforward to evaluate, the evaluation of resilience can be challenging. Resilience refers to how accurately a computed trust value reflects a party's true trustworthiness in the presence of malicious manipulation. Malicious manipulation includes selectively and strategically providing good or bad transactions, or issuing dishonest positive or negative feedback. We emphasize that the evaluation of resilience has to consider adaptive attackers who are aware of how a trust function works and adapt their behavior accordingly to maximize their gain in a system.

Unfortunately, though a large amount of work has been done on the design of trust functions, very few studies are devoted to systematic evaluation of their resilience [11, 14, 20]. Most existing efforts focus on performance evaluation [9, 28], while efforts on resilience evaluation are rather limited and are often built on the assumption that attackers are likely to behave within a fixed set of strategies that can be described by static models [3, 9, 14, 20, 26, 28]. This assumption is unrealistic, as it does not consider the adaptive nature of adversaries. Furthermore, evaluations based on fixed strategies do not offer a fair comparison of different trust functions. A particular trust function may be resilient to one type of attack yet vulnerable to another. Hence, comparing two trust functions using a fixed attack strategy may only offer a biased and incomplete view of their strength against manipulation. To be fair, one has to compare the worst cases of two trust functions. That is, it is necessary to compare their resilience against the *most effective attacks*, which are likely to be different for different trust functions.

In this paper, we propose an evaluation platform for the COMParison of Reputation Systems (COMPARS) that specifically takes into consideration the adaptiveness of adversaries. Essentially, an adversary tries to game a reputation system to achieve maximum profits, while normal users behave more or less consistently. Therefore, from an attacker's point of view, a reputation system sets up a single-player game. The trust function along with the behavior of normal users forms the environment and rules of the game (i.e., how the system will evolve when certain actions are taken), and the goal of the attacker is to carefully choose its actions to maximize certain profit measures (e.g., to meet a profit goal in the shortest time, or obtain the maximum profit in a given period of time).

Theoretically, given a trust function and a model of normal user behavior, there exists an optimal strategy for an attacker to game the system (i.e., the most effective attack). However, in practice, it is often hard to derive such optimal strategies. A system's state is determined by the behavior of a large number of users. Further, although normal users follow some behavior models more or less consistently, they are non-deterministic in nature (e.g., even if an honest user strives to provide good service, unsatisfactory transactions may happen from time to time due to uncontrollable factors such as network delays or interruption of delivery services, which have to be captured through probabilistic models). Therefore, it is very difficult, if not impossible, to purely rely on theoretical analysis to reason about the ultimate future state of a system and derive the optimal strategy directly.

We adopt an empirical approach that approximates the optimal strategy of an attacker. Specifically, COMPARS explores the future system states after an attacker takes up to k actions, similar to the idea of MiniMax [21]. Among these future states, COMPARS picks the one that is most beneficial to the attacker, and uses it to determine the attacker's next action. When doing so, we use a probabilistic reasoning method to deal with the non-deterministic nature of other users, which we will detail in sections 4 and 5.

Our main contributions are summarized as follows.

• We propose a general methodology to unbiasly and practically evaluate the resilience of trust functions by considering the adaptive nature of realistic attackers. The essential principle is to compare different trust functions based on their worst cases, i.e., to examine their vulnerability to manipulation when attackers adopt the optimal strategy specific for each trust function.

• We present the design of a highly configurable platform for trust function resilience evaluation. Through a set of well-defined interfaces, the platform allows us to plug in key modules of a reputation system, including honest user behavior models, initial system environment parameters, attack objectives and trust functions, so that we can study the resilience of a reputation system under various configurations. Once these basic modules are provided, the platform will automatically approximate the optimal strategies for an attacker to reflect the true resilience of the system (i.e., the worst case scenario).

• As a case study, we use COMPARS to compare several influential trust functions, including EigenTrust [13], PeerTrust [29] and TNA-SL [12], and observe their differences in terms of resilience to malicious manipulations.

The remainder of the paper is organized as follows. We start by discussing related works in Section 2. In Section 3 we present an abstract model of reputation systems, and introduce some basic concepts and notations used throughout this paper. Section 5 presents the proposed analysis framework, COMPARS. We provide a detailed description about four functional components of COMPARS in this section. Moreover, we discuss the evaluation criteria of COMPARS that considers the adaptive nature of attackers. Section 6 presents experimental results and analysis of existing reputation systems with the COMPARS framework. Finally, Section 7 concludes the paper.

2. RELATED WORK

A number of reputation systems have been proposed with the goals of ensuring trustworthy transactions between participants [1, 3, 5, 12, 13, 24, 29, 30, 33]. Most research efforts focus primarily on the design of trust functions. For example, some reputation systems utilize a few power (trustworthy) nodes to compute trust values [13, 33]. PeerTrust[29] employs similar ideas, but is built in a decentralized system. Some systems utilize additional information such as relationships between participants (e.g., transitive chains/paths) [12]. These reputation systems have shown their applicability in different application domains, and their effectiveness has been analyzed under a few scenarios in which malicious users attempt to exploit the systems. However, these systems are designed for specific application domains, and are evaluated only on those targeted domains, making general comparison difficult.

In order to facilitate a systematic analysis, several papers discuss design issues of reputation systems [9, 17, 26] and classify trust functions into a few categories [4, 16, 23, 27, 32]. Some of them further discuss attacks and defenses related to design issues [9]. Papers that deal with design issues can offer a clear view on different dimensions of reputation systems with a common criteria [9, 17, 23, 26, 27, 32]. Also, existing reputation systems can be theoretically analyzed with the proposed criteria[2]. A theoretic analysis, however, has the limitation that it does not take a practical perspective into account. Indeed, Jøsang *et al.* address that a reputation system, which is considered theoretically robust, can be nevertheless vulnerable in realistic environments [11]. Unlike theoretical approaches, we construct an evaluation framework that can generate quantitative assessment so that the different systems can be compared in a fair and consistent manner.

A few studies have attempted to address and evaluate the resilience of trust functions [2, 4, 7, 15, 28]. Jøsang *et al.* and Hoffman *et al.* discuss a set of properties to evaluate reputation systems [9, 11]. Although they mention resilience, they do not provide an analysis of resilience, instead focusing on algorithm analysis including how to compute trustworthiness. Zunping *et al.*[4] classify trust-based recommender systems into three categories and pick a representative system in each category to analyze their resilience. Through experiments, the authors prove vulnerabilities in trust-based recommender systems. However, their evaluations are limited to a few specific recommender models and specific attacks.

Fullam *et al.*[7] propose a testbed to evaluate reputation systems, called *ART*. Unfortunately, ART can only be used in a few specific applications and it only allows a small number of participants. Consequently, ART can deal with only simple and static user/attack behavior scenarios, which is unrealistic. For example, every participant is assumed to behave in exactly the same way, attackers do not change their behavior, and no participant can enter/leave during evaluation. Instead, we employ a probabilistic reasoning method to handle a large number of users' behavior.

Kerr*et al.*[15] propose a more general platform, TREET, to analyze reputation systems. Although TREET is more flexible than ART, supporting both centralized and decentralized systems, it can only handle marketplace scenarios and specific attacks in the marketplace. Our COMPARS framework, on the other hand, is general and domain-independent.

West *et al.*[28] and Irissappane *et al.*[10] propose evaluation frameworks based on empirical approaches. West *et al.* define possible user/attacker behavior models and implements a simulator with a static trace. Irissappane *et al.* simulate reputation systems with static user behavior models and fixed attack models. Static behavior models and traces, however, make systems vulnerable by nature, since attackers may be well-aware of how a trust function works and intentionally change their behavior to exploit a reputation system. Further, both [28] and [10] do not take users' non-deterministic nature into consideration. Hence, it is hard to handle real scenarios where each user may have their own behavioral pattern in [28] and [10]. In contrast, we employ a probabilistic user behavior model to capture the non-deterministic nature of other users.

Considerable research on evaluation frameworks lies in the classification of reputation systems and attacks/defenses in reputation systems [2, 9, 10, 18, 26, 28]. These works try to classify reputation systems considering their resilience against a few static attack models [2, 4, 6, 10, 11, 14, 24, 25, 28]. In contrast, we believe an evaluation based on static attack models is not sufficient to reflect the true resilience of a reputation system. Instead, we propose an evaluation framework that captures the adaptive nature of attackers who can exploit the properties of specific trust functions and behave accordingly to maximize their profits.

3. REPUTATION SYSTEM

Reputation systems help users estimate the trustworthiness of other parties in a decentralized system. By decentralized, we mean that entities are autonomous; there is no single centralized authority that asserts the trustworthiness of entities, or makes decisions on the appropriate actions of an entitie. This concept is orthogonal to the underlying exchange structures of a system including centralized (e.g., e-commerce systems) or decentralized (e.g., a peer-to-peer file sharing systems). We assume that entities in a decentralized system interact with each other through transactions. Transactions are not limited to monetary interactions; they also include activities such as retrieving information from a website, downloading files from a peer, and etc. We assume that a transaction is unidirectional, i.e., given a transaction, there is a clear distinction between a service provider and a service consumer.

In general, reputation systems share a common structure[9], typically modeled as a 5-tuple (C, P, R, F, A), where C is a set of service consumers, P is a set of service providers, R is a set of feedbacks, F is a trust function, and A is a set of actions. We describe the five components in detail below.

Service consumers. A service consumer is an entity who seeks services from a decentralized system, such as a buyer in an e-commerce market and a downloader in a file-sharing system. Each consumer is associated with a profile, which is a set of properties that are relevant to reputation management. For example, a profile may include the time a consumer joins the system and demographic information. A service consumer may have a set of services that it would like to get from the system at a certain time, but such information is not likely to be publicly known. Hence, we do not explicitly model it as a part of a consumer's profile. Instead, it is modeled by a consumer behavior model, which is essential to reason about the evolution of a reputation system.

Service providers. A service provider is an entity who offers a set of services that may be requested by consumers (e.g. sellers in an e-commerce market and uploaders in file-sharing applications). Usually, the set of services offered by a provider is public (e.g., in ebay we can see all the items a seller is selling, but we do not know what items a buyer may need). Therefore, the services offered by a provider is a part of its profile. Its profile may also have other similar properties to that of consumers. Note that the set of consumers and providers may not be disjoint; an entity may be a consumer in one transaction and a provider in another.

Feedbacks. A feedback γ for a transaction takes the form (c, p, i, r, t), meaning that consumer c received service i from provider p at time t, and its rating is r. We leave the format of the rating opaque as it is application specific. In many systems, it is a single numerical/categorical value (e.g., 0 to 5 stars, or from poor to excellent). In others, it may instead be a vector that reflects a transaction's quality from multiple aspects, e.g., price, product quality, and responsiveness of customer service.

Trust function. If an entity a wants to evaluate the trustworthiness of another entity b, then we call a and b the source and target of the trust evaluation, respectively. A trust function takes a source, a target and a set of feedbacks, and returns the target entity's trust score. Similar to feedbacks, the format of a trust score is also opaque, and depends on the specific trust function. Most trust functions return a single numerical value as a trust score, while some others advocate returning a vector of numerical values [32], each corresponding to an aspect of the target.

Most trust functions in the literature are subjective (i.e., from the point of view of different sources, the same target may have different trust scores). Some other functions are objective (or global), meaning the trust score of an entity does not depend on the source. Nevertheless, the above modeling of trust functions is general enough to capture both types of functions.

Actions. An entity can take many different actions in a reputation system. For example, a provider may list a set of services that are available to others; a consumer may choose to start a transaction with a provider; a consumer may post a positive or negative feedback regarding a transaction; a provider may provide good or poor transactions intentionally or unintentionally, etc.

The set of actions essentially defines the capabilities with which an attacker can manipulate a system, and is highly system-specific. For instance, many e-commerce systems have mechanisms to ensure that an entity cannot post a feedback unless it was indeed a consumer in a transaction. Other systems, such as online rating systems, cannot verify whether an entity has direct experience with a service before rating it. As another example, some systems require a user to present some real-world credentials (e.g., credit cards) before creating an account, to circumvent Sybil attacks[3]. Many other systems however allow free entry. Therefore, attackers may create multiple accounts and launch coordinated manipulation through these accounts.

As discussed earlier, an adaptive attacker does not stick with a fixed strategy to game a reputation system. Instead, it would evaluate the possible consequences of actions available at any given time, and decide which action is the best to achieve its goal. Note that the consequence considered might not be just the immediate ones. Instead, it is often desirable to consider an action's long term impact to identify the best action at present. For example, cheating in a transaction at present might give the attacker an immediate payoff, but it may dramatically hurt its trust score such that consumers are much less likely to come to the attacker for service in the future, which is not desirable for achieving its goal (e.g., get a certain amount of profits in the shortest time). Thus, in terms of the overall long-term payoff, it may not be the best action to cheat immediately at present. To model this reasoning process, we introduce the concept of the state of a reputation system and its transition.

The *state* of a reputation system is a 3-tuple (C, P, R), consisting of the set of consumers, the set of providers, and the set of feedbacks in the system. When an action occurs in a system, its state will change accordingly. For instance, when a new user joins the system, C is updated; when a provider lists a new service, its profile is updated; R is updated when a new feedback is issued. Let S_t be the system state at time t_0. After an action a happens at time t_1, the system state *transitions* to S_{t_1}, denoted $S_t \xrightarrow{(a,t_1)} S_{t_1}$. Given a sequence of actions $X = (a_1, t_1), \ldots, (a_i, t_i)$, we denote the transition as $S_{t_0} \xrightarrow{X} S_{t_e}$.

4. OPTIMAL ATTACK STRATEGIES

The attackers' goal is to manipulate a reputation system to gain advantages (e.g., money payoff, free downloading, or spreading of malware). To do so, attackers often change their behavior according to the change of system states. Hence, it is important to model attackers' adaptive behavior to evaluate the resilience of trust functions. Attackers essentially try to find and perform the *most effective attack* for a specific trust function. Clearly, the most effective attack will be different for different trust functions; and attackers should have *an optimal strategy* to perform the most effective attack for the trust function. We model a reputation system as a game, in which players are either normal users or attackers. In Section 4.1, we first explain our model of a game in reputation system, and we describe how to find the optimal strategy in sections 4.2 and 4.3

4.1 Games in Reputation Systems

Generally, there are multiple players in a game whose type can be normal users and attackers in a reputation system. Depending on the setting, a player may join or leave during the game. Each player has multiple choices of actions in each turn and a system state transitions to the next system state depending on each player's action.

For a reputation system to reflect a user's true trustworthiness, users are expected to behave in a way that the system wants them to (i.e., to behave in a predictable manner). For example, a reputation system requires consumers to give a high (low) rating to a good (bad) service[31]. However, each user may behave in its own way, not the same way as other users. For instance, one user may tend to give relatively low ratings habitually, even when it is satisfied with a service; an honest user strives to provide good service, but unsatisfactory transactions may still occur due to unreliable delivery from third-party. Hence, the behavior of normal users is predictable yet non-deterministic, and each normal user has its own behavior model.

An attacker, on the other hand, tries to game a reputation system to achieve its goal. Therefore, from an attacker's point of view, a reputation system sets up a single-player game. The trust function along with the behavior models of normal users forms the environment and rules of the game. The goal of an attacker is to carefully choose its actions to maximize its profit. To choose an action in each turn, an attacker considers the consequence of the action. That is, the attacker needs not only to examine a current system state, but also to estimate a future system state. For example, an attacker will choose a certain action, if it estimates the action will bring a profit; an attacker will not choose a certain action, if it estimates the action will punish itself.

Theoretically, given a trust function and models of normal user behavior, there exists an optimal strategy for an attacker to achieve its goal. Unlike classical games such as chess, however, there are a large number of users whose behavior is non-deterministic in a reputation system. Also, the design of reputation systems is more complicated. Therefore, purely using a theoretical analysis to pick the optimal attack strategy can be very difficult. We thus employ an empirical approach to explore future system states after an attacker takes a sequence of actions and approximate the optimal strategy of the attacker. Similar to the idea of MiniMax[21], we represent possible system states as a tree, called *an attack tree*, but only in an attacker's point of view. In the following subsections, we delineate how we generate an attack tree.

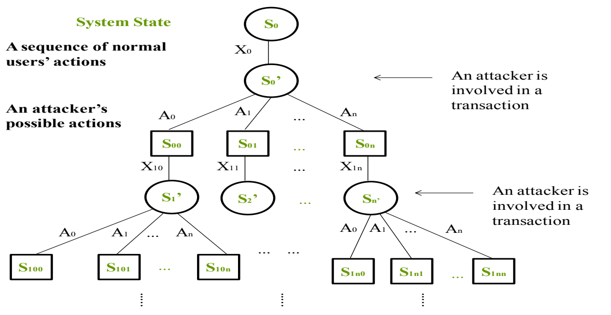


Figure 1: An attack tree

4.2 Attack Tree

A conceptual view of an attack tree is shown in Fig.1. The formal definition of an attack tree is given as follows.

Definition 1. *An attack tree is a rooted tree given by <S, X, A>, where*

• *S is the set of nodes, each of which ($S_i = (C_i, P_i, R_i)$) represents a system state at a certain time point.*

• *X is the set of edges, each of which ($X_i = \{(a_1, t_1), \ldots, (a_m, t_m)\}$) represents a sequence of normal users' actions.*

• *A is the set of edges, each of which ($A_i = (a_i, t_i)$) represents one of an attacker's possible actions.*

As discussed in Section 3, a system state transitions into the next system state, whenever an action occurs. The action may be performed by attackers or normal users. The attack tree, however, is employed to derive the attackers' optimal strategy, while estimating future system states from an attacker's point of view. Accordingly, we do not represent every possible system state that evolves depending on a normal user's action as a node. Instead, we represent system states as nodes only when an attacker is involved in the transactions. In order to differentiate a system state right after an attacker's single action A_i from a system state after a sequence of normal users' actions X_i, we represent the former as a square node and the latter as a round node. Each round node (except the root node) has n square child nodes, each of which corresponds to a system state after an attacker's action A_i occurs. Each square node and the root node have one round child node corresponding to a system state after a sequence of normal users' actions X_i occurs.

We assume that an attacker is involved in a transaction at time $t_{j'} (j = 0, \ldots, n)$. And $S_{j'} (j = 0, \ldots, n)$ are the system states at time $t_{j'} (j = 0, \ldots, n)$. As shown in Fig.1, when an attacker tries to estimate a system state, a sequence of normal users' actions is generated first by randomly choosing users and their actions according to the users' behavior models. Each action results in a transition into a new system state. A node S_0 *transitions* to a node S_0' after a sequence of normal users' actions X_0 happens, as shown in Fig.1. At time $t_{0'}$ when an attacker is involved in a transaction, the attacker has a set of choices A_i's, i.e. the attacker's possible actions. After taking one of the actions, an attacker will estimate its next system state. Depending on which ac-

```
AttackBehavior()
1: $S \leftarrow S_0$
2: $r \leftarrow 0$
3: $Gen\_AttackTree(S, r)$

Add_Node(Parent $S_p$, Child $S_c$)
1: Add $S_c$ as one of $S_p$'s children into an attack tree.

Gen_AttackTree(System state $S_p$, $r$)
1: $r \leftarrow r + 1$
2: $S_c \leftarrow Normal\_Sequence(S_p)$
3: $Add\_Node(S_p, S_c)$
4: for $k = 0$ to $N_A$ do
5:     // Do an action $a_k$
6:     $S_k \leftarrow Get\_SystemState(S_c, a_k)$
7:     $Add\_Node(S_c, S_k)$
8:     if $r < M_T$ then
9:         $Gen\_AttackTree(S_k, r)$
10:    end if
11: end for

Get_SystemState(System state $S_p$, Action $a$)
1: $S_p \xrightarrow{a} S_c$
2: $S_p$ transitions to $S_c$ after $a$ happens.
3: return $S_c$;

Normal_Sequence($S_p$)
1: for $k$=0 to $L_S$ do
2:     Randomly pick a user and its action, $\omega$.
3:     $S_p \leftarrow Get\_SystemState(S_p, \omega)$
4: end for
5: return $S_p$;
```

S_0: an initial system state
S_i: System states at time i
M_T: Maximum height of an attack tree.
r: A counter for the height of an attack tree
A: The set of attackers' possible actions to achieve their goals.
a_k: An attacker's one possible action at a certain time point. ($\in A$)
N_A: The number of attackers' possible actions at a certain time point.
Ω: The set of normal users' possible actions.
ω: One of normal users' actions. ($\in \Omega$)
L_S: Chosen length of one sequence of normal users' actions.

Figure 2: The pseudocode for generation of an attack tree

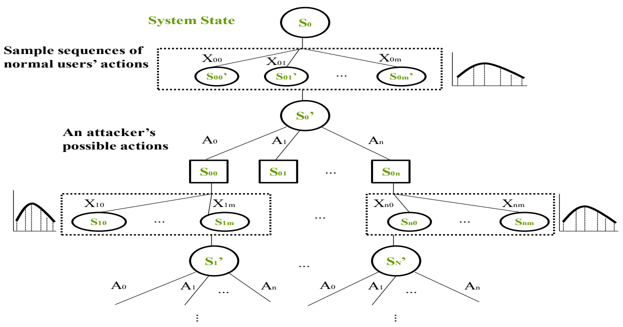


Figure 3: An attack tree with sample sequences

tion is taken by the attacker, a node S_0' *transitions* to its n child nodes, $S_{0i}(i = 1, \dots, n)$.

An attacker repeats this *reasoning process*, until the number of the attacker's transactions exceeds a certain number, which is defined by the attacker. We call the number of the attacker's transactions in an attack tree as the *height* of the attack tree, which is the same as the depth of a tree only considering round nodes. For example, the height of the attack tree in Fig.1 is two. Ideally, if we can explore all possible system states, we can get the most accurate estimate of an attacker's optimal strategy. In practice, it would be computationally infeasible to explore the whole space of state transition. We thus limit the maximum height of an attack tree so that attackers can predict future system states with a reasonable computation. Fig. 2 shows the pseudocode to generate an attack tree.

4.3 Attack Tree with Sample Sequences

A system state is determined by the behavior of a large number of users, which is likely to be probabilistic, as discussed earlier. Since the generation of an attack tree is an estimation process of the attackers, only one sequence of normal users' actions would not be representative to reflect a large number of users' probabilistic behavior. We thus sample the users' actions to make better estimation of system states. That is, we generate sample sequences of normal users' actions, instead of one single sequence of normal users' actions. We now describe how we generate an attack tree with sample sequences of normal users' actions.

Fig.3 depicts an example of an attack tree with sample sequences of normal users' actions. Similar to Fig.1, we assume that an attacker is involved in a transaction at time $t_{j'}(j = 0, \dots, n)$ and $S_{j'}(j = 0, \dots, n)$ are system states at time $t_{j'}(j = 0, \dots, n)$. Whenever an attacker tries to estimate a system state, m sample sequences of normal users' actions $X_{jk}(k = 0, \dots, m)$ are generated first. Each sequence X_{jk} is generated by randomly choosing users and their actions according to the users' behavior models. Although each action results in a transition into the next system state, we only represent system states after the last action in the sequence X_{jk} occurs as nodes in an attack tree. Hence, a node S_0 *transitions* to nodes $S_{0k}'(i = 0, \dots, m)$ after sequences of normal users' actions $X_{0k}(k = 0, \dots, m)$ happen, as shown in Fig.3.

To estimate a system state at $t_{j'}(j = 0, \dots, n)$, we need to analyze the distribution of system states $S_{jk}'(k = 0, \dots, m)$. Then, the attacker picks a *representative system state*, S_j', among $S_{jk}'(k = 0, \dots, m)$'s for the time, $t_{j'}(j = 0, \dots, n)$. The representative system state can be defined by the attacker. For example, let $v_{jk}'(k = 0, \dots, m)$ be the trust score of an attacker at each system state, $S_{jk}'(i = 0, \dots, m)$. Then, the attacker may choose a system state $S_{j,avg}'$, at which an attacker's trust score $v_{j,avg}'$ is the average of trust scores $v_{jk}'(i = 0, \dots, m)$, as a representative system state for the time, $t_{j'}(j = 0, \dots, n)$.

Similar to Fig.1, the attacker has a set of choices A_i's, i.e. attackers' possible actions, at time $t_{j'}$ when an attacker is involved in a transaction. After taking one of the actions, an attacker will estimate its next system state. Depending on which action is taken by the attacker, a node S_j' *transitions* to its n child nodes, $S_{ji}(j = 1, \dots, n)$. An attacker repeats this reasoning process, until the number of the attacker's transactions exceeds a certain number, which is defined by the attacker.

5. COMPARS: A FRAMEWORK FOR COMPARISON OF REPUTATION SYSTEMS

The goal of the proposed approach is to evaluate trust functions in the presence of adaptive attack behavior. In Section 5.1, we first

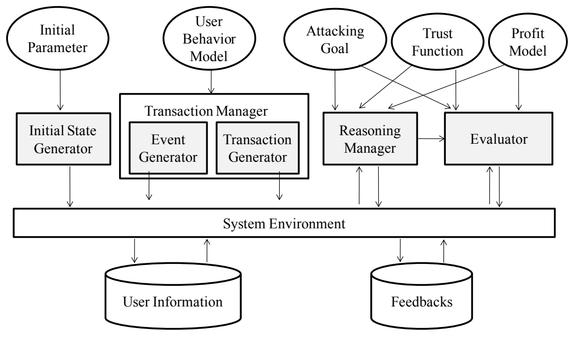


Figure 4: The architecture of COMPARS

give a brief overview of the proposed framework, called *COMPARS* (COMPArison of Reputation Systems). In sections 5.2, 5.3, 5.4, and 5.5, we discuss four major functional components of COMPARS and the evaluation criteria of COMPARS.

5.1 Overview of COMPARS

COMPARS simulates the evolution of a reputation system. The framework is built with basic components common to reputation systems so that any reputation system can be easily integrated into COMPARS and so that different trust functions can be evaluated with COMPARS. Fig.4 shows the architecture of COMPARS, which consists of four functional components: *initial state generator, transaction manager, reasoning manager*, and *evaluator*.

As noted before, strategic attackers often change their behavior, depending on specific properties of trust functions and of normal users. In order to reflect an attacker's adaptive strategy, COMPARS considers a reputation system from the attacker's point of view. Essentially, given the initial system state, the goal of an attacker is to carefully choose its behavior to maximize its profits. COMPARS thus derives an attacker's optimal strategy to achieve its goal. First, the *initial state generator* generates an initial system state, which is defined by basic user information (e.g., the list of consumers, providers, and items each user has). Given the initial system state, *the transaction manager* controls who will be involved in each transaction. In general, normal users will not change their behavior much. Along with this observation, *the transaction manager* takes user behavior models as input. *The transaction manager* consists of *an event generator* and *a transaction generator*. *The event generator* controls who will be a consumer, *c*, based on a current system state and user behavior models. Depending on the user behavior models, *the transaction generator* decides who will be chosen as a provider, *p* by a consumer, *c*. In Section 5.3, we discuss how we define behavior models for normal users in detail.

Given a trust function and a system state, attackers attempt to game a reputation system to achieve their goals. Different from normal users, attackers are adaptive so that they are able to choose the optimal strategy under a specific system state. For attackers to derive their optimal strategy, COMPARS explores the future system states after an attacker takes up to k actions. COMPARS represents possible system states with different attacking actions as an attack tree, as illustrated in Section 4.

The *reasoning manager* handles the reasoning process of attackers, generating attack trees to reason about the attackers' future actions. By monitoring a generated tree, COMPARS picks the most beneficial action to the attacker and uses it to determine the next action that the attacker should take. Given the optimal strategy, the *evaluator* carries out the optimal strategy and evaluates the resilience of the reputation systems.

In the following sections, we describe each component in detail.

c: A consumer who wants to get a service
S_t, S_{t_1}, S_{t_2}: System states at given time t, t_1, and t_2, respectively
Σ_c: The set of consumers' strategies to pick service providers
σ_c: c's strategy to pick service providers ($\in \Sigma_c$)
γ : A feedback from a single transaction

ConsumerBehavior()

```
1: Pick a service provider p who meets the requirements of $\sigma_c$.
2: Do a transaction with p.
3: $S_t \xrightarrow{c\ gets\ a\ service} S_{t_1}$
4: Issue a feedback, $\gamma$.
5: $S_{t_1} \xrightarrow{c\ issues\ a\ feedback\ \gamma} S_{t_2}$
```

Figure 5: A consumer's behavior and the evolution of system states

5.2 The Initial State Generator

An initial system state is the system state before an attacker's action. Note that COMPARS simulates the evolution of a reputation system in an attacker's point of view. Hence, the initial state does not mean no transactions ever happen in the system. An attacker may join in the middle of a system; or, normal users can be compromised by attackers and start to behave maliciously. To generate the initial system state, the *initial state generator* takes initial parameters as input, including basic information (e.g., the list of consumers, providers, and services offered by providers). With the given parameters, the *initial state generator* generates the initial system state $S_0 = (C, P, R)$, where C is a set of service consumers, P is a set of service providers, and R is a set of feedbacks.

5.3 The Transaction Manager

Given the initial system state, the *transaction manager* controls who will be involved in each transaction. As mentioned earlier, normal users in reputation systems are likely to behave consistently. The *transaction manager* takes user behavior models as input. Our abstract model of normal user behavior is as follows. Note that the abstract model is not fixed, but is flexible and able to accommodate different user behavior model.

A. A Consumer Behavior Model: A consumer seeks services from a decentralized system. While doing so, a consumer needs to choose a set of services it would like to get as well as a provider from whom it would like to get the services. Given a trust function, a consumer's *strategy* for choosing a provider may vary depending on a current system state. For example, one consumer may choose a provider whose trust value is over certain threshold, whereas another may only choose the top-ranked providers in the system. Based on its own strategy at a given system state, consumer c chooses provider p and starts a transaction with p. After c gets service i, c issues feedback γ. Fig.5 describes how a system state evolves depending on a consumer's action.

B. A Provider Behavior Model: A provider offers a set of services, which is usually publicly known. Therefore, provider p's set of services should be a part of its profile, as discussed previously. However, even if a provider offers the same services at different time, the quality of transactions may vary due to uncontrollable factors such as network delays or interruption of delivery services. Also, a provider may want to change the quality of transactions for the same services depending on the trust function and consumers' strategies. For example, uploaders in file-sharing applications may publish normal quality files rather than high quality files to save their resources. As another example, sellers in e-commerce markets who do not have a good transaction history may try to post

p: A provider who wants to provide a service
c: A consumer who wants to get a service
S_t, S_{t_1}, S_{t_2}: System states at given time t, t_1, and t_2
Σ_p: The set of providers' strategies
σ_p: p's strategy to provide a service ($\in \Sigma_p$)

```
ProviderBehavior()
1: while $p$ is not chosen by any consumer do
2:     $p$ checks a current system state.
3:     $p$ picks a strategy $\sigma_p$.
4:     $p$ lists a set of services, which meet requirements of $\sigma_p$.
5:     $S_t \xrightarrow{p\ lists\ a\ set\ of\ services} S_{t_1}$
6: end while
7: if p is chosen by a consumer c then
8:     $S_{t_1} \xrightarrow{p\ offers\ a\ service\ to\ c} S_{t_2}$
9: end if
```

Figure 6: A provider's behavior and the evolution of system states

only best items to repair their reputations. Note that this is different from the attackers' behavior, in which an attacker intentionally provides good or bad services to maximize its profits. To capture such factors, we use a provider behavior model in addition to a provider's profile. Fig. 6 shows a provider's behavior and how a system state evolves depending on a provider's action.

Provider p lists a set of services and waits until it is chosen by a consumer. If p is not chosen within a certain time, p checks the current system state and changes its *strategy* to offer the services. For instance, p may offer the same quality of services at a lower price or p may offer a better quality of services at the same price.

Given the initial system state and user behavior models, the *transaction manager* controls who will be involved in each transaction. The transaction manager consists of the *event generator* and the *transaction generator.* Based on the current system state and user behavior model, the event generator picks a consumer who will be involved in each transaction. For example, the event generator may pick a user as a consumer only when its trust value is over a certain threshold. Once the event generator picks consumer c, the transaction generator selects provider p depending on c's strategy to pick a service provider. The quality of each transaction is decided depending on a specific system configuration. In some reputation systems, one provider may explicitly mention (advertise) a service quality; or, a reputation system may have its own measure to judge a service quality [28].

5.4 The Reasoning Manager

As described in Section 4, sophisticated attackers often change their behavior intentionally based on the trust functions. If a framework evaluates trust functions based on a set of pre-defined strategies, this always leaves chances for attackers to exploit trust functions by using different strategies that are not adopted in the evaluation. COMPARS thus employs an empirical approach to model attackers' adaptive behavior, which is delineated in Section 4.

A reasoning process begins when an attacker is chosen for a transaction, generated by the transaction manager. An attacker can be a consumer or a provider. The *reasoning manager* takes an attacker's goal as input. The attacking goal can be defined with a few parameters, which may include, but are not limited to a trust score, a profit, and a time period. For example, an attacker may want to achieve its profit goal within a certain time period, while maintaining a trust score above a certain value; or, an attacker may want to demote a competing provider's trust score so as to prevent the provider from being chosen by consumers. To compute profits from a given action, the reasoning manager takes *a profit model* as input. A profit model is a method to compute profits resulting from a single action at a specific system state.

Depending on a specific reputation system, the attacker may have different choices of actions at each system state to achieve its goal. For example, if allowed by a system, an attacker may also create a new account, so that it can control multiple accounts in one reputation system. Accordingly, the reasoning manager takes into account the attacker's capabilities in a given reputation system and handles the attacker's reasoning process in the system, so that COMPARS can accurately predict the optimal action at present.

The reasoning process continues until the maximum reasoning step (i.e., the maximum height of an attack tree) is reached. More reasoning will lead to better estimation. The amount of reasoning performed, however, will impact computation overhead. Therefore, we limit the maximum number of reasoning steps so that COMPARS can predict future system states with a reasonable amount of computation.

5.5 The Evaluator

A good trust function will restrict the chances for an attacker to exploit a system. In other words, if an attacker can achieve its goal early, it is expected that the trust function is vulnerable to manipulation. Hence, the *evaluator* observes how many transactions are required for an attacker to achieve its goal with its optimal strategy.

6. EXPERIMENTAL RESULTS AND ANALYSIS

This section describes an analysis of reputation systems using COMPARS. Many reputation systems have been developed in different application domains[32]. To show the validity of COMPARS, three influential reputation systems—EigenTrust [13], PeerTrust [29], and TNA-SL [12]—have been integrated into the COMPARS framework. Note that COMPARS is general and domain-independent, so that there are many ways to materialize COMPARS depending on which reputation system will be evaluated. For example, different user behavior models and profit models can be used to evaluate different reputation systems. Here, we provide a few case studies with the three reputation systems in an eBay-like e-commerce system; and show how COMPARS can be used to observe the resilience of different reputation systems with adaptive attackers.

We implemented EigenTrust and PeerTrust ourselves, and modified and adjusted the TNA-SL code by Andrew G. West *et al* [28]. As discussed in Section 2, EigenTrust is designed for peer to peer file sharing applications with the assumption that there are peers who always behave in an honest way and can thus be pre-trusted. PeerTrust is implemented in a decentralized P2P environment without any pre-trusted users. TNA-SL utilizes a theoretical approach with a greater emphasis on prior direct interaction. Details of the three reputation systems can be found in their respective papers [12, 13, 29]. Considering the fundamental differences between these three systems, we believe other reputation systems can be integrated easily into COMPARS and evaluated as well. We first present the parameters used for our experiments in Section 6.1 and we present experimental results in Section 6.2.

6.1 Materialization

Table.1 summarizes the notation for the parameters used in our experiments. Except for experiments where we needed to change some parameter values, we used the default values listed in the table.

Normal user behavior model: As mentioned earlier, the behavior of normal users in e-commerce markets (e.g., eBay) is typi-

Parameter	Description	Default
N_U	the number of users in our network	100
N_S	the number of samples for reasoning	15
H_i	feedback reliability rate of a consumer i	1.0
Q_i	service quality rate of a provider i	1.0
σ_c	consumers' strategy to choose a provider	OVER TRUST
G	an attacking goal (the amount of profit that an attacker wants to get)	60
M_T	the maximum height of an attack tree for reasoning	1
T_{avg}	the average of trust values	*
α	a range value to calculate profits	0.01
δ_T	threshold to be selected as a provider with "OVER TRUST" strategy	T_{avg}
δ_R	threshold to be selected as a provider with "OVER RANK" strategy	$N_U \times \frac{50}{100}$

Table 1: List of parameters for experiments

cally predictable, but non-deterministic. Such nature can be captured by probabilistic models [28, 31]. We thus employed two parameters-*feedback reliability rate* H_i and *service quality rate* Q_i, each of which was defined for consumers and providers, respectively. Feedback reliability rate H_i is the probability that consumer i gives feedbacks consistent with true service quality; service quality rate Q_i is the probability that provider i offers good services.

The value for H_i ranges from 0.0 to 1.0, where consumers whose H_i is close to 0.0 are untrustworthy and those whose H_i is close to 1.0 are trustworthy. Although normal consumers' H_i should usually be close to 1.0, 0.0 does not necessarily mean the consumer is an attacker. This is because it is possible that one consumer may keep offering bad feedbacks for good services unintentionally, because of uncontrollable factors (e.g., a consumer is under a bad network condition or a bad delivery service); or, a consumer may accidentally give good feedbacks for bad services. We used a default value of 1.0 for H_i.

The value for Q_i also ranges from 0.0 to 1.0, where providers whose Q_i is close to 0.0 are untrustworthy and those whose Q_i is close to 1.0 are trustworthy. Similar to H_i, 0.0 does not necessarily mean the provider is an attacker, because uncontrollable factors (e.g., a provider is under a bad network condition or a bad delivery service) may affect the provider's service quality. We used a default value of 1.0 for Q_i.

Each trust function in three reputation systems (i.e., EigenTrust, PeerTrust, TNA-SL) returns a single trust score for each user reflecting their trustworthiness. In many reputation systems, the trust score is often represented as either a trust value that ranges from 0.0 to 1.0 (i.e., a trust value-based approach) or a rank among users (i.e., a rank-based approach) [9, 32]. We thus defined two strategies for a consumer to choose a provider. One is "OVER TRUST" with which a consumer will choose a provider whose trust value is over a certain threshold. A consumer can choose a threshold δ_T to pick a provider when the "OVER TRUST" is selected. Another is "OVER RANK" with which a consumer will choose a provider who is in the set of top-ranked providers. A consumer can choose a threshold δ_R to define top-ranked providers so that a user will be chosen as a provider if its rank is higher than δ_R. If a consumer does not have any strategy, a provider will be chosen randomly. Most reputation systems employ a trust value-based approach. We thus assumed that users choose "OVER TRUST", except when we compare trust value-based with rank-based approaches.

Although each user may behave in its own way, we assumed in this experiment that every user follows the same behavior model

Reputation	Profit (Cheating)	Profit (No cheating)
$> T_{avg}$ + 5 *α	20	10
$> T_{avg}$ + 4 *α	18	9
$> T_{avg}$ + 3 *α	16	8
$> T_{avg}$ + 2 *α	14	7
$> T_{avg}$ + α	12	6
$> T_{avg}$	10	5
$> T_{avg}$ - α	8	4
$> T_{avg}$ - 2 * α	6	3
$> T_{avg}$ - 3 *α	4	2
$> T_{avg}$ - 4 *α	2	1

Table 2: Profit setting

for simplicity. That is, every user was assumed to share the same feedback reliability rate, the same service quality rate, and the same strategy to choose a provider. Also, normal users are often expected to behave consistently regardless of reputation systems. Accordingly, we plugged in the same normal consumer/provider behavior models to COMPARS for a fair evaluation of three reputation systems. That is, if a consumer whose feedback reliability rate is 1.0 has a transaction with a provider whose service quality rate is 1.0 , the consumer's feedback about the transaction is the highest value in each system.

Attacker: An attacker can be a consumer or a provider. In this experiment, however, we assumed that the attacker is a malicious provider who wants to increase its profit in e-commerce markets so as to clearly show the attacker's profits while it carries out the best actions. For simplicity, we assumed that there is a single attacker in the system who has two possible actions at each system state, i.e. provide a bad service (*cheating*) and provide a good service (*not cheating*). Note that we can add more choices of actions to deal with various types of attacks. For example, we may allow creating a new account as one of an attacker's possible actions to capture Sybil attacks.

Reasoning: When an attacker reasons, COMPARS should decide the maximum height M_T of an attack tree for a reasoning process, i.e. the maximum number of reasoning steps. We used 1 as default value of M_T.

While generating an attack tree, we produced 15 sample sequences of normal users' actions based on their behavior models, resulting in transitions to 15 system states at each timepoint. Although an attacker is not involved in the transactions, the attacker's trust score at each system state may or may not be different, because of global aggregations in some reputation systems [13]. For simplicity, we picked a system state as representative for the given time at which an attacker's trust value is the average of an attacker's trust values at 15 system states.

Attacking goal and profit model: We assumed the goal of the attacker is to get an amount of profits G with a default value 60 by gaming the system. Table 2 shows how we calculated an attacker's profits at each system state, whose rationale is explained below.

Although it is hard to define a definite relation, a number of studies have found that in e-commerce markets, the provider's reputation has impacts on the profits it can get in each transaction [19, 22] . Also, Hazard *et al.* [8] and Melnik *et al.* [19] have found that the reputation and profits have a multiplicative relationship on e-commerce markets such as eBay. Along with previous studies, we allocated different amounts of profits that an attacker gains from a single action, depending on an attacker's reputation. Since we used "OVER TRUST" as the default strategy for normal users to pick providers, we assumed that an attacker gains profits depending on its trust value.

An attacker will essentially try to look like normal users so that it will try to maintain its trust value similar to that of normal users. If the attacker's trust value is too low compared to that of normal users, it would be easily identified as untrustworthy and avoided by others; therefore, the attacker will not be able to make profits [19, 22]. We thus assumed that an attacker's profit will depend on how far its trust value is from the the average trust value of users. We divided users into 10 groups according to how far a user's trust value is from the average trust value. Depending on the range of users' trust values, we set a value α. Let T_{avg} and T_{max} denote the average and maximum trust value of users, respectively. Then,

$$\alpha = \frac{T_{max} - T_{avg}}{5}$$

We generated 100 sequences of normal users' actions to compute T_{avg} and T_{max} for the three trust functions.

An attacker in e-commerce markets attempts to make more profits by cheating [9, 14]. Clearly, if an attacker can gain a large profits without cheating, it does not need to cheat. Therefore, we assumed that an attacker makes more profits by cheating, and set a larger value of profit when an attacker is cheating at a specific state.

The entries of Table 2 indicate that if an attacker's trust value is over a defined trust value in the column, $Reputation$, it will get profits depending on its action, cheating or no cheating. The first entry, for instance, means that if an attacker's trust value is larger than T_{avg} + 5 *α, it will get 20 profits if it cheats; 10 profits if it does not cheat.

Even though an attacker may gain more profits by cheating at a specific system state, it does not mean that the attacker's best strategy is to keep cheating all the time. That is because its trust value will change depending on its actions and the profit at a given system state is affected by its trust value. The relationship between attack goals and optimal strategies will be discussed in Section 6.2.

Since an attacker is a malicious provider in our experiments, we assumed for simplicity that normal providers' Q_i is 1.0. Note that a consumer chooses a provider whose trust value is over a certain threshold. Therefore, the attacker can still be chosen by a consumer as long as it maintains its trust value over the threshold.

6.2 Analysis of Existing Reputation Systems with COMPARS

In this section, we present our experimental results and analysis of the three reputation systems by plugging in parameters discussed in Section 6.1 to COMPARS. Note that different conclusions can be drawn from evaluations with different user behavior models, profit models and attack goals. In other words, our results do not mean an absolute conclusion that one reputation system is more or less resilient than another regardless of parameters.

Clearly, an attacker will try to choose the optimal strategy with which it obtains large profits within a small amount of time. That is, an attacker's goal is to reduce the number of transactions to satisfy a given profit goal G; whereas, the goal of trust functions is to increase the number of the attacker's transactions. We thus evaluated the number of the attacker' transactions N_{trans} to satisfy a given profit goal with different normal user behavior models (Section 6.2.1) and different number of reasoning steps (Section 6.2.2).

6.2.1 Analysis with Normal User Behavior Models

As mentioned before, we used consumers' feedback reliability rate and provider's service quality rate to handle non-deterministic nature. Since an attacker in our experiments is assumed to be a malicious provider, we assumed service quality rates of normal providers are 1.0 for simplicity.

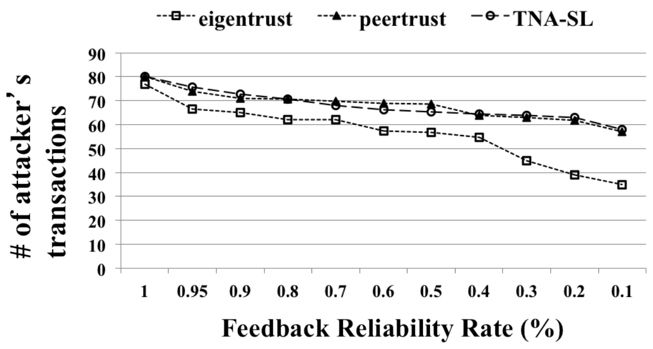


Figure 7: The required number of an attacker's transactions with change in consumers' feedback reliability rate

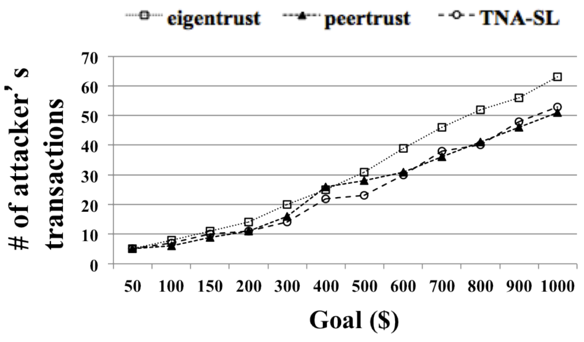
Fig. 7 shows the required number of an attacker's transactions N_{trans} (Y-axis) with varying consumers' feedback reliability rates H_i (X-axis). Even though each user can have different feedback reliability rates, we assumed that every consumer shares the same feedback reliability rate for simplicity.

A low feedback reliability rate of a consumer means that the consumer's feedback is not consistent with true service quality. Therefore, a consumer with a low feedback reliability rate has a high probability of giving a good feedback for an attacker's bad service (i.e., cheating). In other words, the attacker's misbehavior will not be punished under a given trust function, if most users have low feedback reliability rates. In such a case, the attacker's optimal strategy is cheating continuously to achieve its profit goal early, because the attacker obtains more profits by cheating at each system state and cheating will not damage the attacker's reputation much. Consequently, the required number of an attacker's transactions N_{Trans} decreases as consumers' feedback reliability rate H_i decreases, as shown in Fig. 7.

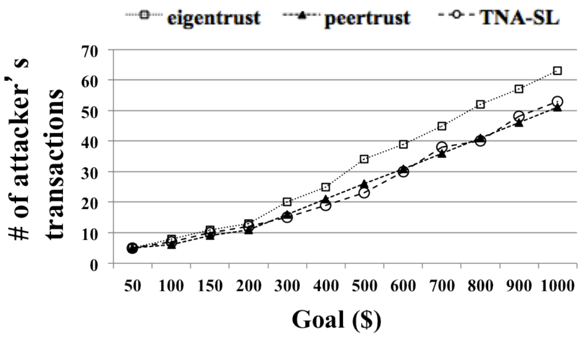
EigenTrust uses the weighted sum of each user's local trust value to compute global trust values and pre-trusted peers are responsible for a big part of the computation because of their large weight. Although pre-trusted peers should have a high H_i, it is possible for pre-trusted peers to issue bad feedbacks because of uncontrollable factors as discussed in Section 3. We thus assumed even pre-trusted peers share the same H_i with other users. Hence, a low H_i of pre-trusted peers allows an attacker to manipulate a system easily. That is, the resilience of EigenTrust greatly depends on the feedback reliability rate of normal users (especially that of pre-trusted peers), compared with PeerTrust and TNA-SL. Accordingly, N_{Trans} under EigenTrust greatly decreases (i.e., less resilient than PeerTrust and TNA-SL), as the feedback reliability rate gets lower.

We considered two typical strategies (i.e., OVER TRUST and OVER RANK) for consumers to choose a provider. To compare the resilience of trust functions under different consumers' strategies, we thus assessed the number of an attacker's transactions N_{Trans} with those two strategies as shown in Fig. 8.

PeerTrust takes 1.0 as a trust value for every user at an initial state. An attacker thus has a high trust value for a while from the beginning of transactions under PeerTrust, because of its initial parameters. An attacker under PeerTrust can thus reach a goal early (i.e., less resilient) with a high trust value compared to EigenTrust and TNA-SL; because the trust value of an attacker is still relatively high, even though it continues to cheat in the first several steps. Since an attacker can get more profits by cheating at one specific state, an attacker's optimal strategy under PeerTrust is to cheat continuously for a while from a initial state. However, the attacker will not have a high trust value if a lot of continuous cheat-



(a) A trust value-based approach (OVER TRUST)



(b) A rank-based approach (OVER RANK)

Figure 8: The number of required transactions to reach a goal with two strategies

ing actions are performed. Hence, N_{Trans} increases gradually, as G increases as shown in Fig. 8.

Similar to PeerTrust, the curve of N_{Trans} shows a gradual increase under TNA-SL, as the goal G increases as shown in Fig. 8; because TNA-SL weighs information from direct interaction. As the number of transactions increases, more direct information will be collected. Hence, the trust values under TNA-SL reflect more accurate information with more direct information, as the number of transactions increases. However, EigenTrust utilizes normalized global trust values so that the difference between an attacker's trust values with more or fewer transactions is relatively small [13]. Therefore, N_{Trans} increases in almost direct proportion to the G under EigenTrust as shown in Fig. 8.

When "OVER TRUST" is employed, a consumer should choose δ_T that defines the minimum trust value required for a provider to be selected. If "OVER RANK" is employed, a consumer should choose δ_R to define the maximum rank for a provider to be selected. To compare the resilience of trust functions under different thresholds, we evaluated the required number of an attacker's transactions N_{Trans} with different values for δ_T and δ_R.

Fig. 9 represents the required number of an attacker's transactions with different values for a threshold δ_T in trust value-based reputation systems. Initially, we used the average trust value T_{avg} of users for δ_T and increased by $p\%$ with the following equation until an attacker does not satisfy the required threshold.

$$\delta_T = T_{avg} + (1 - T_{avg}) * p$$

A large value of δ_T implies that consumers set high standards for providers' reputations. As δ_T increases, the attacker should have more honest transactions (more actions with no cheating) to build a reputation and to be selected as a provider. At each system state, an attacker's profits from not cheating are smaller than the profits from cheating. Hence, the required number N_{Trans} of the attacker's transactions increases gradually, as δ_T increases. And, N_{Trans} increases dramatically, when δ_T increases by 40 % over T_{avg}.

Fig. 10 illustrates the required number of an attacker's transactions with different values for a threshold δ_R in rank-based reputation systems. Initially, we used the top 50 % for δ_R and decrease by p % with following equation until an attacker does not satisfy the required threshold.

$$\delta_R = N_U \times \frac{50 - p}{100}$$

A small value of δ_R means that consumers set high standards of providers' reputations. As δ_R decreases, an attacker should behave honestly in more transactions (more actions with no cheating) to build a reputation and to meet consumers' requirements. As mentioned above, an attacker's profits from not cheating are smaller than the attacker's profits from cheating at each system state. Hence, N_{Trans} increases incrementally, as δ_R decreases.

6.2.2 Effect of Different Number of Reasoning Steps

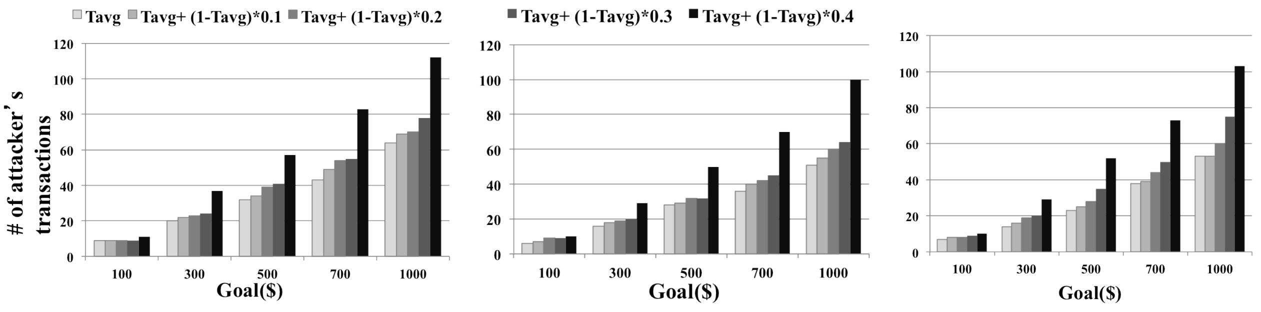
COMPARS can choose different number of reasoning steps to make more or less accurate estimation of the attacker's optimal strategy. Therefore, we evaluated the number of the attacker' transactions N_{trans} to satisfy a given profit goal with changes in the number of reasoning steps as shown in Fig. 11.

As shown in Fig. 11, N_{trans} (Y-axis) decreased as the attacker performs more reasoning steps (X-axis). This indicates that the attacker can achieve its goal much more efficiently with more reasoning steps. The number of reasoning steps essentially mean that the attacker behaves more or less adaptively (i.e., 0 reasoning step means static behavior and more reasoning steps mean more adaptive behavior). Accordingly, Fig. 11 shows that highly adaptive attackers can indeed better game the system, compared to less adaptive attackers.

EigenTrust assumes that there exist pre-trusted peers who are the most trustworthy and computes trust values with a weighted sum of each user's local trust value. Pre-trusted peers are thus assumed to follow static behavior model and the weight of them is much greater than that of other normal users. If an attacker acts badly to pre-trusted peers, it will greatly damage the attacker's trust value, because of their weight. Consequently, it is relatively easy (i.e., less reasoning is needed) for an attacker to guess how a system works and to estimate its optimal strategy (i.e., no cheating, if a pre-trusted peer is involved in a transaction; otherwise, behave depending on each user's behavior). In other words, the attacker can save its time in satisfying a given profit goal G with a small number of reasoning steps. As shown in Fig. 11, an attacker can greatly reduce N_{trans} with only three reasoning steps under EigenTrust.

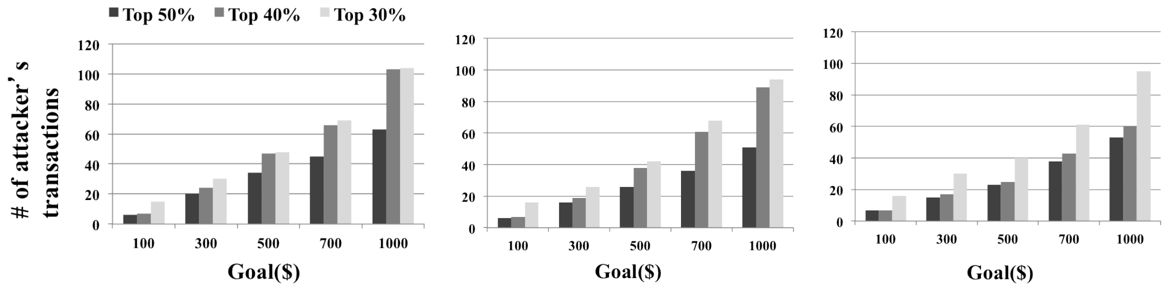
PeerTrust assumes that every user has the highest trust value at an initial state, so that an attacker's first few actions will not have significant impacts on the attacker's trust value. That is, it will be hard for the attacker to guess how a system works within a few reasoning steps. Hence, more reasoning is needed to reach a goal sooner. As shown in Fig.11, an attacker will begin to reduce the number of transactions greatly with reasoning steps more than three.

TNA-SL weighs information from direct interaction. As more transactions have done in the system, users who have relatively many direct interactions may appear under TNA-SL. In such a case, the feedback of those users will have bigger impact on others. In other words, those users who had a lot of direct interactions under TNA-SL can be considered to play a similar role to pre-trusted peers under EigenTrust. Similar to EigenTrust, it will greatly damage the attacker's trust value for an attacker to act badly to those users. It is thus relatively easy (i.e. needs less reasoning) for an



(a) EigenTrust (b) PeerTrust (c) TNA-SL

Figure 9: The required number of an attacker's transactions with different δ_T



(a) EigenTrust (b) PeerTrust (c) TNA-SL

Figure 10: The required number of an attacker's transactions with different δ_R

attacker to guess how a system works. As shown in Fig. 11, an attacker can greatly reduce the number of transactions with only three reasoning steps under TNA-SL.

An interesting finding in Fig. 11 was that with 1 reasoning step, EigenTrust was much more resilient than PeerTrust and TNA-SL; but with 5 reasoning steps, EigenTrust and TNA-SL offer similar resilience. This corroborates that the evaluation of resilience with less reasoning (i.e., static or less adaptive attack) do not reflect the true resilience of a trust function.

7. CONCLUSION

The adaptive nature of strategic attackers presents challenging issues for the evaluation of resilience of trust functions. Specifically, reputation systems based on static user models leave opportunities for malicious parties to exploit systems easily by changing behavior arbitrarily with knowledge of trust functions. This paper presents an evaluation framework for the COMPArison of Reputation Systems (COMPARS), which models adaptive attackers. COMPARS simulates attackers' optimal strategies with an attack tree. We evaluated the resilience of trust functions against attacks by observing how many transactions are required for attackers to achieve their goal based on assumption that a good trust function will restrict the opportunities for attackers to exploit a system,

Acknowledgement

This work is supported in part by the National Science Foundation under the awards CNS-0747247, CCF-0914946 and CNS-1314229, and by an NSA Science of Security Lablet grant at North Carolina State University. We would also like to thank the anonymous reviewers for their valuable feedback.

References

[1] R. Aringhieri, E. Damiani, S. D. C. D. Vimercati, S. Paraboschi, and P. Samarati. Fuzzy techniques for trust and reputation management in anonymous peer-to-peer systems. *J. Am. Soc. Inf. Sci. Technol.*, 57(4):528–537, 2006.

[2] P. Chandrasekaran and B. Esfandiari. A model for a testbed for evaluating reputation systems. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 296–303. IEEE, 2011.

[3] A. Cheng and E. Friedman. Sybilproof reputation mechanisms. In *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 128–132. ACM, 2005.

[4] Z. Cheng and N. Hurley. Analysis of robustness in trust-based recommender systems. In *Adaptivity, Personalization and Fusion of Heterogeneous Information*, pages 114–121. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 2010.

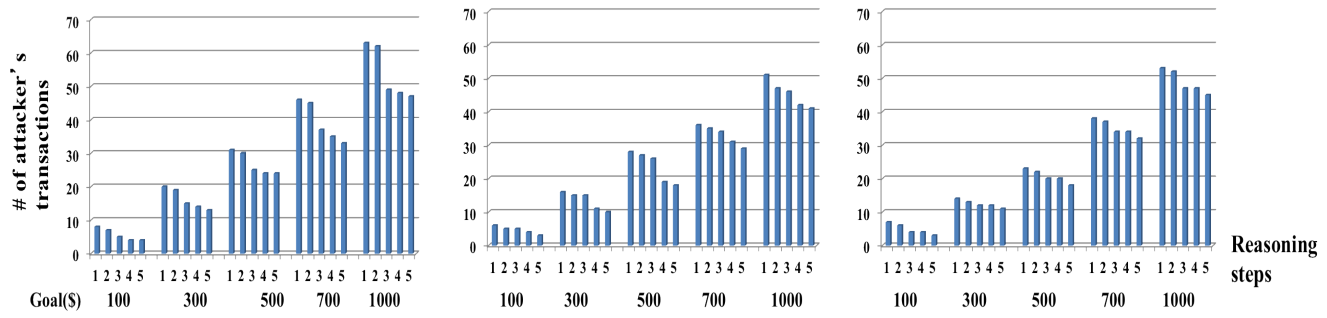
[5] M. Fan, Y. Tan, and A. B. Whinston. Evaluation and design of online cooperative feedback mechanisms for reputation management. *IEEE Transactions on Knowledge and Data Engineering*, 17:244–254, 2005.

[6] J. Feng, Y. Zhang, S. Chen, and A. Fu. Rephi: A novel attack against p2p reputation systems. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 1088–1092. IEEE, 2011.

[7] K. K. Fullam, T. Klos, G. Muller, J. Sabater-Mir, K. S. Barber, and L. Vercouter. The agent reputation and trust (art) testbed. In *Trust Management*, pages 439–442. Springer, 2006.

[8] C. J. Hazard and M. P. Singh. Reputation dynamics and convergence: A basis for evaluating reputation systems. 2009.

[9] K. Hoffman, D. Zage, and C. Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM*



(a) EigenTrust (b) PeerTrust (c) TNA-SL

Figure 11: The effect of the number of reasoning steps

Comput. Surv., 42(1):1–31, 2009.

[10] A. A. Irissappane, S. Jiang, and J. Zhang. Towards a comprehensive testbed to evaluate the robustness of reputation systems against unfair rating attack. In *UMAP Workshops*, volume 12, 2012.

[11] A. Jøsang and J. Golbeck. Challenges for Robust of Trust and Reputation Systems. In *Proceedings of the 5th International Workshop on Security and Trust Management*. Elsevier Science B. V., 2009.

[12] A. Jøsang, R. Hayward, and S. Pope. Trust network analysis with subjective logic. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*, pages 85–94. Australian Computer Society, Inc., 2006.

[13] 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*, pages 640–651. ACM, 2003.

[14] 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, pages 993–1000. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[15] R. Kerr and R. Cohen. Treet: the trust and reputation experimentation and evaluation testbed. *Electronic Commerce Research*, 10(3-4):271–290, 2010.

[16] H. Li and M. Singhal. Trust management in distributed systems. *Computer*, 40:45 –53, 2007.

[17] S. Marti and H. Garcia-Molina. Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50(4):472 – 484, 2006.

[18] K. McNally, M. P. O'Mahony, and B. Smyth. A comparative study of collaboration-based reputation models for social recommender systems. *User Modeling and User-Adapted Interaction*, pages 1–42, 2013.

[19] M. I. Melnik and J. Alm. Does a seller's ecommerce reputation matter? evidence from ebay auctions. *The Journal of Industrial Economics*, 50(3):337–349, 2002.

[20] J. Mundinger and J. Le Boudec. Analysis of a reputation system for mobile ad-hoc networks with liars. In *Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 41 – 46, 2005.

[21] N. J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co., 1971.

[22] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood. The value of reputation on ebay: A controlled experiment. *Experimental Economics*, 9:79–101, 2006.

[23] J. Sabater and C. Sierra. Review on computational trust and reputation models. *Artificial Intelligence Review*, 24:33–60, 2005.

[24] A. Selcuk, E. Uzun, and M. Pariente. A reputation-based trust management system for p2p networks. In *IEEE International Symposium on Cluster Computing and the Grid*, pages 251 – 258, 2004.

[25] Y. Sun and Y. Liu. Security of online reputation systems: The evolution of attacks and defenses. *Signal Processing Magazine, IEEE*, 29(2):87–97, 2012.

[26] Y. L. Sun, Z. Han, W. Yu, and K. J. R. Liu. A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks. In *IEEE INFOCOM*, pages 230–236, 2006.

[27] G. Suryanarayana and R. N. Taylor. A survey of trust management and resource discovery technologies in peer-to-peer applications, 2004.

[28] A. G. West, S. Kannan, I. Lee, and O. Sokolsky. An evaluation framework for reputation management systems. 2009.

[29] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16:843–857, 2004.

[30] Y. Yanchao Zhang and Y. Yuguang Fang. A fine-grained reputation system for reliable service selection in peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(8):1134 –1145, 2007.

[31] Q. Zhang, W. Wei, and T. Yu. On the modeling of honest players in reputation systems. *Journal of Computer Science and Technology*, 24:808–819, 2009.

[32] Q. Zhang, T. Yu, and K. Irwin. A classification scheme for trust functions in reputation-based trust management. In *International Workshop on Trust, Security, and Reputation on the Semantic Web*. Australian Computer Society, Inc., 2004.

[33] R. Zhou and K. Hwang. Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing. *IEEE Transactions on Parallel and Distributed Systems*, 18:460–473, 2007.