

TREET: The Trust and Reputation Experimentation and Evaluation Testbed*

Reid Kerr · Robin Cohen

the date of receipt and acceptance should be inserted later

Abstract To date, trust and reputation systems have often been evaluated using methods of their designers' own devising. Recently, we demonstrated that a number of noteworthy trust and reputation systems could be readily defeated, revealing limitations in their original evaluations. Efforts in the trust and reputation community to develop a testbed have yielded a successful competition platform, ART. This testbed, however, is less suited to general experimentation and evaluation of individual trust and reputation technologies. In this paper, we present TREET, an experimentation and evaluation testbed based directly on that used in our investigations into security vulnerabilities in trust and reputation systems for marketplaces. We demonstrate the advantages of TREET, towards the development of more thorough, objective evaluations of trust and reputation systems.

1 Introduction

The area of *multiagent systems* is concerned with scenarios where a number of agents (who may be acting on behalf of different users) must interact in order to achieve their goals; often, an agent must depend on other agents in order to achieve its objectives. In such scenarios, trust can be an important issue—an agent's ultimate success may depend on its ability to choose trustworthy agents with which to work. For this reason, *trust and reputation systems* (TRSes)¹ have received much attention from researchers. Such systems seek to aid agents in selecting dependable partners (or in avoiding undependable ones).

* This is a pre-publication version of this paper. The final publication is available at www.springerlink.com.

David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, Ontario, Canada
<http://www.cs.uwaterloo.ca/~rckerr>
E-mail: {rckerr,rcohen}@uwaterloo.ca

¹ For convenience, we use the abbreviation TRS, for 'Trust/Reputation System', in reference to both trust systems and reputation systems.

A particular focus for researchers has been on the electronic marketplace scenario, a well-established and important example of a multiagent system. In this setting, agents act as traders, buying and selling amongst one another. The ability to find trustworthy partners is critical to an agent's success, because an untrustworthy agent may deliver an inferior good (or fail to deliver at all), or may not pay for goods purchased. The nature of electronic marketplaces complicates the evaluation of trustworthiness: identity is difficult to establish (because new accounts can be created easily), agents may not engage in repeated transactions together (because of the size of the market and the diversity of products), and an agent may have an advantage over another during a transaction (for example, when a buyer must pay in full before a seller ships (or fails to ship) the good).

Along with the multitude of TRS proposals have come a similarly large number of methods to evaluate the proposals. It has been widespread practice for researchers in the field to develop their own testing methods. A common approach has been to conduct simulations using a scenario of the authors' own devising to show the value of their model, often achieved by pitting their new proposal against the authors' implementations of other existing models. There is nothing fundamentally unreasonable about this approach, in the absence of established testing tools. Unfortunately, there have been significant limitations in the evaluations typically used by authors. It is not surprising that testing scenarios used by authors often favour their own work. This is not to suggest any misdeeds on the part of these authors; when designing a system, it is natural to have a particular scenario in mind, and for subsequent tests to reflect that scenario.

Members of the trust and reputation community have invested significant effort in developing the Agent Reputation and Trust (ART) testbed [1]. A primary purpose of ART is to serve as a competition platform, and it serves this purpose well [2]. While ART is a valuable contribution, a number of design choices make it less appropriate for broad use in the experimental evaluation of TRSes. We discuss these issues in Sect. 2.1.

Perhaps more important than issues of bias, the evaluation procedures typically used obscure critical problems that have received insufficient attention to date by trust and reputation researchers. Simulations typically make use of agents that are simple or naive in their dishonest activities. For example, many simulations (e.g., [3,4]) are populated by random selections of agents that either always cheat or always behave honestly, or by agents whose cheating is governed by simple probability distributions, where each time step is independent of previous ones. Such simulations ignore the possibility that cheaters might behave in a more sophisticated manner—for example, trying to identify and exploit a specific weakness in the system—providing little comfort to those who might wish to consider these proposals for real-world use. In earlier work [5], we identified a number of common vulnerabilities that might allow attackers to defeat the protection offered by TRSes, and argued the critical importance of security in TRSes. Recently [6], we demonstrated the practicality of such attacks by soundly defeating a number of noteworthy TRS proposals. These results demonstrate the need for more rigorous tests, and more objective tests, of TRSes.

This paper is an extended version of an earlier one [7]. In that paper, we proposed a testbed formulation designed to support diverse, flexible experimentation

with TRSes, and more thorough, objective evaluation of such systems. The proposed formulation was based directly on the platform used in our experimentation in [6], which our experience and results had shown to strongly support both goals. The presentation of this proposal was intended to gauge interest and gain feedback from members of the community, towards development of a working platform that would meet the needs of a broad range of trust and reputation researchers. This paper presents the product of this process, a testbed incorporating input from researchers as well as our own subsequent insights. In concert with this paper, an initial release of the TREET software, written in Java, will be made available for use by researchers. This paper notes important aspects of the TREET implementation. While this platform represents a working, useful testbed, we do not see it as a final product. Instead, it is meant as a first iteration, a foundation for ongoing development and refinement within the community.

The TREET platform has a number of important advantages making it well suited for its intended purposes, including:

- It models a general marketplace scenario, allowing systems to be tested under realistic conditions. This includes reasonably large marketplace populations, turnover in the agent population, a large number of products/prices, etc.
- It is modular, allowing new TRSes, buying and selling agents, instrumentation, etc., to be added easily.
- It can support a wide range of trust/reputation approaches (for example, both centralized and decentralized models). It does not impose any particular view of trust on agents, nor does it impose a particular protocol or trust representation on agents.
- It allows collusion to be incorporated into agent behaviour.
- It allows individual marketplace ‘components’ to be tested in isolation. For example, it allows the protection a TRS provides buyers from cheating sellers to be evaluated, without being obscured by other potentially irrelevant issues (for example, whether or not sellers are dishonest with one another). In contrast, the success of an agent in ART requires competence in a number of abilities.
- Given the standardized platform, as new agents/TRSes are developed, they can be evaluated against all existing implementations; at the same time, new implementations constitute new tests for all of the existing systems. In this way, a continually improving battery of rigorous tests can be developed, which can be used by researchers to evaluate their work. (The ‘smart’ cheating agents used in [6], for example, constitute an initial set of tests that have proven very difficult for existing proposals.)
- Standardization also allows for objective benchmarking, permitting meaningful comparison between systems. Moreover, the availability of components allows for results to be reproduced by other investigators.

We believe TREET to be a valuable tool in its own right; moreover, we believe it to be an important step towards more thorough, objective evaluation of trust and reputation systems.

2 Related Work

The majority of TRS proposals applicable to marketplaces have been evaluated using methods of their authors' own devising. (Subsequently, these systems might appear as comparison data points in later proposals' own self-devised tests.) Methods used have included mathematical analysis of properties (e.g., [5, 8]) and simulation (e.g., [3, 4, 9]). Both of these approaches are reasonable. The evaluations performed, however, have proven to be problematic. Because each evaluation is different, results presented by different authors are not comparable. The evaluations presented are often quite brief, leading one to question whether the results thoroughly reveal the performance of the systems in question. Indeed, our investigations [6] revealed numerous ways in which the systems cited above can be defeated—issues that were not revealed in the authors' own analyses. These issues highlight the need for more thorough, objective testing of TRSes, ideally using tools that allow comparison and reproducibility of test results.

2.1 The ART Testbed

A standardized, common testing platform can potentially address the issues noted above. The Agent Reputation and Trust (ART) Testbed [1] is the only well-known trust and reputation testbed of which we are aware. ART has been well supported by the community, and has been used as a competition testbed at a number of conferences.

In ART, agents are art experts, each with varying levels of expertise in different eras. Agents are periodically asked to appraise pieces of art by clients. The accuracy of the appraisals given to clients determines how much business each agent will receive in the future, according to a fixed mechanism used by the testbed. The agent can choose how much to invest in generating its appraisal—greater investment yields greater accuracy. If an agent is asked to evaluate a piece from an era about which he is not knowledgeable, he can seek appraisals from other agents. Agents can also share information with one another about the reliability of other agents' appraisals.

ART is very well-designed for its primary purpose: evaluating agents (who use a number of abilities) in a competitive manner, using a small social trust scenario. ART has a number of desirable properties for a testbed. It offers a well-specified, standardized testing scenario and set of rules. It allows new agents to be easily implemented and plugged into the system; agents can then be used by others for future experimentation. It provides objective metrics for comparison between systems. That said, ART has a number of features that make it less suited for general-purpose trust and reputation experimentation. Other authors (e.g., [10, 11]) have noted obstacles to using ART for evaluating their own work.

Under ART, the distinction between buying and selling agents is unclear, making some forms of experimentation problematic. The ultimate purchasers of appraisals (the 'clients') are buyers, and as such, agents serve as sellers for these transactions. Note, however, that clients' method of choosing appraisers (based on past performance) is fixed by the ART specification, precluding experimentation with buyer-

side modelling of sellers for these transactions; similarly, it obviates investigation of sellers modelling potentially unreliable buyers. In contrast, as agents buy and sell appraisals with one another, each agent acts as both buyer and seller. Success under these circumstances requires skill in several areas: determining when to make do with your own knowledge, and when to seek help; determining how much to invest in appraisals; determining whom to trust when seeking appraisals; and determining whether or not to be honest when another agent asks you for help. While this is a demanding test, and one appropriate for a competition testbed, it can also obscure the role of each individual skill in an agent's performance. This makes it difficult to isolate individual marketplace components for evaluation. For example, if a researcher wishes to evaluate the performance of a system intended to allow sellers to model untrustworthy buyers, it may not be useful to have the results clouded by the same agent's performance in the unrelated task of deciding whether or not to make honest sales to other agents.

In its role as a competition testbed, ART requires a very well-defined scenario. Unfortunately, this requirement seems to limit the flexibility of the system for experimentation. A number of design choices constrain the range of investigations that can be performed using ART. For example, the ART architecture allows decentralized and direct experience models, but precludes testing of centralized models, because the method of sharing information amongst agents is specified by the testbed. It also prevents experimentation with models that regulate an entire marketplace (e.g., mechanism-design based approaches). Features of the chosen scenario prevent investigation of important issues. For example, each appraisal has a fixed price under ART, preventing exploration of vulnerabilities such as Value Imbalance (where a seller builds reputation by honestly executing small-value sales, then uses the reputation gained to cheat on larger ones [5]). The quality of an agent's appraisal is reflected in clients' decisions in the next timestep, preventing exploration of vulnerabilities such as Reputation Lag (where a seller can cheat a large number of sellers for a period of time before his reputation is updated to warn other potential victims [5]).

ART provides a heterogeneous environment where agents share reputation information with agents using other trust and reputation models. To permit communication between agents with different internal models, the format of communication is determined by the ART specification. This imposes a specific trust representation for communication between agents (if not for agents' internal use); the imposed format may not map well to the TRS's native representation, potentially disadvantaging the TRS.

Seeking to clarify why ART is not well-suited for some forms of experimentation, we have focused on a number of its limitations for those purposes. After so doing, we wish to reiterate that ART is an excellent tool for its intended purpose: providing a competition testbed for decentralized social trust and reputation models.

In contrast to the competition focus of ART, TREET is designed specifically to support general-purpose experimentation and evaluation of trust and reputation technologies. TREET's design is based directly on the platform used in our study of the security of TRS proposals [6]; pertinent aspects of this study are discussed later in the paper.

3 Testbed

We sought to formulate a testbed that would support flexible experimentation and meaningful evaluation of trust and reputation technologies. Complete marketplaces may have many TRS components, from a range of possibilities: agents who have individual (and heterogeneous) internal models of other agents' trustworthiness, networks of agents that share reputation information, centralized repositories of reputation data, market-wide mechanisms that regulate trading between agents, etc. A potential adopter of a TRS may have to choose between multiple proposals, despite the fact that the proposals use very different methods internally. An adopter may have to assemble multiple TRS technologies to meet the needs of their complete working system, and may need to understand how well these components work together. For these (and other) reasons, a testbed will ideally support experimentation with a wide variety of such components. Thus, we set out to design an architecture that was quite flexible.

At the same time, too general a testbed formulation might also be difficult to apply in practical terms. At best, it may be of little benefit to the researcher, leaving much work to be done simply in preparing the testing platform. Worse, a formulation that is too general can make evaluation of TRSes and comparison of results problematic: different researchers are likely to use very different instantiations of the testbed scenario, raising many of the same issues as the author-devised testing that has occurred to date. For this reason, we have specified a well-defined scenario that we believe is useful for a wide range of experimentation. We believe that this is an appropriate and useful balance between flexibility and standardization.²

3.1 Nature of Tests

For a competition testbed, it is sufficient to supply the testing platform itself; competitors supply the agents, which seek to defeat one another. In contrast, a testbed intended for evaluation and benchmarking requires meaningful tests for candidates to perform. In some fields (for example, computer component benchmarking), a typical approach would be to develop a set of standardized tasks to perform, with well-defined metrics used for comparison. Ideally, the tasks would be representative of real-world demands. For TRSes, however, it is difficult to envision representative 'tasks' that do not involve actual interaction with other agents. The most illuminating tests are likely to be those conducted in a realistic scenario, interacting with other agents. Thus, in our formulation, tests consist of two components (in addition to the TRS technology being evaluated): a well-defined marketplace scenario, and a population of agents with which the candidate TRS must cope.

This formulation provides a great deal of flexibility, as well as the ability to test specific components under controlled circumstances. For example, to test TRSes that attempt to allow buyers to cope with cheating sellers, a test would consist of a set of market parameters, and a population of sellers with specific cheating behaviours. These components are experimental controls; each TRS would then be tested against

² Feedback from researchers seems to indicate agreement with this view.

the same scenario, allowing comparison of the results. (This was the approach used in our study [6].) In comparison, to test TRSes that allow sellers to cope with untrustworthy buyers, a test would include a set of buying agents.

Beyond the benefits noted above, this approach has a number of advantages. First, as agents are developed (both TRS technologies, and ‘tests’), they can be made available to other researchers. This allows the test suite to grow, increasing in thoroughness and rigor, as understanding of TRSes increases. (Our set of cheating agents constitutes an initial set of tests, as outlined in a later section.) Second, the standardization of the platform and the availability of agents allows results to be reproduced by other researchers. In fact, the terms-of-use of the platform are designed to ensure the latter, and foster the former; for details, see Section 4.

3.2 Scenario

We sought to develop a testbed that employs a reasonably general scenario, one in which a variety of roles and strategies can be evaluated. For tests to be meaningful, the platform should model as realistic a scenario as practically possible. In the following, the parenthesized values are default settings representative of a reasonable scenario. (These values were used in our own experimentation [6].) A test specification would include a set of parameter values; the values can be adjusted as desired for experimentation.

We model an ‘advertised-price’ marketplace: sellers offer goods for sale, and buyers choose whether or not to make purchases, and from whom. A fixed set of products (1000) is available for sale. Because we wish to study trust primarily, and not other price-/cost-based forms of competition, there is an established market price for each good: every seller charges that same price for a given good. A typical marketplace will have more inexpensive items for sale than expensive ones. To reflect this, the price of each good is randomly determined using the right half of a Gaussian distribution (i.e., the median occurs at \$0, and probability decreases as price increases).

A seller incurs cost in producing or acquiring each good that he sells. This is a primary motivation for cheating—to avoid incurring this cost, and thereby increase profit. Again, to keep focus on issues of trust and reputation, rather than profit margins, all sellers incur the same cost to produce each product (75% of selling price).³ Similarly, a buyer who wishes to avoid being cheated can simply refuse to make any purchases. Doing so, however, means that she is doing without products that she needs, incurring a loss of sorts. Thus, each product has a utility value (110% of selling price) that a buyer realizes if a needed product is acquired successfully, which provides motivation to make purchases.⁴ It is assumed that all participants have access to the cost and utility values for each good. These factors support investigation

³ TREET also allows a commission to be charged for each sale (0% by default). The commission is charged based on the sale price, regardless of whether the seller chooses to fulfill the sale honestly or not. This feature allows investigation into issues such as impeding ballot-stuffing by making it costly to engage in fabricated transactions, because each such transaction will incur a commission.

⁴ While these parameters are configurable, cost should be strictly less than selling price, and utility should be strictly greater than selling price, or the motivation to engage in sales is not present.

of a variety of aspects of trust and reputation; for example, both buyers' modelling of sellers, and sellers' modelling of buyers can be examined.

Each seller is assigned a random number of products that she is able to produce, selected from a uniform distribution (maximum of 10). To reflect the greater availability of less expensive products, the products are again randomly assigned using the right half of a Gaussian distribution (i.e., the median occurs at the least expensive product, with declining probability as price increases).

A simulation run can be populated by an assortment of agents, as desired by the researcher, or as defined in a test specification.

Marketplaces are usually dynamic—traders join and leave regularly. This is important for TRSes, because new agents are unknown (and have no knowledge of other agents), and departing agents result in obsolete knowledge. For efficiency, agents join/exit the market at specific intervals (100 days). After each such interval, each agent departs the marketplace with a fixed probability (0.05). That said, it may be undesirable for the performance of TRSes to be clouded by changes in market size (e.g., profits increasing because the number of buyers increases.) Thus, for every departing agent, one agent of the same type joins, keeping the participant count constant.

3.3 Architecture

TREET is designed to be quite versatile for experimentation, within the constraints of the defined scenario. The architecture is depicted in Fig. 1. In this diagram, BA and SA refer to Buying Account and Selling Account respectively. BE and SE refer to Buying Entity and Selling Entity respectively. All components labelled in italic text are components that are intended to be provided/modified by investigators making use of the testbed. The grey box denotes those components that are observable by marketplace participants, although this does not imply complete visibility. For example, seller accounts may be visible to buyer accounts, but this does not imply that all seller account data is visible. Such limitations are described in more detail below.

Each complete run of the testbed is represented by a *SimulationRun*, into which the necessary arguments and objects are passed. A *SimulationRun* is responsible for setup and configuration of a run—creation of the product set, initialization of components, etc.—and initiating the *Simulation Controller*. A set of numerous tests can be executed by creating multiple instances of *SimulationRun*.

A *Simulation Controller* is responsible for actual execution of a simulation run. The controller triggers each of the day's events in turn, signalling the appropriate parties when they are required to take action. For example, the controller cues sellers to make product offers at the appropriate times, cues buyers to select products/sellers when offers have been posted, etc.

The scenario makes use of a single centralized marketplace (a notable example of which would be eBay), represented by a *Marketplace* object. All offers, acceptances, and payments are made through the *Marketplace*. All accounts reside in the *Marketplace*, and requests to open accounts are processed through it.

One important aspect of TREET is the role of TRSes and agents. Some TRSes are implemented entirely centrally, some entirely within the agents; many fall between

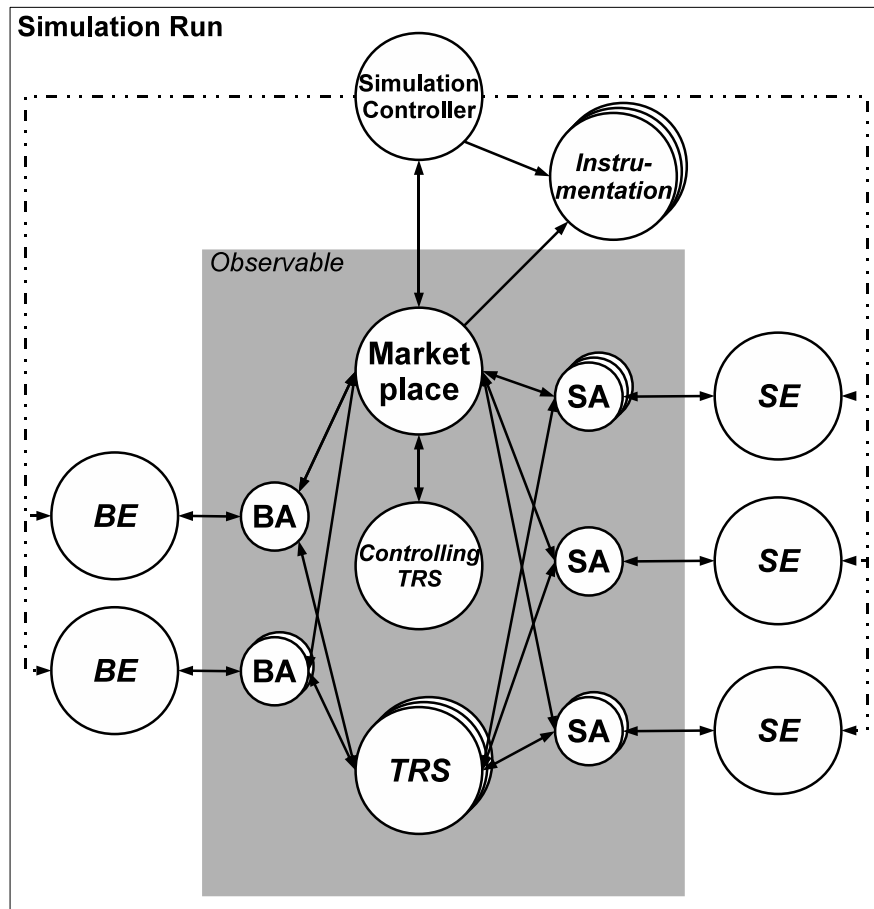


Fig. 1 The TREET Architecture.

these two extremes. The use of both agents (represented by accounts and entities, as described below), and TRS objects, facilitates a wide range of approaches. A *TRS* object implements those components of a TRS that are shared by multiple agents. For example, in a model that makes use of a centralized repository of reputation information, the TRS object would provide that service. TRS objects are useful even in fully decentralized systems, if only to coordinate trust-related actions. For example, when a buyer requests reviews from other buyers, this request could be processed through the TRS object, whose role is simply to co-ordinate such communication.

Some important points should be made regarding TRSes. First, as depicted in the diagram, multiple TRSes may be in use simultaneously, for example, by a heterogeneous population of agents. Second, in order to implement a system for experimentation, matching TRS objects and entities typically must be developed. The role of each component, and the nature of interaction between TRS and entity, is dependent on the

characteristics of the TRS in use. For example, in a completely decentralized model, entities may do all reputation tracking and computation; in this case, the TRS might simply serve as the communication channel between agents. At the other extreme, with a completely centralized model, the TRS may perform all reputation-related functions, while entities simply make use of the services provided. This model seems to provide a great deal of flexibility, without undue complication. Third, in some cases (e.g., a market-wide mechanism), a TRS is tightly integrated into the operation of the marketplace itself. For example, Basic Trunits [5] controls what offers may be made by sellers. TREET allows for a single *ControllingTRS* to be installed in the marketplace. When a ControllingTRS is present, it can control which offers are permitted to be posted by sellers, modify those offers, control whether a buyer will be permitted to make a certain purchase, etc. In the absence of a ControllingTRS, the marketplace executes all offers/purchases/etc. without interference. Finally, note that both buyers and sellers can make use of TRSes. This allows for experimentation and evaluation of, for example, TRSes that help sellers to select buyers that will provide honest or favourable reviews.

Note that no particular trust representation, or communication protocol is enforced between agents. In fact, it is up to the designer of the TRS component, along with the associated agents, to determine exactly how (or if) communication takes place between agents. This provides support for a wide range of approaches to trust. Note, too, that communication between heterogeneous agents can be supported. Certainly, mapping from one agent's trust representation to another's may be necessary, but the method for doing so is up to the designer of the TRS component used for communication. (This, in turn, also allows experimentation with different means of allowing heterogeneous agents to interact.)

Another important feature of TREET is the separation of the agent roles into two components: accounts and entities. *Accounts* represent actual user accounts within the marketplace; these are the identities that are observable by other parties in the market. *Entities*, however, represent the actual agents performing actions by using the accounts. Entities are not observable by marketplace participants, reflecting the fact that identity is difficult to establish, particularly in large electronic marketplaces. This distinction is important for a number of reasons. It allows entities to interact with the marketplace (via accounts) without revealing their true identities to other marketplace participants, important in modelling many real-world scenarios. It allows the re-entry phenomenon to be incorporated into experiments (where agents can simply open new user accounts to shed a disreputable identity). It allows for a single agent to control multiple user accounts (as they may in real-world scenarios), as used in attacks demonstrated in our study [6]. It also allows for investigation of collusion. In the case of perfectly loyal and coordinated collusion, a single entity can represent the entire coalition; in the case of less perfect coalitions, entities can be implemented that communicate with one another outside the observable marketplace.

Buying and selling accounts are shown as distinct in our architecture diagram, as are buying and selling entities. TREET supports this strict separation, which is useful for some forms of experimentation. (For example, one might wish to charge different amounts for opening buying and selling accounts.) In the interest of flexibility, however, TREET also allows a single account to be used for both buying and selling, and

a single entity to act as both buying and selling entity. Whether or not these dual roles are permitted may be configured at runtime. This flexibility allows, for example, the study of forms of collusion such as ballot stuffing, by permitting entities to play the role of both buyer and seller.

For different components of the testbed to communicate with one another, certain aspects of communication must be standardized. In TREET, communication for actual market transactions (i.e., actual purchases) is defined by the specification: the syntax and semantics of product offers, offer acceptances, etc. These are items that are likely to be standardized in a real marketplace situation. Note, however, that the characteristics of communications between TRS components (e.g., exchange of reputation information between agents) are not imposed by the specification, instead left to be determined by those implementing TRS/agents for the system. This ensures maximum flexibility and fair treatment of different TRS approaches. Using this architecture, we evaluated five noteworthy TRSes [3–5, 8, 9]. These models use a variety of approaches, including direct experience, witness information, and centralized mechanisms; for each model, agents were able to represent and communicate trust in the native form as described by the authors, without conforming to a specification imposed by the testbed. This demonstrates the versatility of the platform.

To support flexibility of experimentation and evaluation, TREET must support flexibility of measurement, as researchers investigate a wide range of issues. This support is provided by *Instrumentation* modules. Each instrumentation object is provided with the identity and group-membership of every entity, the ownership of every account, and the details of every sale completed. With this data, Instrumentation objects can provide arbitrary output and metrics, from detailed turn-by-turn or sale-by-sale information, to heavily summarized aggregate data. A default Instrumentation class that accumulates sales/profit/cheating statistics over time, by agent group, was used in our experiments, and is provided with TREET. (A chart illustrating the data accumulated by this module is provided in Section 3.6.) Researchers are also free to develop their own. Multiple Instrumentation objects may be in use during a simulation. Instrumentation modules also serve an important role in test suites, determining whether a test has been passed or failed.

Several other details are worth noting:

- Buyers do not know of selling accounts until that seller makes an offer. Sellers do not know of the existence of a buying account until it makes itself known by accepting an offer.
- At the time of making an offer, sellers do not know or control whether an offer will be accepted, or by whom.
- A seller can only *provide* products that she is able to produce. A seller is able to *advertise* and *sell* (dishonestly) any product, however.
- It is possible for an agent to connect to multiple TRS objects. This allows experimentation with situations where, for example, an agent might make use of shared reputation information with trusted neighbours, as well as accessing data in a centralized repository (i.e., a different TRS).
- Entities can create new accounts at will. TREET provides the ability to charge a fee for each account opened; the fee is determined by the Controlling TRS, if

one is in use. Entities can query to determine the fee, before deciding whether to create an account.

- A ControllingTRS has the ability to close accounts, rendering them unusable by their owners.
- Entities are not resource-constrained—they have unlimited money available. Instead, we track each entity’s net financial position, the sum of all monetary gains and losses, so we can determine how effective the entity’s strategy has been.

3.4 Simulation Execution

Each round represents one day. Each round consists of the following steps (co-ordinated by the System Controller):

1. All participants are notified that a new round is beginning, so that they may do any needed processing/initialization. Such initialization might include resetting internal data structures, opening new accounts, etc.
2. After entering into a sale, a buyer will not know whether or not he has been cheated until after some number of days has passed, reflecting processing, shipping, etc; we refer to the rendering of feedback after this *lag* (14 days) as the *completion* of the sale. At the beginning of each day, each buyer is notified whether each completing sale was executed honestly or not, representing ‘delivery’ of the item.⁵ The buyer’s net utility is updated to reflect the result: the utility is equal to the full utility for the product times the degree of fulfillment. (Note that a buyer only earns utility for a product that meets an assigned need (see Step 6); if it buys an unneeded product, it receives zero utility.)
3. After learning about the outcome of each completing transaction, the buyer determines its satisfaction with the transaction. Each buyer has the opportunity to submit feedback to all TRSes to which it belongs (applicable to those TRSes that require agents to report results, rather than those in which agents respond to queries). It is also prompted to submit feedback to the ControllingTRS (if one is in use.)
4. Each sellers is prompted to submit feedback to all TRSes to which it belongs, and to the ControllingTRS, as above.
5. Each market participant, including each TRS in use, is prompted to process all feedback that it has received in this round.
6. Each buyer’s needs are determined for the day. Each buyer is randomly assigned a set of products (up to 5) that it needs to purchase that day; again, these are selected using the right half of a Gaussian distribution, so there is a greater likelihood of needing lower-priced items.
7. Each seller is prompted to make offers, submitting them to the marketplace. No limits are placed on sellers’ capacity or inventory; only one offer per product is allowed for a given account. If a ControllingTRS in use, it may reject offers in the

⁵ The degree of fulfillment is represented by a value in the range [0, 1], with 1 representing complete fulfillment, and 0 representing complete lack of fulfillment, e.g., not shipping the good at all. In our experiments, sellers selected between these two extremes, but researchers are free to implement other behaviours.

set submitted by a seller. After submitting its offers, the seller is returned a list of the offers that were accepted for posting by the marketplace/ControllingTRS. If it wishes to, it can then modify its offers and post a new set, replacing the previous set. This cycle can be repeated as many times as needed, until the seller is satisfied.

8. Buyers select the products they wish to purchase, from which sellers, by consulting the posted offers. Buyers are free to consult their TRSes in so doing. For each purchase they decide to make, an offer acceptance is communicated to the corresponding seller account, via the marketplace. The acceptance of an offer can be rejected by the ControllingTRS. A seller can also refuse to make the sale to the buyer (e.g., if the buyer has a bad reputation for giving poor reviews); it can consult its TRSes in making this decision. The buyer is notified if the purchase was successful; if not, it can try again. This cycle can be repeated as many times as needed, until the buyer is satisfied.
9. For each sale that the seller agrees to make, it decides whether or not to fulfill it honestly or dishonestly. The cost to the seller for providing the good is equal to the full cost of the product times the degree of fulfillment (as described above). Acceptances are communicated to the marketplace, which forwards each to the corresponding buyer account.
10. Payment is transferred from the buyer account to the seller account, for each sale.
11. Each sale's status (honest or dishonest) is communicated by the seller to the marketplace for storage. This value is not observable by any other marketplace participant, until the buyer is notified during Step 1 of a later round.
12. All participants are notified that the round is ending, so that they may do any needed processing/cleanup.

3.5 Implementation

TREET is implemented in Java 6. Release versions of the platform will be available from the authors' web site. In this section, we detail several noteworthy implementation issues.

The testbed software consists of three main packages: *platform*, containing the core classes required to run the testbed; *interfaces*, specifying the requirements for developers to create TREET components, and *provided*, a set of pre-built components that may be used in TREET (either directly, or as the basis for extension). The majority of the classes in the platform package are marked 'final': they must not be altered, for the results to be comparable to others generated using TREET. (Researchers are free to modify these classes for their own purposes, however; please see Section 4 for details.)

3.5.1 Entities

The TREET design contains buying and selling entities; while there are tasks that might apply to both types of entities (e.g., beginning of round initialization), there are tasks that are specific to one type or another (e.g., making offers for selling entities,

accepting offers for buying entities). TREET also allows for entities that are both buyers and sellers, however, meaning that a single entity may need to perform the tasks of both roles. Unfortunately, limitations of Java (especially the lack of support for multiple inheritance) make a clean implementation of this situation challenging. In TREET, this has been addressed by using a single entity class that can perform all of the tasks for both the buying and selling roles. The entity class can be configured, however, to act as buyer-only, seller-only, or both-buyer-and-seller; tasks and signals that are not appropriate for the configured role are disabled. The same approach has been used to support accounts that may be used for buying, selling, or both.

Entities are implementations of agents, developed by researchers as desired for experimentation. As such, entities are responsible for executing the tactics/strategies desired by their creators. In addition, however, TREET entities have a number of administrative roles to perform: they respond to signals, they store operational information, they communicate with other marketplace components using pre-defined protocols, etc. We sought to provide the administrative functionality common to all entities, so that researchers could focus on behavioural issues. Further, we sought to ensure that critical operational information (e.g., the entity's ID and group membership, used for tracking statistics) isn't modified by entity implementations, inadvertently or otherwise. Given peculiarities of Java's inheritance mechanism, to meet these goals entities had to be decomposed into two parts. The `EntityBase` class is a provided, final class that implements the administrative functionality. To create a new type of entity, the designer implements the `EntityIntel` interface, which encapsulates all of the 'intelligent' aspects of the entity. Every `EntityIntel` object is paired with an `EntityBase` when it is created in TREET; all interactions between the `EntityIntel` and the marketplace take place through the `EntityBase` (which, in turn, connects to the marketplace using one or more `Accounts`).

Each variety of `EntityIntel` is likely to have its own configuration requirements; a designer must provide a `PopulationFactory` class that supplies configured instances of the `EntityIntel` (along with any `TRSEs` with which the `EntityIntel` is designed to operate), freeing other marketplace components from dealing with configuration issues.

3.5.2 *TRSEs*

As noted earlier, TRS design varies widely. For this reason, TREET imposes very few requirements on TRS structure or operation. The TRS interface provides methods to receive timing signals, and to receive information from the marketplace about completing transactions. Beyond this, the TRS interface specifies only methods for communication between agents and the TRS. Note that (using the services provided by TREET) entities do not communicate with TRSEs directly; instead, entities use their `Accounts` to join and communicate with TRSEs. This helps to preserve the secrecy of each entity's identity, and allows, for example, a single entity to use multiple `Accounts` while participating in multiple TRSEs.⁶

⁶ Note that there is nothing preventing `EntityIntels` from communicating directly with one another, or with a TRS, if the designer wishes to implement such functionality. Doing so, however, may allow

Because the communication takes place via Accounts, pre-defined methods must exist in the Account class, and in the TRS interface, to allow passing of messages. It is an explicit goal of TREET, however, not to impose any particular representation or form of communication on TRSes. This is accomplished by leaving the message formats completely unspecified. For example, to request a review of a particular seller from a TRS, a buyer sends a Request object to the TRS. Request is an interface containing no methods; the TRS designer can implement it to convey whatever information desired. Thus, the communication methods provided are essentially mechanisms for passing arbitrary messages.

3.5.3 Running Tests

TREET provides three means for launching tests.

The first, most flexible means, is to create an instance of SimulationRun from within your own Java program. This allows you to create and configure the desired objects before passing them into the run. It also allows you to run a series of tests, by creating multiple SimulationRuns.

The second means is via a launcher that accepts text-file based configuration. This allows you to launch a single run, configuring simulation parameters and market components via the configuration file.

The third means is provided by a test suite. Such a suite consists of a set of tests for a TRS/entity implementation. To execute the test, the test suite launcher is executed, passing in the PopulationFactory that produces the TRS/entity instances. The test suite then executes the tests, and outputs the results.

3.6 Initial Test Set

Much of the value to be gained from an evaluation testbed such as this is the value of the tests: their difficulty, their breadth, and their representativeness of the sorts of issues TRSes might face in a real environment. As an initial set of tests, we provide the set of agents used in our earlier investigations [6]. These agents employ a number of different tactics (consisting only of honest/dishonest transactions that can be executed within the marketplace) based in part on the problems described in [5]. These agents were designed to test the robustness of TRSes that attempt to cope with dishonest sellers; as such, each agent described below is a seller. They seek to cheat profitably, despite the use of the TRS in question. This test set is far more extensive and difficult than any we have seen used for evaluation of TRSes to date. (We also provide our implementations of the TRSes and buying entities used in this study.)

The test set includes the standard, randomly cheating agents employed in so many evaluations. Beyond this, it includes the following agents:

- The **Proliferation agent**, who seeks to win an abnormally large portion of sales by offering products through many user accounts simultaneously, crowding competitors out of the market.

participants to gain information about the true identity of other participating EntityIntels, which may be inappropriate for some types of investigation.

- The **Reputation Lag** agent depends on the delay that exists in many marketplaces between the time he is paid by the buyer, and the time that his reputation is updated to reflect the outcome of the transaction. This agent seeks to build a positive reputation, then use this reputation to cheat as many buyers as possible in the brief period before his reputation is updated to warn other buyers of the change in behaviour.
- The **Re-entry agent**, who creates a new account, attempts to use the non-disreputable status of this new account to cheat as many buyers as possible, then abandons the account and creates a new one, to begin the cycle anew.
- The **Value Imbalance agent**, who seeks to gain good reputation through honest small-value sales, then use that reputation to cheat buyers on large-value sales.
- The **Multi-tactic agent**, who knows how to use all of the tactics described above, and attempts to profitably wield the entire portfolio. This agent is especially intended to undermine the notion of ‘security by obscurity’, that a TRS might be safe from such tactics if the would-be cheater doesn’t know which TRS is in use (and hence its specific vulnerabilities).

As demonstrated in [6], this set of attacks was quite devastating to the set of TRSes evaluated—all of the TRSes were defeated by numerous attacks, and none withstood the Multi-tactic agent.

In actual implementation, we found it useful to decompose our entities into two parts: the actual entity itself, and tactic modules. Each tactic module contains a particular behaviour, for example, the method for launching a particular attack. An entity, then, makes use of one or more tactics to execute its trading activity. This design allows tactics to be re-used. More importantly, it facilitates the design of agents that employ multiple tactics. As noted above, our multi-tactic agents provide an extremely difficult (but realistic and practical) test for TRSes.

Figure 2 depicts one run from our study [6], pitting the Basic Trunits TRS against the Multi-tactic agents. This figure illustrates some of the data generated by the default Instrumentation module. In this chart, ‘smart’ agents are those employing the multi-tactic approach. Honest sellers, and sellers who cheat randomly are included for comparison. The dashed lines represent the revenue from sales for the day in question (smoothed for presentation), while the solid lines represent profit (i.e., revenue less the cost incurred to furnish the goods). There are an equal number of agents in each group. Note that the multi-tactic agents are far more profitable than the other groups—cheating is by far the most profitable policy, meaning that the TRS has failed this test.

Table 1 shows the results obtained by the multi-tactic agent against all of the TRSes tested in our study. The first column in each table represents the average sales (in dollars) per multi-tactic cheating agent, relative to those of an honest seller. The second column reflects the profit realized by a cheating agent, relative to an honest one. (Sales are generally more profitable for cheating agents, because they do not incur the cost of honestly furnishing the good.) Results greater than 100% mean that the average multi-tactic agent makes more money than an honest agent. For example, a value of 124% would mean that cheating agents earned 24% more than honest agents per capita. Here, the cheating agents make far more money than honest ones—

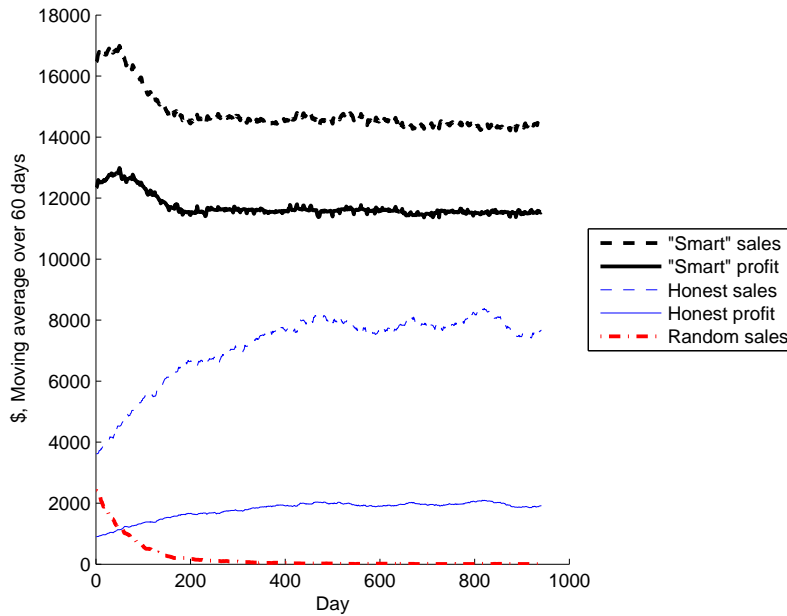


Fig. 2 Basic Trunits, vs. Multi-tactic cheating sellers.

TRS	Cheater sales (% of honest)	Cheater profit (% of honest)
Tran & Cohen [9]	1775.3%	6765.1%
Beta [8]	107.1%	288.3%
TRAVOS [3]	274.6%	613.0%
Yu & Singh [4]	274.9%	723.4%
Basic Trunits [5]	181.8%	577.7%

Table 1 Sales/profit for sellers using multiple attacks.

quite troubling, because this means that a profit-maximizing agent should choose to cheat rather than be honest. All of the TRSes have failed the test.

This set of tests is certainly not exhaustive. Further, it only tests TRSes against dishonest sellers, rather than dishonest buyers, agents that lie to one another about their opinions of other agents, etc. Nonetheless, it constitutes a substantial initial test suite for this important aspect of TRS operation.

4 Use and License

TREET is meant to provide an experimentation platform for researchers. Thus, TREET will be released as open source software, so that researchers can freely use and modify it for their own purposes. A key goal for TREET, however, is to provide objective,

reproducible, and increasingly thorough evaluation of TRSes. To that end, there will be some requirements for publishing of results, enforced by the software license, to support the vision for TREET and its value to the trust and reputation community.

In order for authors to state that their results were ‘Generated using TREET version X’ or ‘Generated using TREET Testsuite version Y’, certain conditions must be met. First, the researchers must not have modified any of the original TREET software; only additions can be made. No classes may be added to the TREET Java packages. In addition, the authors must explicitly state the configuration used (in particular, all parameters that differ from the default values must be noted). These restrictions are meant to allow comparison of results from one publication to another, and to allow readers to have faith in claims. Further, when results generated using a test suite are published, the statement must be included, “Passing these tests does not imply that the system is completely secure, but only that it has passed existing tests.” This is to provide context to readers not familiar with the testbed, or with the issue of security of TRSes.

In addition, any code used to generate published results must be made publicly available on a web site. The authors must reference the location of this code in the publication/presentation, so that others can examine their methods and reproduce their results. Alongside this code, a link to the official version of the TREET software must be provided. The authors must also grant the right for the code to be modified and incorporated in future releases of TREET, under the TREET license; this will allow the test suite to grow in scope and rigour.

If a researcher wishes to modify TREET for her own purposes, she is free to do so, and can publish or present the results, but under three conditions. First, the results in the publications/presentation must be clearly accompanied by the statement, “These results were generated using a modified version of the TREET version X testbed. As such, they are not comparable to other results generated using TREET.” Second, the researchers must make the modified version used, as well as any additional code used to generate the results, available as described above. Finally, all of the original TREET license terms must hold on the newly released version (as is common for open source software).

While we are reluctant to impose any restrictions on use, we believe that the value of TREET to the community would be undermined without them. Note that the description of terms in this section is only a summary; the detailed and official terms of use will be available with the software.

5 Discussion

As noted earlier, TREET models a centralized marketplace scenario. We are careful to note the distinction between decentralized *marketplaces*, and decentralized *TRSes*. TREET provides no impediments to the use of decentralized TRSes, which can be readily implemented. We note, as well, that while the Marketplace is centralized, there is little about the architecture that prevents experimentation with decentralized marketplaces (e.g., a peer-to-peer network where agents sell directly to one another). In this perspective, the ‘marketplace’ is an abstract notion, representing the means

by which offers are accepted, payments are made, etc., within the decentralized system. In this scenario, no centralized TRS would be used. Opening an account, in this case, represents the act of an agent creating a new identity; the marketplace does not limit the creation of accounts, nor communicate their existence to other agents until they reveal themselves by offering/buying products. The one notable violation of the decentralized marketplace perspective is that of offer advertisement. Under TREET, each offer that a seller makes is posted centrally, for all buyers to see; this is analogous to all offers being broadcast throughout a decentralized system. If this limitation is acceptable, then TREET may be suitable for experimentation with decentralized marketplaces.

TREET was inspired by the desire to study the security of trust and reputation systems. It provides (at minimum) a platform for researchers to try to defeat existing trust and reputation technologies. We must be careful to distinguish, however, between defeating a TRS, and defeating TREET itself. While TREET attempts to provide checks against some forms of ‘inappropriate’ use (for example, ‘forging’ an offer by another entity), it is easy to envision ways in which one might bypass these protections. For example, one might create a TRS that is rigged to steal identities and provide them to colluding agents to be used for crimes. Given the nature of the platform, and the fact that developers have access to the source code, it is trivial to ‘defeat’ the platform. Such activities are well outside the intended purpose of this platform, and provide no insight into trust and reputation. This is one reason that the TREET license insists upon public availability of researchers’ code in order for them to be able to publish results: to allow peers to verify that the results reflect legitimate insight into trust and reputation.

6 Conclusions and Future Work

The TREET testbed allows a breadth of experimentation and a thoroughness and objectivity of evaluation that have previously been unavailable from publicly-available, standardized testing tools. The design of this testbed is a proven one, having been used to shed light on important issues that had previously been unexplored experimentally—in particular, the degree to which existing TRS proposals can withstand cheating agents that actively attempt to circumvent the protections of the system. The platform has shown itself to be flexible, supporting experimentation with TRSes using a variety of approaches.

With this design, we have attempted to allow the greatest degree of flexibility of experimentation possible, while still providing ease-of-use and the ability to generate meaningful benchmarks and comparisons. As a result, not every TRS can be tested using this platform: for example, ones that do not function in marketplace environments.

TREET is intended to allow more thorough testing than has typically been performed for TRSes in the past. As TRS researchers develop new agents, the test suite will grow, increasing the rigor of the evaluations, and the insights provided. TREET’s terms-of-use have been designed to support this vision.

While we believe TREET to be an important tool in itself, and a significant step towards improving the evaluation of TRSes, we do not contend that it is perfect or complete in its current formulation. We hope that this proposal serves as a foundation not only for experimentation, but for discussion within the trust and reputation community. Future extension and refinement of the platform, and of the accompanying test suites, will only increase its value to the community, and its usefulness in furthering the cause of security in trust and reputation system. To this end, we seek involvement from other trust and reputation researchers, to chart the future course for TREET.

References

1. Fullam, K.K., Klos, T.B., Muller, G., Sabater, J., Schlosser, A., Topol, Z., Barber, K.S., Rosenschein, J.S., Vercouter, L., Voss, M.: A specification of the Agent Reputation and Trust (ART) testbed: experimentation and competition for trust in agent societies. In: AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM (2005) 512–518
2. Teacy, W.T.L., Huynh, T.D., Dash, R.K., Jennings, N.R., Luck, M., Patel, J.: The ART of IAM: The winning strategy for the 2006 competition. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'07) Workshop on Trust in Agent Societies, Honolulu, Hawaii, USA (2007)
3. Teacy, W.T., Patel, J., Jennings, N.R., Luck, M.: Travos: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-Agent Systems* **12**(2) (2006) 183–198
4. Yu, B., Singh, M.P.: Distributed reputation management for electronic commerce. *Computational Intelligence* **18**(4) (2002) 535–549
5. Kerr, R., Cohen, R.: Modeling trust using transactional, numerical units. In: PST '06: Proceedings of the Conference on Privacy, Security and Trust, Markham, Canada (2006)
6. Kerr, R., Cohen, R.: Smart cheaters do prosper: Defeating trust and reputation systems. In: Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'09), Budapest, Hungary (2009)
7. Kerr, R., Cohen, R.: An experimental testbed for evaluation of trust and reputation systems. In Ferrari, E., Li, N., Bertino, E., Karabulut, Y., eds.: *Trust Management III*. Volume 300 of IFIP Advances in Information and Communication Technology., Boston, Springer (2009) 252–266
8. Jøsang, A., Ismail, R.: The beta reputation system. In: Proceedings of the 15th Bled Electronic Commerce Conference e-Reality: Constructing the e-Economy. (June 2002)
9. Tran, T., Cohen, R.: Improving user satisfaction in agent-based electronic marketplaces by reputation modelling and adjustable product quality. In: AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, IEEE Computer Society (2004) 828–835
10. Hang, C.W., Wang, Y., Singh, M.P.: An adaptive probabilistic trust model and its evaluation. In: AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems (2008) 1485–1488
11. Harbers, M., Verbrugge, R., Sierra, C., Debenham, J.: The examination of an information-based approach to trust. In Noriega, P., Padget, J., eds.: *International Workshop on Coordination, Organization, Institutions and Norms (COIN)*, Durham University, Durham (2008) 101–112