A Trust Management Framework for Service-Oriented Environments

William Conner[†]
wconner@uiuc.edu

Arun Iyengar[∗]
aruni@us.ibm.com

Thomas Mikalsen[∗]
tommi@us.ibm.com

Isabelle Rouvellou[∗]
rouvellou@us.ibm.com

Klara Nahrstedt[†]
klara@uiuc.edu

† Department of Computer Science, University of Illinois at Urbana-Champaign
Urbana, Illinois, USA

∗ IBM Research Division, T. J. Watson Research Center
Yorktown Heights, New York, USA

ABSTRACT

Many reputation management systems have been developed under the assumption that each entity in the system will use a variant of the same scoring function. Much of the previous work in reputation management has focused on providing robustness and improving performance for a given reputation scheme. In this paper, we present a reputation-based trust management framework that supports the synthesis of trust-related feedback from many different entities while also providing each entity with the flexibility to apply different scoring functions over the same feedback data for customized trust evaluations. We also propose a novel scheme to cache trust values based on recent client activity. To evaluate our approach, we implemented our trust management service and tested it on a realistic application scenario in both LAN and WAN distributed environments. Our results indicate that our trust management service can effectively support multiple scoring functions with low overhead and high availability.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Experimentation, Performance, Security

Keywords

Reputation, service-oriented architectures, trust management

1. INTRODUCTION

In open distributed environments, reputation-based trust management systems enable one party to evaluate the trust of another unknown party based on the feedback that the unknown party has received during its previous transactions with others [6, 22]. The reputation of the unknown party can determine whether or not it meets a minimum trust threshold for future interactions. For example, many participants in online auction sites decide whether or not to enter into transactions with buyers or sellers based on their reputations [1]. Over the past few years, many reputation systems have emerged for applications ranging from e-commerce to Web service selection to peer-to-peer file sharing [6, 12, 22, 21, 23, 14, 17, 8, 15].

Although previous work on existing reputation systems has explored the efficacy and robustness of particular reputation scoring functions as well as algorithms for reputation management in completely decentralized environments, very little attention has been given to supporting the synthesis of feedback from multiple entities while also supporting the use of different reputation scoring functions by different entities over the same feedback data. Such flexibility in choosing reputation scoring functions would be desirable in an open infrastructure-centric environment hosting many services with different requirements for trust.

For example, consider an infrastructure hosting many different services [3, 2]. In such an environment, each service might have its own individual reputation-based trust metrics that it wants to apply to external clients when deciding whether or not to process a request or perform a service on behalf of a client. Similar to completely decentralized environments, infrastructure-centric environments might also require scalability. However, infrastructure-centric environments do not have the additional requirement for complete decentralization, which presents the opportunity for reputation-based trust management to be offered as an infrastructure service distributed over multiple nodes.

To enable services to make customized trust level assessments of incoming requests from clients based on shared feedback about that client's previous interactions with other services, we have designed and implemented a reputation-based trust management framework. Our trust management framework stores feedback on previous service interactions with clients and allows services to compute their own customized reputation scoring functions over the feedback collected. Specifically, we make the following contributions:

1. *Support for multiple trust evaluation metrics*: Since different services might apply different trust evaluation metrics, we developed a trust management service that

supports multiple reputation scoring functions over the same trust-related feedback shared from multiple services.

2. *Trust evaluation caching*: To reduce communication overhead in our trust management framework, we developed algorithms for caching trust evaluation results using Bloom histograms. Our experimental results show that our trust value caching can significantly improve performance.

3. *Usable API for application developers*: The API for our trust management service provides methods for reporting feedback and evaluating trust over the synthesized trust data. The trust management service frees application developers from writing their own trust management software components.

4. *Service implementation, deployment, and evaluation*: Our trust management service has been implemented and deployed in both LAN and WAN distributed environments comprising several nodes with a realistic Web services application scenario.

In the next section, we present the design of our trust management framework. Section 3 describes the implementation of a prototype for our trust management service. Section 4 presents our experimental evaluation, which indicates that our trust management service can be effectively integrated into a realistic application with low overhead and high availability. Related work appears in Section 5. Finally, our conclusions are presented in the last section.

2. TRUST MANAGEMENT FRAMEWORK

In this section, we first present an overview of our trust management framework for combining feedback from multiple services while still supporting custom trust level evaluations by each individual service. After providing a high-level overview, we then present our assumptions and the individual components of our trust management framework.

2.1 Overview

In a large-scale infrastructure hosting many different services (such as [3, 2]), assessing the level of trust in incoming requests can enhance the protection of the services by identifying requests from clients with poor transaction histories. In an open system without credentials issued to clients, reputation-based trust management allows services to make trust level assessments based on the previous behavior of a client. Specifically, services within the infrastructure should be able to deny requests from clients with a history of initiating bad transactions at one or more other services.

Our *trust management service* (TMS) synthesizes trust-related feedback reported from multiple services based on their previous interactions with clients. Unlike many reputation systems that consider a single distributed application with a single reputation-based trust metric, the TMS allows each service to make trust level assessments with its own individual custom reputation scoring function.

The TMS API consists of two methods. The first parameter to both methods is the client's identifier. The second parameter to `report()`, which is described in Section 2.3, contains transaction feedback. The second parameter to `evaluate()` is the reputation scoring function that the
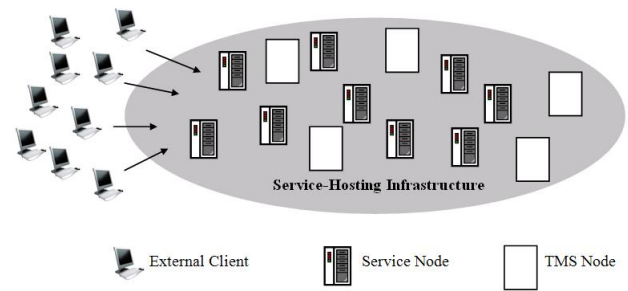

Figure 1: Trust Management Framework

caller wants to apply for the client's trust evaluation. Trust level evaluations are described in Section 2.4.

```
report( Client id, InvocationRecord feedback );

evaluate( Client id, ScoringFunction function );
```

Since our trust management framework must support a large infrastructure, we have also designed the TMS to span multiple distributed nodes to support scalability and high availability. Due to the potential increase in communication and processing overhead during trust evaluations, we have also developed efficient algorithms to support the caching of trust level evaluations to reduce overhead. An overall view of a large-scale service infrastructure using the TMS appears in Figure 1. In Figure 1, external clients make requests to service nodes. These service nodes, which host one or more services, will use the TMS API to report feedback and request trust evaluations from the TMS nodes.

2.2 Assumptions

In our trust management framework, we make several assumptions. The first assumption that we make is that clients do not mask their malicious behavior by spreading it across an unlimited number of identities (i.e., no Sybil attacks [9]). This assumption is common to many reputation management systems [21, 8]. The second assumption that we make is that there is secure communication between the services and the trust management service instances. For secure communication between services, the infrastructure hosting the services can issue digital certificates to each service and trust management service instance to enable public key cryptography for confidentiality and authentication. This would prevent clients from impersonating a legitimate service to report false feedback or initiate a flood of trust evaluation requests. Since our framework is reputation-based, our attack model only considers attacks that can be characterized by negative feedback (e.g., a client not paying for an ordered item). Other attacks, such as SQL injection or buffer overflow, are beyond the scope of this work. Lastly, we assume that the possibility of services intentionally reporting bad feedback is handled by the scoring functions as described in Section 2.9.

2.3 Collecting Feedback

In our trust management framework, the past behavior of a client is represented as a collection of service invocation history records. Each service invocation history record consists of the following fields: client C that initiated the transaction, service S invoked by C during the transaction, nor-

Table 1: Service Invocation History Records

Service	Example Record
Online auction OA	$(C_1, OA, -0.4, \{amount{=}20.50\})$
Video hosting VH	$(C_2, VH, +1.0, \{length{=}90\})$
Bookseller B	$(C_3, B, 0.0, \{items{=}3, amount{=}30.00\})$

malized feedback $Fdbk$ on the transaction (i.e., a feedback value ranging from -1 being the most negative to $+1$ being the most positive), and zero or more optional attributes in a (possibly empty) set of attributes $Attrs$. Since transactions for different services might have very different trust-related attributes, we have chosen to include optional attributes to provide additional contextual information that can be used later by the scoring functions. For example, an online auction service might record the dollar amount of a transaction corresponding to the winning bid. A video hosting service might record the length of the video in seconds. An online bookseller might record both the total dollar amount of an online purchase and the number of items to be shipped. Some service invocation history records for these three scenarios appear in Table 1.

Service invocation history records are created and reported using the following steps.

1. If a client C completes a transaction with service S, then S will create a service invocation history record $H = (C, S, Fdbk, Attrs)$.[1]

    a. $Fdbk$ is the normalized transaction feedback value.

    b. $Attrs$ is the set of optional attributes.

2. Once service invocation history record H has been created, service S will report H to the TMS.

In the service infrastructure, each service has a partial view of client behavior based on its local interactions with each client. By reporting feedback to the TMS, each service can report feedback on these local interactions to the TMS. Each client's aggregate behavior will then be available for trust level evaluations by all services.

2.3.1 Composite Scenarios

In a service-oriented environment, a single transaction initiated by a client might actually invoke many different services rather than a single service. The list of composite services invoked during a transaction might provide some additional context for certain trust level assessments. For example, a scoring function might only want to consider feedback for transactions passing through some service W although no transactions actually terminate at W.

The TMS can handle such composite scenarios by using a special attribute called $path$ that contains the sequence of invoked services during the transaction. The last service in $path$, which will report feedback for the transaction, will appear as the service S in the service invocation history record. An example of a $path$ attribute from a transaction that invoked three services S_1, S_2, and S_3 would be $path = S_1 \rightarrow S_2 \rightarrow S_3$.

---

[1]The record H will not be created until $Fdbk$ and $Attrs$ can be completely determined. For example, there might be a delay between a client requesting a service and detecting that the client did not send payment to the service, which would affect feedback for the transaction.

2.4 Assessing Trust

Since each service hosted within the infrastructure might have its own individual notion of trust for external clients, services supply their custom reputation scoring functions to be evaluated by the TMS using the service invocation history records stored at the TMS. In addition to the client identifier C and feedback value $Fdbk$, the reputation scoring functions can consider many other parameters in the service invocation history records including the service S reporting the feedback as well as any of the attributes $Attrs$. Based on the reputation score computed, services can determine whether or not the reputation of the client making the request exceeds its minimal trust threshold to grant the request. The following steps are performed by services to evaluate the trust level of external clients.

1. Whenever a service S receives a request from some client C, it can send its custom reputation scoring function F_S defined over a collection of service invocation history records along with the client identifier C to the TMS.

2. Upon receiving (C, F_S) from some service S, the TMS will compute F_S over the collection of C's service invocation history records and return the resulting reputation score $Rep_C = F_S(C)$ to S.

3. S will decide whether or not to grant the service request to C based on its own minimum trust threshold T_S and Rep_C computed from F_S (i.e., grant if $Rep_C \geq T_S$).

Since feedback on all client transactions is stored at the TMS, each service can apply its own reputation scoring function to an aggregate view of a client's overall behavior rather than being restricted to a partial view based on the service's local interactions.

2.5 Customizing Reputation

Since many reputation systems only consider a single application with a single reputation metric, one of our primary goals was to allow services to compute their own customized reputation scoring functions over the trust-related feedback data collected by the TMS. This would allow services with different trust requirements to make different trust level assessments for a client with the same given behavior. Example 1 illustrates how two different scoring functions can be used to make different trust level assessments by two services W and X over the same transaction history data for some client C, who has previously invoked services M, N, and P.

In Example 1, although the minimum trust threshold T_X for X is lower than T_W for W, the service W would actually grant a future request from C while service X would deny the request. The reason for these two different trust level assessments for the same transaction history is that each service uses scoring functions that place emphasis on different transaction attributes. Specifically, service W is only interested in transactions initiated by C that invoked M at some point in the service $path$ attribute. In contrast, service X weights transaction feedback by the monetary amount of the transaction specified with the $amount$ attribute. This scoring function F_X has some similarities to PeerTrust, which we explain in more detail in Section 3.3.2 [21].

Example 1. Customization

$F_W(C) = \sum trans.Fdbk$ for all $trans$ initiated by C
where $M \; \epsilon \; trans.Attrs.path$
$F_X(C) = \sum trans.Attrs.amount \cdot trans.Fdbk$
for all $trans$ initiated by C

Minimum trust threshold $T_W = 1$
Minimum trust threshold $T_X = 0$

TMS record $H_1 : (C, M, 1, \{amount = 10.00,$
$path = J \rightarrow K \rightarrow L \rightarrow M\})$
TMS record $H_2 : (C, N, -1, \{amount = 20.00\})$
TMS record $H_3 : (C, P, 0.5, \{path = M \rightarrow P\})$

$F_W(C) = 1 + 0.5 = 1.5,$
so W will grant request to C
$F_X(C) = 10(1) + 20(-1) = -10,$
so X will deny request to C

2.6 Load Balancing

Due to the potentially large number of clients and transactions, it will be necessary for multiple nodes to collectively provide the trust management service. In our trust management framework, we use consistent hashing to uniquely map all of the service invocation history records for a particular client to a particular node running an instance of the trust management service. Consistent hashing has previously been used for some distributed hash tables [16, 19, 18, 11, 24].

Assuming that each service knows all the service names and corresponding identifiers for the trust management service instances in the infrastructure, the services can use some discovery service (e.g., UDDI) to locate a given trust management service instance. In our trust management framework, the following steps are performed for load balancing during feedback collection or trust level evaluation.

1. Whenever service S wants to report feedback or request a trust level assessment for some client C, it will locally determine the trust management service instance identifier $tid = hash(C)$ where $hash$ is a consistent hash based on a cryptographic hash function (e.g., SHA-1 or MD5).

2. The feedback report or trust level assessment request from S will be sent to trust management service tid, which is the trust management service instance responsible for C's service invocation history records.

3. Trust management service instance tid will manage service invocation history records similar to the methods discussed in Sections 2.3 and 2.4.

2.7 Availability

In order to support high availability, the trust management service must handle the possibility of one or more trust management service instances crashing. If a node hosting an instance of the trust management service becomes unavailable, then all of the service invocation history records managed by that node will also become unavailable for trust level evaluations. Also, any service invocation history records that should be reported to some crashed instance would be lost during the downtime.

Replication can be used to enhance availability. In a system with N trust management service instances, each trust management service instance tid will replicate all newly reported service invocation history records on up to K nodes where $K \leq N - 1$. If replication is used and we assume that nodes are ordered $tid = 0, ..., N - 1$, the K nodes that will receive the replicas from tid are $(tid + i)$ mod N for $i = 1, ..., K$. This is similar to the replication scheme used in at least one particular distributed hash table [19].

In the trust management service, whenever a service S wants to report a service invocation history record or request a trust score evaluation for some client C, then it will first make an attempt with the primary trust management service instance tid for that client C. If the remote call for the report or evaluation from S to tid times out, then S will contact the replicas in order $(tid + i)$ mod N for $i = 1, ..., K$ as necessary until a TMS node responds. Whenever a previously unavailable trust management service instance becomes available, it will contact its K replicas to recover any lost data that was reported to its replicas during its downtime.

2.8 Trust Level Caching

Although supporting arbitrary reputation scoring functions defined over service invocation history records allows services to have greater flexibility in evaluating the trust level of external clients compared to many existing approaches, it could also increase the overall average response time of a service if that service evaluates the reputation of a client on each incoming request.

In order to improve performance, we describe an approach in this section that allows services to use previously evaluated trust assessment values until a client has initiated enough transactions to change its trust level assessment. Unlike the simple trust caching scheme used in [21], the TMS only requires fresh trust evaluations when necessary based on the amount of recent client activity. To determine the recent activity of a client, the trust management service instances periodically share data synopses with services in the infrastructure. Each data synopsis is an approximation of the number of new service invocation history records generated by each client. Specifically, we use Bloom histograms for the approximation [20]. In the next subsection, we will describe the Bloom histograms data structure. In the following subsection, we will describe how approximate trust level assessment is accomplished by using previously computed trust values.

2.8.1 Bloom Histograms

Bloom histograms were originally developed to approximate XML path frequency distributions during cost-based XML query optimization [20]. Essentially, Bloom histograms are histograms where each bin has associated membership information represented as a Bloom filter [7]. As we describe in the next section, we use Bloom histograms to approximate recent client activity with a compact representation.

The histograms that we use in our variation of Bloom histograms consist of an array of cells where each cell has a frequency count for the number of values that fall within the range of that cell. Each histogram cell's range is specified by an upper bound with cell membership approximated by a Bloom filter associated with each cell.

A Bloom filter is a compact representation of set member-

Example 2. Approximation

Minimum trust threshold $T_{WS} = 0$
$F_{WS}(C) = \sum trans.Fdbk$ for all $trans$ initiated by C

Case 1: Suppose $Rep_1(C) = 100$ and $X = 5$
Case 2: Suppose $Rep_2(C) = -100$ and $X = 5$

Example 3. Constructing Bloom Histogram

Recent Service Invocation History Records (Step 1)
$(C_1, WS, Fdbk, \{\}), (C_2, WS, Fdbk, \{\}), (C_2, WS, Fdbk, \{\})$
$(C_3, WS, Fdbk, \{\}), (C_3, WS, Fdbk, \{\}), (C_3, WS, Fdbk, \{\}),$
$(C_4, WS, Fdbk, \{\}), (C_4, WS, Fdbk, \{\}), (C_4, WS, Fdbk, \{\}),$
$(C_4, WS, Fdbk, \{\})$

Frequency Table (Step 2)
(Client:C_1, Count:1)
(Client:C_2, Count:2)
(Client:C_3, Count:3)
(Client:C_4, Count:4)

Bloom Histogram (Step 3)
(Elements:$BF(\{C_1, C_2\})$, Upper bound:2)
(Elements:$BF(\{C_3, C_4\})$, Upper bound:4)

---

ship [7]. Bloom filters consist of m bits initially set to 0 with k hash functions that hash inputs into the range $[0, m-1]$. To insert an element x into the Bloom filter, we set each bit indexed by hash function $hash_i(x)$ for $i = 1, ..., k$ to the value 1. An element x is considered to be a member of a Bloom filter if each bit indexed by $hash_i(x)$ for $i = 1, ..., k$ is equal to 1. The possibility of false positives exists in Bloom filters, but false negatives are impossible. Specifically, a Bloom filter with m bits and k hash functions storing n elements has a false positive probability of $(1 - e^{kn/m})^k$ during a membership test [7, 10].

2.8.2 Approximating Trust Levels

The Bloom histograms used by the TMS approximate the frequency distribution of the number of new service invocation records generated by clients during the last P transactions at some trust management service instance. Assume some service WS and some client C. Depending on the scoring function used by WS and the number of new transactions initiated by C, it might not be necessary to compute a new trust score for C. For example, if eBay is the scoring function used by WS, then the maximum positive feedback that C can receive is $+1$ per transaction and the maximum negative feedback that C can receive is -1 per transaction. Therefore, the scenarios in Example 2 would not require WS to re-evaluate C's trust level using scoring function F_{WS} if C has initiated X new transactions since its last evaluation.

In the first case from Example 2, C's current trust level would be 95 in the worst case, which exceeds the minimum threshold to grant the request. Similarly, in the second case, C's current trust level would be -95 in the best case, which would not exceed the minimum threshold to grant the request. Since the reputation score evaluation will lead to the same trust assessment by WS using the previous trust score, then there is no need to re-evaluate the score for C given its current trust level and the number of new transactions since its last trust level evaluation.

Given X recent transactions from some client and its current reputation score, best-case and worst-case estimators can determine the best/worst possible scores by simulating the scoring function with the best/worst possible feedback for the past X transactions. We implemented best-case and worst-case estimators for three different scoring functions, which are described in Section 3. These estimators allow services to determine when a new trust evaluation needs to be initiated for a client.

Using Bloom histograms to estimate the number of transactions initiated by each client, each service can decide whether or not to re-evaluate the trust level of a particular client. Since each client is uniquely assigned to one TMS instance, then that TMS instance can locally determine the number of new transactions initiated by that client based on the reported service invocation history records. For each group of P service invocation history records, the TMS instance will

generate a frequency table mapping each client appearing in those P records to the number of transactions initiated by that client. Here, P refers to the transaction period at the TMS. The Bloom histogram is then computed from the frequency table. Example 3 illustrates how a collection of service invocation history records can be transformed into a Bloom histogram. Assume for a set of elements S that $BF(S)$ represents the Bloom filter for the elements in S.

After every P transactions reported at a TMS instance, that TMS instance will send Bloom histograms to services that have recently requested trust score evaluations. Each service will maintain the most recent reputation score for each client seen as well as an estimate of the amount of recent activity for that client based on Bloom histogram updates from the TMS.

To estimate the amount of recent activity for each client with a previously evaluated trust value, upon receiving a new Bloom histogram update from the TMS, each service will check for client membership in the Bloom histogram bins in order starting with the highest upper bound going down to the lowest upper bound.

The Bloom histogram bins are checked from highest upper bound to lowest upper bound so that Bloom filter false positives in the Bloom histogram bins will lead a service to overestimate recent client activity rather than underestimate. Overestimation might lead to unnecessary trust level evaluations, which could increase the load on the TMS, but it will not help malicious clients mask their behavior by underestimating their recent activity.

2.9 Handling Malicious Services

In our trust management framework, malicious client behavior is addressed by services providing negative feedback ratings on previous interactions with those clients. Another concern is handling malicious services. For example, malicious services might collude to deliberately provide inaccurate feedback in order to boost the reputation of bad clients or hurt the reputation of good clients. Malicious services might also suppress their feedback to be disruptive. To handle inaccurate feedback, some reputation metrics consider the credibility of the feedback sources explicitly or implicitly [21, 15]. In the case of feedback suppression, PeerTrust is an example of a reputation metric that captures community-context factors, such as willingness to provide

feedback [21]. Since the TMS is flexible enough to support customized reputation metrics, services can optionally add components from existing reputation metrics to their own custom scoring function to handle feedback credibility and suppression. A default implementation of popular predicates (e.g., PeerTrust credibility) can be provided by the TMS to make scoring function customization easier.

3. PROTOTYPE IMPLEMENTATION

We implemented a prototype of our trust management service and evaluated it within a realistic composite Web service application scenario. In our prototype implementation, we included multiple reputation scoring functions. In the next three subsections, we will describe the application scenario, our specific prototype implementation details, and the various scoring functions used.

3.1 Supply Chain Management

With input from multiple companies (including IBM, Intel, Oracle, SAP, and others), the Web Services Interoperability Organization defined a standard Supply Chain Management (SCM) application [5]. The SCM application consists of the following components: consumers (i.e., "clients" in our framework), retailers, warehouses, and manufacturers. Each retailer, warehouse, and manufacturer corresponds to a Web service. The consumer submits an order consisting of line items to the retailer. Each line item identifies a product and the corresponding quantity to be ordered. The retailer goes through each line item and finds a warehouse with sufficient stock to ship the line item. This warehouse will then ship the line item to the consumer. Each warehouse has some minimum inventory level for each product and will order additional items from the manufacturer whenever inventory levels fall below this threshold for a particular product.

We used Java Remote Method Invocation (RMI) to build our prototype SCM application for both simulations on a single machine and distributed experiments involving multiple nodes in LAN and WAN environments. A production-quality SCM implementation could use other technologies, such as the Java Web Services Developer Pack.

3.2 Trust Management Service

Our current version of the TMS is also implemented using Java RMI with an interface that provides the two functions for our TMS API described in Section 2.1. The first function allows services to report service invocation history records to the TMS by passing a client identifier and a Java object representation of a service invocation history as a parameter. The service invocation history record information is then stored by the TMS. The second function allows services to perform reputation-based trust level assessments by requesting that the TMS evaluate a client's reputation by passing a client identifier and a Java object representation of their custom scoring function as parameters.

In order to evaluate our trust management service, we enhanced our implementation of the basic SCM application by providing the TMS as an additional service. All of the services in the SCM application were modified to pass along service invocation history records in addition to their regular functionality. For our prototype, we assumed that all client requests for a particular line item would lead to one of the following outcomes: (1) all requested items shipped in exchange for payment, (2) all requested items appeared in catalog, but not all items could be successfully shipped, and (3) some items requested do not even appear in the catalog. These outcomes correspond to positive, neutral, and negative feedback, respectively. The reason that shipped items are considered positive is because a payment has been made. Items unavailable for shipment that appeared in the catalog are considered neutral because no payments are made, but the customer is not at fault for selecting items that appeared in the catalog. Negative transactions are considered to be those transactions where the customer is trying to purchase items that do not exist in the catalog, because such repeated activity might indicate a client attempting to waste resources within the services infrastructure.

Although our prototype only considers one particular type of attack for our performance evaluation (i.e., clients intentionally ordering non-existent items to waste infrastructure resources), the TMS design is general enough to be easily extended for other types of attacks for different applications (e.g., posting spam on some user-generated content hosting service or non-payment to an e-commerce site).

3.3 Scoring Functions

Since our trust management framework is designed to support different services using different reputation metrics, we implemented three reputation scoring functions for our prototype. Each function is described in this section.

3.3.1 eBay

Both buyers and sellers are allowed to provide feedback on their auction transactions with other buyers and sellers on eBay [1]. The eBay scoring function is a summation of positive ($+1$), neutral (0), and negative (-1) feedback values used for the online auction site. Users with higher eBay scores are considered to have higher reputations. The eBay feedback ratings are added up to determine a client's overall reputation according to the eBay scoring function. This overall reputation can be used by potential buyers and sellers to determine whether or not to purchase/sell an item from/to another eBay user.

3.3.2 PeerTrust

PeerTrust is a reputation management system for peer-to-peer networks [21]. The PeerTrust model presents a trust metric based on the following five features: feedback from other peers, number of transactions completed, credibility of feedback, transaction context factor, and community context factor. The transaction context factor represents the importance of the transaction (e.g., transactions involving more money should receive more weight). The community context factor represents "community-specific" information that should be taken into account (e.g., willingness to provide feedback to benefit the overall community).

The general trust metric $T(u)$ for some peer u appears in the following Equation 1 from [21].

$$T(u) = \alpha \cdot \sum_{i=1}^{I(u)} S(u,i) \cdot Cr(p(u,i)) \cdot TF(u,i) + \beta \cdot CF(u) \quad (1)$$

$I(u)$ represents the total number of transactions performed by some peer u with other peers in a given time window and $p(u,i)$ represents the other peer in the ith transaction of

peer u. $S(u, i)$ denotes the normalized amount of user satisfaction peer u receives from $p(u, i)$ in its ith transaction. $Cr(v)$ denotes the credibility of feedback received from peer v. $TF(u, i)$ denotes the adaptive transaction context factor for peer u's i^{th} transaction and $CF(u)$ denotes the adaptive community factor. The variables α and β are weight factors.

Our particular implementation of PeerTrust treats each service as a peer. We ignored the community factor $CF(u)$ (i.e., we set $\beta = 0$ from Equation 1). This is the basic metric with transaction context mentioned in [21]. We also assumed that the credibility $Cr(v)$ of each service was static and equal to 1. The possibility of malicious services providing bad feedback can be handled by adjusting the credibility factor.

3.3.3   Exponentially Weighted Moving Average

In order to test a new user-defined custom scoring function, we developed a third reputation metric, which is an exponentially weighted moving average (EWMA) of the series of feedback ratings x_i for a particular client with an adaptive smoothing constant θ:

$$x_{-2} = x_{-1} = 1$$

$$Rep_0 = 0$$

$$Rep_{i+1} = (1 - \theta)x_i + \theta Rep_i$$
$$\text{where } \theta = \begin{cases} 0.75 \text{ if } x_i, x_{i-1}, x_{x-2} < min_{feedback} \\ 0.95, \text{ otherwise} \end{cases}$$

The intuition behind using the adaptive smoothing constant θ in the above EWMA is that we want a user's reputation to quickly degrade in response to three or more consecutive transactions below the minimum threshold feedback value $min_{feedback}$. On the other hand, we want reputations to increase slowly in response to a series of good transactions that fall at or above $min_{feedback}$. Although EWMA is very aggressive in reducing client reputations quickly, users with consistently good transactions are more easily identified.

4.   EXPERIMENTAL EVALUATION

To evaluate our trust management service, we ran several experiments in both LAN and WAN distributed environments using nodes from the Trusted ILLIAC cluster [13] and the PlanetLab wide area network testbed [4]. We also ran several simulations of our prototype on a single machine. We will first provide some Trusted ILLIAC and PlanetLab results that explore the effects of the trust management service on request latency and throughput for a real application distributed over both local-area network and wide-area network environments. Next, we will present simulation results to measure the effects of Bloom histogram-based trust evaluation caching on communication savings as well as its effects on the accuracy for different scoring functions. Finally, we will present simulation results to evaluate the availability of the TMS when one or more nodes crash.

4.1   Latency and Throughput

In order to evaluate the effect of the trust management service on the end-to-end latency and throughput of a real application, we deployed our implementation of the SCM application on both the Trusted ILLIAC cluster and PlanetLab testbed. Figure 2 depicts our deployment of the TMS with the SCM application. The Trusted ILLIAC cluster
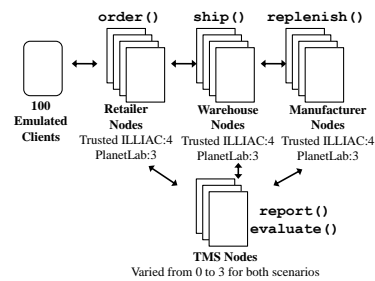

Figure 2: SCM with TMS Deployment

represents services deployed in LAN environments, such as data centers. PlanetLab represents services distributed over WAN environments, such as the Internet. For each experiment, we added between zero and three instances of the trust management service to the SCM application. We considered the following modes: no TMS, TMS without caching, and TMS with caching. When the TMS was used, the scoring function was eBay. Our Trusted ILLIAC experiments consisted of between twelve and fifteen nodes. Our PlanetLab experiments consisted of between nine and twelve nodes. All PlanetLab nodes were located at different institutions throughout the United States. The exact number of nodes involved in each experiment depended on the number of TMS nodes, which varied from zero to three.

In the SCM application, when the TMS was used without caching, the retailer would invoke a TMS node to evaluate each incoming request from the client before ordering a shipment from the warehouse. When the TMS was used with Bloom histogram-based caching, the retailer would evaluate incoming requests only when necessary according to the methods described in Section 2.8. The transaction period (defined in Section 2.8.2) was 100 for all experiments where TMS caching was used. The Bloom histograms for these experiments had 5 bins with 32 bits per bin (i.e., 20 bytes total) and 4 hash functions.

To evaluate end-to-end latency and overall system throughput on a realistic application, we had 100 client threads from a single machine make one hundred random requests for items to the SCM application hosted on either the Trusted ILLIAC or PlanetLab. The Trusted ILLIAC deployment of the SCM application ran four retailers, four warehouses, and four manufacturers. The PlanetLab deployment of the SCM application ran three retailers, three warehouses, and three manufacturers. The number of TMS service instances varied between zero and three. For these experiments, we also examined the effect of using multiple TMS nodes both with and without caching.

The results in Figure 3 indicate that the additional latency when the TMS is used is not significant in LAN environments. In contrast, there is additional latency when the TMS is used in WAN environments. However, Figure 3 also indicates that caching can significantly reduce the end-to-end latency in WAN environments. Increasing the number of TMS nodes also seems to significantly reduce the latency in WAN environments.

As we expected, using the TMS decreases throughput in our experiments (shown in Figure 4), but increasing the number of TMS nodes and caching trust values lead to performance improvements for both the Trusted ILLIAC (i.e.,
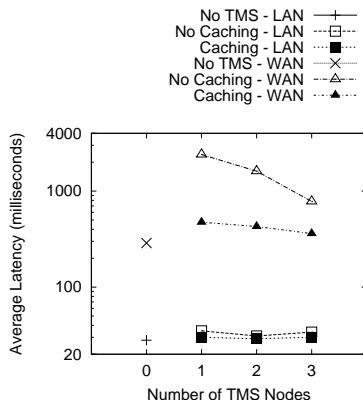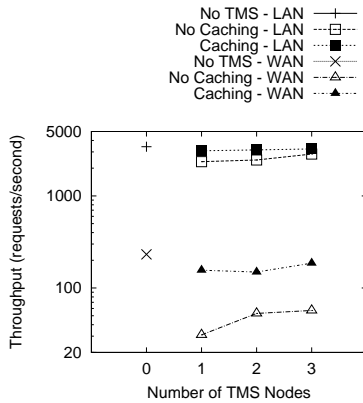
Figure 3: Latency (LAN and WAN)

Figure 4: Throughput (LAN and WAN)

LAN) and PlanetLab (i.e., WAN) deployments. Our experimental results show that even a small number of TMS nodes can come close to the performance of using no TMS nodes at all. Due to our load balancing scheme described in Section 2.6, if the TMS had been more of a bottleneck in our experiments, then we would have expected to see greater performance improvements with the addition of each TMS node.

4.2 Different Scoring Functions

To compare trust level assessments and the effects of Bloom histogram-based caching for different scoring functions, we ran simulations with all three scoring functions. For our simulations, we created an SCM application scenario with ten retailers, ten warehouses, ten manufacturers, and ten instances of the trust management service. Each simulation also had 1000 clients where each client has an activity probability AP and issued approximately $AP \cdot 100$ transactions to randomly chosen retailers. Each simulation used the same random transaction workload.

Each transaction in the simulations had an optional attribute *amount* that corresponds to the total monetary amount of the transaction based on the line items and their corresponding prices in the catalog for the SCM application. The attribute *amount* is only used by the PeerTrust scoring function as a transaction context factor. To create different types of behavior, each client is assigned a probability $Malprob$ for issuing a malicious transaction. As described in Section 3.2,

a malicious transaction attempts to order items that do not appear in the catalog.

To study the effect that different scoring functions might have on trust level assessments for the same collection of service invocation history records, we determined the request rejection rate for different scoring functions in different client behavior categories. Each client behavior category falls into ten bins characterized by their average $Malprob$. As shown in Figure 5, the rejection rate for each scoring function increases as the probability of issuing a malicious transaction increases for the different categories. However, Figure 5 also shows that different scoring functions might lead to different trust level assessments, which supports our goal of providing flexibility in our trust management framework for services with different trust requirements. For example, since EWMA aggressively lowers a client's reputation after consecutive negative transactions, its rejection rate is higher than the other two scoring functions as shown in Figure 5.

Since trust level caching (as described in Section 2.8) would be expected to reduce the number of requests for trust level evaluations (i.e., reduce communication overhead) at the cost of a possible reduction in accuracy, we explore the effects that different transaction periods at the TMS nodes might have on different scoring functions. Shorter transaction periods at the TMS nodes should give services more up-to-date information regarding recent client activity. In general, we expect shorter periods to lead to better accuracy at the cost of more trust level evaluations and more Bloom histogram updates. Similar to our distributed experiments, we used Bloom histograms containing 5 bins with 32 bits per bin (i.e., only 20 bytes total) and 4 hash functions.

The total number of updates sent from the TMS nodes to other nodes during the simulations for different periods appears in Figure 6. The transaction period represents the number of transactions that a TMS node will receive between sending Bloom histogram updates to the services.

In Figures 7 and 8, we show the trade-off between communication overhead and the accuracy of trust level evaluations for three different scoring functions when Bloom histogram-based caching is used. For each transaction period, we consider the following metrics for different scoring functions: false grant rate, false denial rate, and trust evaluation rate. The false grant rate is the fraction of requests granted based on cached trust values when the request should be denied based on the actual trust value. The false denial rate is the fraction of requests denied based on cached trust values when the request should be granted based on the actual trust value. The trust evaluation rate is the fraction of requests that trigger a new trust level evaluation based on best-case/worst-case estimators for a particular scoring function and the client's recent activity as described in Section 2.8.2. Ideally, the rate for false grants and denials should be low for good accuracy. The trust evaluation rate should also be low to reduce communication overhead.

Our results indicate that shorter transaction periods lead to higher accuracy (shown in Figure 7) at the cost of higher communication overhead (shown in Figure 8). In Figure 8, EWMA has a higher trust evaluation rate than the other two functions due to its aggressive response to negative feedback. In Figure 7, the false grant and denial rate of all three scoring functions is extremely low until the transaction period equals 2000. Even when the transaction period is 2000 or above, the false grant and denial rates do not exceed 3% for any of
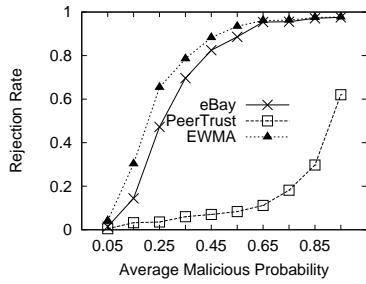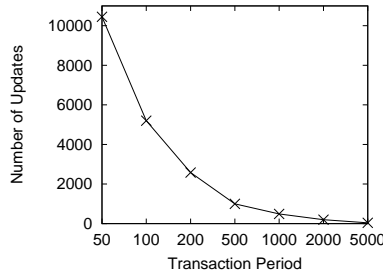
Figure 5: Rejection Rates

Figure 6: Bloom Histogram Updates

Figure 8: Trust Evaluation Rates

Table 2: Robustness to TMS Node Crashes

K	$Prob$	$Crashes$	$Failures$	$Records$
0	0.2	2	13857	47571
0	0.4	4	28617	42667
0	0.6	5	36922	39955
1	0.2	2	0	92635
1	0.4	4	13857	85024
1	0.6	5	13860	84581
2	0.2	2	0	134884
2	0.4	4	0	133330
2	0.6	5	0	133097

the three scoring functions used (i.e., eBay, PeerTrust, and EWMA).

Our simulation results demonstrate how three different scoring functions, which were implemented in our prototype, can lead to three different rejection rates for the same workload. Of the three scoring functions, EWMA had the most aggressive rejection rate. Our simulation results also indicate that Bloom histogram-based caching can effectively reduce communication overhead with a slight reduction in accuracy for three different scoring functions. Bloom histogram-based caching reduced communication overhead more for eBay and PeerTrust, than for EWMA.

4.3 High Availability

To test the availability of the TMS, we simulated the same workload from Section 4.2 with the addition of random TMS node crashes. In each simulation, we varied the degree of replication as described in Section 2.7. As in our earlier simulations, the number of TMS nodes was ten. A TMS invocation (i.e., `report()` or `evaluate()`) is considered a success if at least one replica responds. Otherwise, if no replicas are available, then the invocation is considered a failure. Our results appear in Table 2 where K is
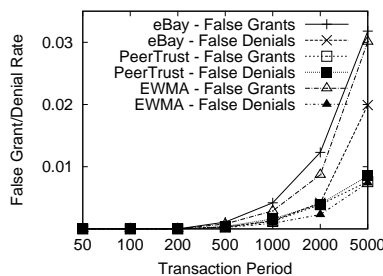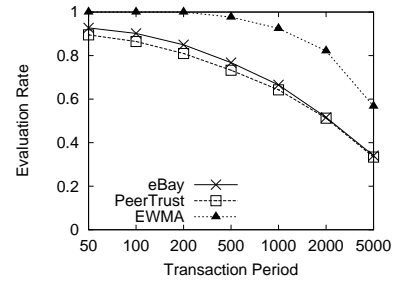

Figure 7: False Grant and Denial Rates

the degree of replication, $Prob$ is the probability of a TMS node crashing, $Crashes$ is the number of actual TMS node crashes during the simulation, $Failures$ is the number of failed TMS invocations, and $Records$ is the total number of TMS records maintained in the system.

As shown in Table 2, even a small degree of replication can tolerate a substantial number of TMS nodes crashing. For example, a single replica can handle up to 2 out of 10 TMS nodes crashing without any failures. Two replicas can handle up to 5 out of 10 TMS nodes crashing without any failures. As expected, the number of records that must be maintained increases as the degree of replication increases.

5. RELATED WORK

Much previous research has been done on reputation management systems in applications ranging from online auctions to Web service selection to peer-to-peer networks. eBay is one of the best known examples of a reputation management system for an online auction site [1]. eBay allows buyers and sellers to rate each other after each transaction as we described in Section 3.3.1. The eBay scoring function is vulnerable to strategic behavior as pointed out in [21].

Reputation management systems have also been developed for the problem of clients wanting to select the most reputable Web services. Zeng et al. proposed a Web service quality model where reputation is one of their five quality attributes considered [23]. *Verity*, which measures the consistency in service providers to deliver the QoS level specified in their contracts, has been proposed as a metric to evaluate the reputation of Web services [14]. The robustness of reputation systems for Web service selection has also been considered [17]. Unlike existing work on Web service selection, our work looks at reputation from the perspective of services that want to avoid granting requests to untrustworthy clients rather than helping clients select the most reputable Web services.

899

The prevalence of bogus content being disseminated in peer-to-peer file sharing applications has also led to much research on reputation management systems for peer-to-peer networks. XRep runs a polling protocol that allows peers to judge the reputations of resources as well as resource providers [8]. EigenTrust relies on the notion of transitive trust [15]. PeerTrust includes a trust metric that considers five features described in Section 3.3.2 [21]. Unlike a peer-to-peer network running a single application, an infrastructure hosting many services will potentially run many different applications with each one having its own criteria for trust.

6. CONCLUSION

Our reputation-based trust management framework supports the synthesis of trust-related feedback from multiple services hosted within an infrastructure while still providing the flexibility for each service to apply its own reputation scoring function. Rather than assuming a single global trust metric like many existing reputation systems, we allow each service to use its own trust metrics to meet its local trust requirements. Since our framework supports multiple reputation scoring functions, our trust management service complements existing work on reputation management systems. We have evaluated our approach in both LAN and WAN environments with a realistic application. We also compared different scoring functions within our framework. Our results indicate that different scoring functions can be effectively supported within our framework with little additional performance overhead and high availability.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] ebay. In *http://www.ebay.com.*

[2] Google app engine. In *http://code.google.com/appengine/.*

[3] Ibm application hosting. In *http://www-935.ibm.com/services/us/index.wss/offerfamily/ebhs/a1000394.*

[4] Planetlab. In *http://www.planet-lab.org.*

[5] Sample applications working group of the web services interoperability organization. In *http://www.ws-i.org/deliverables/workinggroup.aspx?wg=sampleapps.*

[6] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Conference on Information and Knowledge Management*, 2001.

[7] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), July 1970.

[8] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *ACM Conference on Computer and Communications Security*, 2002.

[9] J. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, 2002.

[10] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *SIGCOMM*, 1998.

[11] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *International Workshop on Peer-to-Peer Systems*, 2003.

[12] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. In *International Workshop on Network and Operating Systems Support for Audio and Video*, 2003.

[13] R. Iyer. Application-aware reliability and security: The trusted illiac approach. In *IEEE International Symposium on Network Computing and Applications*, 2006.

[14] S. Kalepu, S. Krishnaswamy, and S. Loke. Reputation = f(user ranking, compliance, verity). In *International Conference on Web Services*, 2004.

[15] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *International World Wide Web Conference*, 2003.

[16] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hasing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, 1997.

[17] S. Park, L. Liu, C. Pu, M. Srivatsa, and J. Zhang. Resilient trust management for web service integration. In *International Conference on Web Services*, 2005.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *International Middleware Conference*, 2001.

[19] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1), February 2003.

[20] W. Wang, H. Jiang, H. Lu, and J. Yu. Bloom histogram: Path selectivity estimation for xml data with updates. In *International Conference on Very Large Data Bases*, 2004.

[21] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7), July 2004.

[22] G. Zacharia, A. Moukas, and P. Maes. Collaborative reputation mechanism in electronic marketplaces. In *Hawaii International Conference on System Sciences*, 1999.

[23] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality driven web services composition. In *International World Wide Web Conference*, 2003.

[24] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004.