Trust Modeling for Peer-to-Peer based Computing Systems

Farag Azzedin
University of Manitoba and TR*Labs*
Winnipeg, MB R3T 2N2
Canada
fazzedin@cs.umanitoba.ca

Muthucumaru Maheswaran
McGill University
Montreal, PQ H3A 2A7
Canada
maheswar@cs.mcgill.ca

Abstract

The peer-to-peer approach to design large-scale systems has significant benefits including scalability, low cost of ownership, robustness, and ability to provide site autonomy. However, this approach has several drawbacks as well including trust issues and lack of coordination and control among the peers. In this paper, we present a trust model for a peer-to-peer structured large-scale network computing system and completely define the trust model and describe the schemes used in it. Central to the model is the idea of maintaining a recommneder network that can be used to obtain references about a target domain. Simulation results indicate that the trust model is capable of building and maintaning trust and also identifying the bad domains.

1. Introduction

Network Computing (NC) systems can be considered as a set of interconnected domains interacting in a peer-to-peer fashion. One goal of such systems is to encourage domain-to-domain interactions and increase the confidence of the domains to share their resources (a) without losing control over their own resources, and (b) ensure confidentiality for other domains. Sharing resources across institutional boundaries creates several issues related to *quality of service* (QoS) and trust. Handling these issues are complicated in NC systems due to distributed ownership, site autonomy, resource provider heterogeneity, and diverse resource clients.

An entity that is part of a large scale NC system will have the privilege of using pools of resources that would not be available to it otherwise. Unfortunately, the idea of having a virtual network framework is not appealing to some entities because of the risk associated with the notion of "sharing" resources or services. Because of the sensitivity and the vitality of data or information, such entities prefer to use their own "closed box" resources. This is not just costly for the individual entities but also an inefficient way to utilize resources.

In a NC environment, organizations are primarily concerned with trust. There are different types of trust an organization might be concerned about: (a) identity trust which focuses on verifying the authenticity of an entity and determining the authorizations that the entity is entitled to access and is based on techniques including encryption, data hiding, digital signatures, authentication protocols, and access control methods, and (b) behavior trust [2] which deals with a wider notion of an entity's "trustworthiness". A malicious web server could accept to host Web document replicas but deliver modified versions to the user or refuse requests directed to these replicas [9]. A digitally signed certificate does not convey if the issuer is an industrial spy and a digitally signed code does not convey if the code is written by competent programmers [1].

2. Concepts and Definitions

There is a lack of consensus in the literature on the definition of behavior trust and on what constitutes behavior trust management [8, 6, 1]. The definition we use in this paper is as follows:

> *Trust is the firm belief in the competence of an entity to act as expected such that this firm belief is not a fixed value associated with the entity but rather it is subject to the entity's behavior and applies only within a specific context at a given time.*

That is, the *firm belief* is a dynamic value and spans over a set of values ranging from *very trustworthy* to *very untrustworthy* as illustrated in Table 1. The *trust level* (TL) is built on past experiences and is given for a specific context. For example, entity y might trust entity x to use its storage resources but not to execute programs using these resources. The TL is specified for a given time frame because the TL today between two entities is not necessarily the same TL a year ago.

When making trust-based decisions, entities can rely on others for information pertaining to a specific entity. For example, if entity x wants to make a decision of whether to have a transaction with entity y, which is unknown to x, x can rely on the reputation of y. The definition of reputation used in this paper is as follows:

> *The reputation of an entity is an expectation of its behavior based on other entities' observations or the collective information about the entity's past behavior within a specific context at a given time.*

Seeking the reputation of a specific entity, entity x relies on information from a set of other entities referred to as *recommenders' set* (R). A recommender is an entity that gives recommendation using its *direct trust table* (DTT) that includes trust values for entities with which the recommender had prior direct transactions. Recommenders might have different criteria for evaluating other entities. Hence, different recommenders might give different recommendation about entity y. Therefore, Entity x associates an *accuracy* measure with each recommender in the recommender set R. The information (i.e. the *accuracy* measure) on the set of entites that act as recommenders being used by x is kept in a *recommender trust table* (RTT). Entity x uses the *accuracy* measure to minimize the deviation between the information received from each recommender and the actual "trustworthiness" of y. The definition of accuracy used in this paper is as follows:

> *A recommender is said to be accurate, if the deviation between the information received from it pertaining to the"trustworthiness" of a given entity y in a specific context at a given time and the actual trustworthiness of y within the same context and time is less than a precision threshold.*

3. Computing Accuracy, Trust, and Reputation

Let the accuracy of recommender z as observed by entity x for a specific context c at a given time t be denoted as $A(x, z, t, c)$. Let $RE_x(z, y, t, c)$ denote the recommendation for entity y given by z to entity x at time t for context c and $TTL_x(y, t, c)$ denote the *true trust level* (TTL) of y obtained by x as a result of monitoring its transaction with y for context c and at time t. Entity x can monitor the transaction using *audit data* [7] generated by the operating system or post-mortem analysis tool such as *Intrusion Detection systems* (IDSs) [10]. Let Δ_{RE} denote the difference and is given by

$$\Delta_{RE} \quad = \quad |RE_x(z, y, t, c) - TTL_x(y, t, c)| \qquad (1)$$

The value of Δ_{RE} will be an integer value ranging from 0 to 4 since $RE_x(z, y, t, c)$ and $TTL_x(y, t, c)$ are TLs. Then, $A(x, z, t, c)$ can be computed as:

$$A(x, z, t, c) = -\frac{1}{4}\,\Delta_{RE} + 1 \qquad (2)$$

Notice that A is a real number in the interval $[0, 1]$ and if there is no difference (i.e. $\Delta_{RE} = 0$), then $A = 1$ meaning that z has a maximum accuracy as far as x is concerned. Inversely, for a maximum difference (i.e. $\Delta_{RE} = 4$), $A = 0$ meaning that z is completely inaccurate as far as x is concerned. Before x can use the recommendation given by z (i.e. $RE_x(z, y, t, c)$) to calculate the reputation of y, $RE_x(z, y, t, c)$ will be adjusted to reflect recommender z's accuracy. Hence a *shift function* (S) that applies Δ_{RE} to $RE_x(z, y, t, c)$ is given by

$$S(\Delta_{RE}, RE_x(z, y, t, c)) = \left\{ \begin{array}{l} RE_x(z, y, t, c) + \Delta_{RE} \\ \qquad \text{if } RE_x(z, y, t, c) + \Delta_{RE} \leq 5 \\ \textit{or} \\ |RE_x(z, y, t, c) - \Delta_{RE}| \\ \qquad \text{if } RE_x(z, y, t, c) + \Delta_{RE} \geq 1 \end{array} \right. \qquad (3)$$

It should be noted that if S is: a) greater than 5, it is set to 5, and b) less than 1, it is set to 1.

In computing trust and reputation, several issues have to be considered. First, the trust may decay with time. For example, if x trusts y at level p based on past experience five years ago, the trust level today is very likely to be lower unless they have interacted since then. Similar time-based decay also applies for reputation. Second, entities may form alliances and as a result would tend to trust their allies more than they would trust others. Finally, the TL that x holds about y is based on x's direct relationship with y as well as the reputation of y, i.e., the trust model should compute the eventual trust based on a combination of direct trust and reputation and should be able to weigh the two components differently.

Let x and y denote two entities. The trust relationship for a specific context c at a given time t between the two entities, expressed as $\Gamma(x, y, t, c)$, is computed based on the direct relationship for the context c at time t between x and

Table 1. Description of the different trust levels.

Trust Level (TL)	Equivalent Numerical Value	Description
A	1	very low trust level
B	2	low trust level
C	3	medium trust level
D	4	high trust level
E	5	very high trust level

y, expressed as $\Theta(x, y, t, c)$, as well as the reputation of y for context c at time t expressed as $\Omega(y, t, c)$. Let the weights given to direct and reputation relationships be α and β, respectively such that $\alpha + \beta = 1$ and $\alpha, \beta \geq 0$. The trust relationship is a function of direct trust and reputation. If the "trustworthiness" of y, as far as x is concerned, is based more on direct relationship with x than the reputation of y, α will be larger than β.

The direct relationship is computed as a product of the TL in the DTT and the *decay function* ($\Upsilon(t - t_{x-y}, c)$), where c is the context, t is the current time, and t_{x-y} is the time of the last transaction between x and y. The reputation of y is computed as the average of the product of $RE_x(z, y, t, c)$ shifted by S and the *decay function* ($\Upsilon(t - t_{z-y}, c)$), for all n recommenders in R (i.e. $z_i \in R$, $\forall(\ 1 \leq\ i\ \leq n)$). In practical systems, entities will use the same information to evaluate direct relationships and give recommendations, i.e., DTT will be used to give recommendations as well as for obtaining the direct TL.

$$\Gamma(x, y, t, c) = \alpha \times \Theta(x, y, t, c) + \beta \times \Omega(y, t, c) \qquad (4)$$

$$\Theta(x, y, t, c) = DTT(x, y, t, c) \times \Upsilon(t - t_{x-y}, c) \qquad (5)$$

And $z\ \neq\ x\ , and\, z\ \neq\ y$, we have:

$$\Omega(y, t, c) = \frac{1}{n} \times \sum_{i=1}^{n} S(\Delta_{RE_i}, RE_x(z_i, y, t, c)) \times \Upsilon(t - t_{z_i-y}, c) \qquad (6)$$

4. The Trust Model

Figure 1 shows the overall trust model in which the NC system is divided into *domains* (Ds). We associate two virtual domains with each D, namely a *resource domain* (RC) to signify the resources within the D and a *client domain* (CD) to signify the clients within the D. Trust agents (TAs) exist in each D with mechanisms to: (a) update the Ds' trust tables, (b) allow entities to join Ds and inherit their trust attributes, and (c) apply a decay function to reflect the decay of trust between Ds. Each D has two data structures, namely DTT and RTT. The DTT and RTT are updated by the TA. The DTT is updated using the trust values observed by the TA based on the direct transactions with other Ds. The RTT is updated by monitoring the accuracy of recommendations given about target Ds.

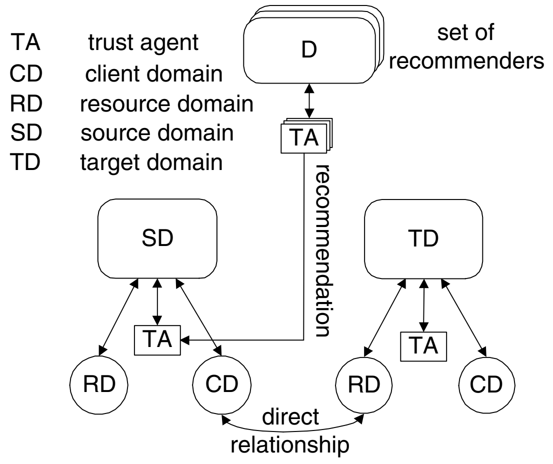

Figure 1. The overall trust management model.

Assume that a RD in a *source domain* (SD) wants to engage in a trust relationship with a CD associated with a *target domain* (TD). The SD gathers information to build its direct relationship (i.e. $\Theta(SD, TD, t, c)$) by obtaining its direct relationship TL of TD from its DTT which is internal to the SD. The SD can also seek to know the reputation of TD by asking its R. Figure 3 shows a *recommendation network* that might exist in a trust relationship. Each member $z\ \in\ R$, provides recommendations based on its DTT.

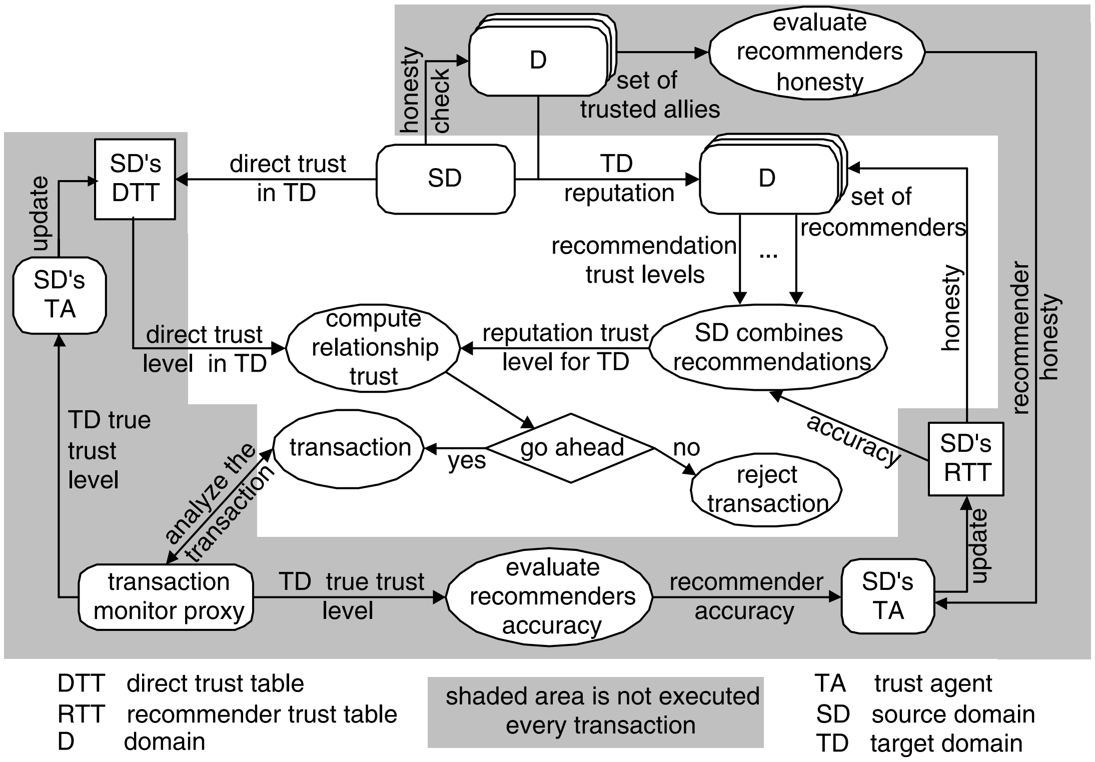

Figure 2. Trust development cycle.

If TD is unknown to recommender z, then z will ask its R. It can be observed that it is very likely the *recommender network* will contain cycles. To solve this problem, the list of the visited Ds is kept attached to every recommendation request. Therefore, if TD is unknown to z and z's R have been already visited by the recommendation request, then z will return U as its recommendation of TD.

For the trust relationship with TD, SD used two sources of information as shown in Figure 2: a) the direct trust relationship with TD obtained from SD's DTT, and b) the reputation trust relationship of TD obtained from SD's R. These two sources of information need to be evaluated and updated if necessary. To update the DTT, a running average of $DTT(SD, TD, t, c)$ can be kept or a simple formula such as the following can be used.

$$DTT(SD,TD,t,c) = (1-\delta) \times \\ DTT(SD,TD,t_{SD-TD},c) + (\delta) \times \\ TTL(TD,c,t) \tag{7}$$

where $0 \leq \delta \geq 1$. If $\delta > 0.5$, more preference is given to TTL resulting from the current trust relationship between the two domains.

To evaluate the set of recommenders (R), SD needs to compute the *accuracy* measures as illustrated in Section 3 and shown in Figure 2. Initially, all recommenders in R will have 1.0 as their A value meaning that all recommenders in R are `accurate`. Suppose that $A(SD, z, c, t)$ is the accuracy of recommender z based on the recommendation z gave for the current transaction to SD and $A_{RTT}(SD, z)$ is the accuracy of recommender z maintained at SD's RTT based on all up to the last recommendation obtained from it. The following simple formula keeps a running average of the accuracy measure.

$$A_{RTT}(SD,z) = (1-\delta) \times \\ A_{RTT}(SD,z) + \delta \times A(SD,z,c,t) \tag{8}$$

The trust between two domains SD and TD is specified for a given context c and time t. In our trust model, a *decay function* is introduced to reflect this drop when modeling the trust between the domains as explained in Section 2. Different domains can have different decay functions and additionally can consider other factors that accelerate or decelerate the trust level decay. For example, domains that belong to the same country or organization might employ

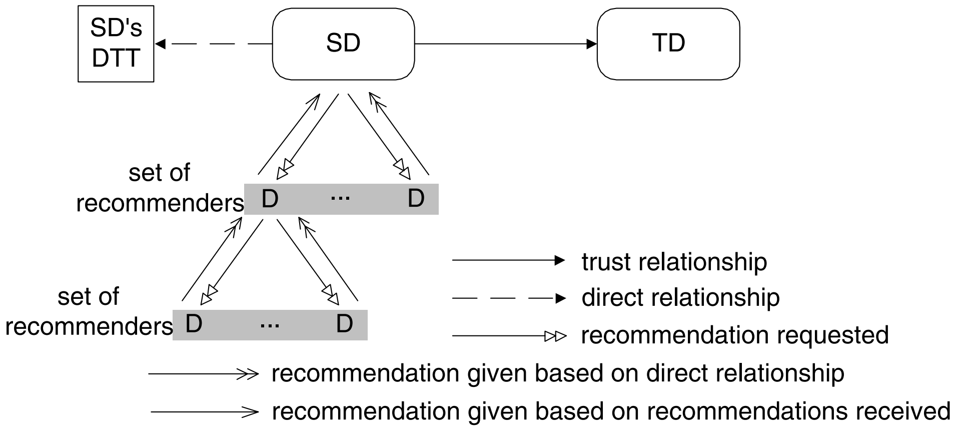

Figure 3. An example of a recommendation network existing in a trust relationship.

shallow decay functions compared to domains that are totally unfamiliar to unrelated to each other. Although decay functions are supported by our model, the study presented in this paper does not employ trust decay.

4.1 Characteristics of the Trust Architecture

The trust architecture determines how the trust model is applied to a NC system. By design, trust modeling is necessary and appropriate only for large-scale NC systems. Therefore, the trust architecture should deal with key issues such as heterogeneity, site autonomy, and scalability that are associated with large-scale network computing systems.

A straight forward approach to mapping the proposed trust model to a large-scale NC system can result in an inefficient implementation. The trust architecture presented in this section employs various techniques to address these issues. First, the NC system is divided into domains called the NCDs. The resources (RD) and clients (CD) inherit the parameters associated with the NCD. Aggregating the resources and clients into groups increases the scalability of the overall approach. Second, we assume the trust to be a slow varying attribute. If, on the contrary, trust varies much faster, then the overall system is in an unpredictable state and the deployment of a trust model is not viable. If the slow variation assumption is true, it means the trust level is updated based on a significant amount of transactional data and the overhead associated with maintaining the trust model can be amortized over a large number of transactions. Third, by limiting the number of contexts, the fragmentation of the trust space is reduced. In the example model considered in this paper, the contexts are limited to primary service types such as printing, storage, and computing.

5. Performance Evaluation

The goal of the evaluation is to use the *success ratio* metric to examine: a) the effectiveness of the proposed trust management architecture model using consistent and inconsistent DTT to show how reliable is the trust learnt by the model, and b) the responsiveness of the proposed trust management architecture model. Responsiveness measures the how quickly does the model identify and isolate “bad” domains. In these simulations. the trust architecture was abstracted to keep the complexity manageable and at the same time provide sufficient detail as explained below. One measure of performance of the trust architecture is its ability to correctly predict the trust that exists between two entities. We quantify this by determining the *success ratio* (SR) of prediction that is computed as follows:

$$SR(t) = \frac{100}{n} \times \sum_{k=1}^{n} \left[D_k^g(detected = 1) + D_k^b(detected = 1)\right] \quad (9)$$

$D_k^g(detected = 1)$ means that D_k^g is a good domain and it is detected as a good domain. In a similar fashion, $D_k^b(detected = 1)$ means that D_k^b is a bad domain and it is detected as a bad domain.

5.1 Simulation Model and Setup

In the simulation model, the physical system that consists of a collection of domains that peer with each other is represented by a collection of peering simulation entities. The transactions that originate and terminate at the resources within domains are modeled as happening between the domains. This aggregation does not introduce any errors because in the trust model, the domains are assumed to represent the client and resource sides.

One of the underlying assumptions of this study is that naturally there exists a network of trust relationships among different domains. The objective of trust modeling is to discover these trust relationships via observations of ongoing transactions. We model the natural trust relationships that exist among the domains by an *Actual Direct Trust Table* (ADTT). This table contains the absolute true trust exists among different domains. This study makes a simplification assumption that models these trust relationships as constant for the duration of the simulation time.

In the actual system, trust monitoring proxies periodically examine the transactions among the domains to determine the trust level. This is referred to as the TTL because through outside observation this is the closest we can get to the *actual* trust. These TTLs are used to update a *computed direct trust table* (CDTT). The CDTT is initially created by adding a random "noise" generated from [0,4] to the ADTT. The random noise makes the CDTT totally uncorrelated with ADTT. The unravelment of the TTLs with the trust monitoring process is simulated by updating the CDTT with values that are closer to the ADTT (i.e., we simulate the elicitation of the true trust value by adding a noise randomly generated from [0,2] to the ADTT). The trust levels resulting from the transactions within domains are modeled by equation 4 and used to update a *predicted direct trust table* (PDTT). The PDTT is initially created by setting its values to U (i.e. -1) meaning that all domains are unknown to each other. All the simulation results are based on the PDTT.

The domains' transactions process was simulated using a discrete event simulator with the requests arrivals modeled using a Poisson random process. The number of domains used in the simulation is set to 30 and the number of transactions between the domains is set to 20000. The size of R is fixed for all domains and is set to 4 whereas the size of T is fixed for all domains and is set to 3.

The actual TLs that the different domains have in each other are assumed and kept in the ADTT. The size of the ADTT is 30×30 and its TLs were randomly generated from [1, 5] representing trust levels A to E, respectively. The TLs along a column of the ADTT indicate the direct TL that domains have in a particular domain and we can refer to the average variation along a column as *trust-in heterogeneity*. The *trust-in heterogeneity* indicates the variation of other domains' TLs in a particular domain. If this variation is minimal, the trust model is said to be a *consistent* trust model, otherwise it is said to be an *inconsistent* trust model.

5.2. Results and Discussion

Table 2 shows the *success ratio* of the trust management architecture when using a consistent DTT. Let us discuss the situation when the post-mortem analysis is done every transaction (i.e. $pm = 1$). When $\alpha = 1.0$ the *success ratio* is in the range of 96.2% to 99.5% and it can be noted that this *success ratio* is not affected by the increase of "malicious" domains. When $\alpha = 1.0$, each domain depends on its CDTT to come up with how trustworthy other domains are and recommendations are totally ignored. Since the post-mortem analysis is done every transaction, the CDTT will converge very fast to the ADTT (i.e. the CDTT will have the actual TL values). On the other hand, When $\alpha = 0.5$ the *success ratio* is in the range of 82.6% to 100% and it can be noted that this *success ratio* is affected by the increase of "malicious" domains because reputation is given the same weight as the direct trust. The effect of the "malicious" domains will become even greater when domains rely totally on recommendations and is shown when $\alpha = 0.0$; the *success ratio* drops to 51%. Similar arguments can be observed when post-mortem analysis is performed every third or sixth transaction.

Table 3 shows the *success ratio* is around 50% when using an inconsistent DTT. This shows that a domain is not learning the actual TLs for other domains. That is, 50% D_i identifies a particular domain as "malicious" and 50% as "honest" which means that D_i is totally confused. When there are few (i.e. 0 to 5) "malicious" domains and $\alpha \neq 1.0$, the *success ratio* is higher than 50% especially when $\alpha = 0.0$. This is because: a) each domain is asking 4 recommenders and combining these 4 recommendations to come up with how trustworthy are other domains, b) since there are few "malicious" domains, these recommenders will be all or mostly "honest", and c) since the DTT is inconsistent, computing the reputation of the target domain will lead to a TL that is close to the actual trustworthiness of the target domain.

Table 4 shows the responsiveness of the trust model when there are zero "malicious" domains. When the post-mortem analysis is done every transaction, it can be observed from Table 4 that the more the model relies on the reputation, the faster is the *success ratio*. For example, when $\alpha = 1.0$, the *success ratio* reached 99.1% in $15,000$ transactions. On the other hand, when $\alpha = 0.0$, the *success ratio* reached 99.4% in just $5,000$ transactions. Similar arguments can be observed when post-mortem analysis is performed every third or sixth transaction. But from Table 5, it

Table 2. Success ratio using a consistent trust model.

Frequency of the post-mortem analysis	α value	Number of bad domains out of 30 domains					
		0	5	10	15	20	25
1	1.0	99.5%	98.6%	98.5%	97.7%	97.1%	96.2%
	0.5	100%	99.4%	97.7%	94.5%	87.2%	82.6%
	0.0	100%	98.2%	94.7%	83.2%	58.6%	51%
3	1.0	95.2%	89.9%	85.3%	81.5%	78.9%	73.3%
	0.5	99.9%	94.4%	85.4%	75.9%	65.3%	56.9%
	0.0	100%	91.6%	77.6%	60.3%	38.5%	20%
6	1.0	87.9%	80.7%	76.6%	70.3%	66.8%	61.3%
	0.5	97.2%	87.1%	75.9%	65.2%	54.8%	42.9%
	0.0	99.9%	84.4%	68.5%	51.8%	34.1%	16.8%

Table 3. Success ratio using an inconsistent trust model.

Frequency of post-mortem analysis	α value	Number of bad domains out of 30 domains					
		0	5	10	15	20	25
1	1.0	66.6%	58.2%	54.3%	49.2%	42.9%	44.4%
	0.5	76.4%	68.2%	58.6%	49.3%	38.0%	33.7%
	0.0	93.9%	80.9%	64.6%	50.3%	34.3%	17.2%
3	1.0	67.1%	61.6%	54.1%	48.7%	43.4%	38.9%
	0.5	81.8%	71.9%	60.2%	50.2%	38.4%	28.6%
	0.0	96.9%	82.2%	66.2%	50.6%	33.9%	16.7%
6	1.0	67.8%	61.3%	53.3%	50.1%	44.3%	36.9%
	0.5	84.3%	73.4%	59.9%	50.5%	37.7%	27.5%
	0.0	98.2%	83.1%	66.2%	50.1%	33.4%	16.7%

can be observed that the more the model relies on the reputation, the slower is the *success ratio* and this becomes even more apparent in Table 6.

6. Related Work

In this section, we examine several papers that examine issues that are peripherally related. *Poblano* [3] is a decentralized trust model implemented in a peer-to-peer fashion for the Project JXTA [5]. This model can be used as guidelines for searching by developing reputation among the peers and potentially creates “Webs of trust”. A model for supporting trust based on experience and reputation is proposed in [1]. This trust-based model allows entities to decide which other entities are trustworthy and also allows entities to tune their understanding of another entity’s recommendations. A survey of trust in Internet applications is presented in [6] and as part of this work a policy specification language called Ponder [4] was developed. Ponder can be used to define authorization and security management policies. Ponder is being extended to allow for more abstract and potentially complex trust relationships between entities across organizational domains.

7. Conclusions and Future Work

Network computing systems are being positioned as a computing infrastructure that will enable pools of resources to be shared across institutional boundaries. Unfortunately, the notion of “sharing” poses some concerns such as privacy, confidentiality, and autonomy. Hence, “trust” should be addressed in such a distributed environment. We view

Table 4. For zero bad domains out of 30 domains: comparison of success ratio progress for different post-mortem intervals and α values.

pm analysis interval	α value	Number of iterations							
		1000	2000	3000	4000	5000	10000	15000	20000
1	1.0	49.3%	68.3%	80.6%	86.7%	90.9%	97.8%	99.1%	99.5%
	0.5	61.1%	81.6%	92.4%	97.1%	98.9%	100%	100 %	100%
	0.0	63.6%	85.6%	94.3%	98.6%	99.4%	100%	100 %	100%
3	1.0	44.1%	59.8%	68.0%	72.9%	76.4%	87.1%	92.1%	95.3%
	0.5	57.5%	78.7%	86.4%	90.5%	92.6%	98.5%	99.8%	99.9%
	0.0	61.5%	83.6%	91.7%	95.3%	97.6%	99.9%	100 %	100%
6	1.0	42.5%	58.3%	65.3%	66.9%	68.9%	77.6%	82.9%	87.9%
	0.5	56.7%	76.3%	82.3%	84.7%	86.2%	91.9%	95.3%	97.2%
	0.0	61.1%	83.8%	91.6%	93.4%	95.3%	99.2%	99.8%	99.9%

Table 5. For 15 bad domains out of 30 domains: comparison of success ratio progress for different post-mortem intervals and α values.

pm analysis interval	α value	Number of iterations							
		1000	2000	3000	4000	5000	10000	15000	20000
1	1.0	55.2%	59.1%	64.6%	68.5%	73.8%	88.9%	95.1%	97.7%
	0.5	55.7%	59.5%	63.2%	65.9%	69.8%	85.3%	92.8%	94.5%
	0.0	53.8%	52.9%	54.9%	56.6%	57.5%	71.6%	80.3%	83.2
3	1.0	52.9%	52.2%	54.6%	57.9%	60.5%	70.8%	77.7%	81.5%
	0.5	53.3%	52.1%	53.8%	55.9%	57.5%	65.2%	70%	75.9%
	0.0	53.3%	50.6%	50.3%	51%	51.1%	52.8%	55.6%	60.3
6	1.0	49.4%	49.6%	51.7%	53.4%	54.4%	60.3%	65.9%	70.3%
	0.5	49.5%	48.4%	50.8%	52.3%	53.1%	57.4%	61.4%	65.2%
	0.0	49.9%	48.7%	50.8%	50 %	50.6%	50%	51.1%	51.8%

trust in two steps: (a) *identity trust* which is verifying the identity of an entity and what that identity is authorized to do, and (b) *behavior trust* which is monitoring and managing the behavior of the entity. Identity trust has been addressed by techniques such as encryption, data hiding, digital signatures, and access control. We proposed a trust management architecture that can evolve and maintain the behavior trust based on direct as well as reputation trust relationships. Simulation experiments were deducted and showed that: a) the trust model can not learn from inconsistent DTT, b) if the trust model relies entirely on the direct trust ($\alpha = 1.0$), then it will take a long time to identify the bad domains, and c) if the trust model relies on both direct and reputation, identifying the bad domains will be faster. However, the trust model will become sensitive to theses bad domains. For the trust management architecture to be of any practical use, more simulation work should be carried to investigate: a) investigating the trade-offs of performing the post-mortem analysis and the overall overhead it imposes on the trust management architecture, and b) implementing the "honesty" measure to filter out the malicious recommenders from the recommenders' set. Each domain will maintain a *trusted allies* set to check the honesty of its recommenders' set. As illustrated in Figure 2, SD can poll

Table 6. For 20 bad domains out of 30 domains: comparison of success ratio progress for different post-mortem intervals and α values.

pm analysis interval	α value	Number of iterations							
		1000	2000	3000	4000	5000	10000	15000	20000
1	1.0	57%	57%	61.5%	65.3%	70.1%	87.2%	93.9%	97.1%
	0.5	54.1%	50 %	52.8%	55.2%	59.1%	75.2%	83.6%	87.2%
	0.0	49.1%	40.7%	38.2%	38%	38.2%	46.8%	53.3%	58.6
3	1.0	53.3%	50.9%	50.5%	52.8%	53.7%	64.6%	73.2%	78.9%
	0.5	46.7%	42.9%	42.3%	44.1%	46.3%	53.9%	59.3%	65.3%
	0.0	44.7%	37.8%	34 %	34.5%	34.1%	34.7%	36.2%	38.5
6	1.0	54.1%	49.9%	50.6%	51.4%	51.8%	57.6%	61.8%	66.8%
	0.5	46.4%	42.8%	42.1%	42.4%	43.1%	48 %	51.7%	54.8%
	0.0	45.1%	37.8%	35.7%	35.4%	35.4%	34.6%	34.1%	34.1%

its recommender z for recommendation about TD. At the same time and for the same context, SD polls its *trusted allies* set to ask recommender z for recommendation about TD. If recommender z is "honest", it will give a consistent information. Otherwise, z is considered to be "malicious" and it can be filtered from SD's recommenders' set.

References

[1] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," *Hawaii Int'l Conference on System Sciences*, Jan. 2000.

[2] F. Azzedin and M. Maheswaran, "Integrating trust into Grid resource management systems," *2002 International Conference on Parallel Processing (ICPP '02)*, Aug. 2002, pp. 47–54.

[3] R. Chen and W. Yeager, "Poblano: A distributed trust model for peer-to-peer networks." "htpp:security.jxta.org", 2001.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder policy specification language," *Workshop on Policies for Distributed Systems and Networks*, Jan. 2001.

[5] L. Gong, "JXTA: A network programming environment," *IEEE Internet Computing*, Vol. 5, No. 3, 2001, pp. 88–95.

[6] T. Grandison and M. Sloman, "A survey of trust in Internet applications," *IEEE Communications Surveys & Tutorials*, Vol. 4, No. 4, Fourth Quarter 2000, pp. 2–16.

[7] N. Habra, B. L. Chalier, A. Mounji, and I. Mathieu, "ASAX: Software architecture and rule-based language for universal audit trail analysis," *European Symposium on Research in Computer Security (ESORIC'92)*, Nov. 1992, pp. 435–450.

[8] B. Misztal, "Trust in modern societies," Polity Press, Cambridge MA, 1996.

[9] G. Pierre and M. van Steen, "A trust model for peer-to-peer content distribution networks." "www.cs.vu.nl/ gpierre/publi/TMPTPCDN_draft.php3", Nov. 2001.

[10] S. E. Smaha and J. Winslow, "Misuse detection tools," *Journal of Computer Security*, Vol. 10, No. 1, Jan. 1994, pp. 39–49.

Biographies

Farag Azzedin is a Ph.D. candidate at the Computer Science Department at the University of Manitoba, Canada. From January 1986 to August 1991, he was attending the University of Victoria, Canada where he received a BSc degree in Computer Science. From 1991 to the end of 1998 he worked with the Ministry of Health, B.C. Canada and the city of Vancouver, B.C. Canada as a computer programmer/analyst and a data analyst, respectively. He received an MS degree in computer science in 2001 from the University of Manitoba, Canada. His research is supported by a fellowship from the University of Manitoba as well as a fellowship from TRLabs, a Canadian research consortium in information and communications technology. His research interests include Grid computing, trust modeling and its application in peer-to-peer computing systems, and resource management in distributed systems. He has coauthored more than 10 technical papers in these and related areas.

Muthucumaru Maheswaran is a joint assistant professor in the School of Computer Science and the Department of Electrical and Computer Engineering at McGill University, Canada. From August 1998 to December 2002, he was an assistant professor in the Department of Computer Science at the University of Manitoba, Canada. In 1990, he received a BSc degree in electrical and electronic engineering from the University of Peradeniya, Sri Lanka. He received an MS degree in electrical engineering in 1994 and a PhD degree in electrical and computer engineering in 1998, both from the School of Electrical and Computer Engineering at Purdue University. He held a Fulbright scholarship during his tenure as an MSEE student at Purdue University. His research interests include autonomic computing, Grid computing, peer-to-peer computing, trust modeling and management in large-scale networked systems, and scalable resource management systems. He has authored or coauthored more than 50 technical papers in these and related areas.