UNIVERSITY of MANITOBA

TRUST MODELING AND ITS APPLICATIONS FOR PEER-TO-PEER BASED SYSTEMS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Farag A. Azzedin

August 2004

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

TRUST MODELING AND ITS APPLICATIONS FOR PEER-TO-PEER BASED SYSTEMS

BY

Farag A. Azzedin

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

DOCTOR OF PHILOSOPHY

Acknowledgments

In the name of Allah, the Merciful, the Compassionate. Praise be to Him for His creation and making me submissive to Him. Thanks to Him that He sent us the prophets to guide us to the straight path.

I express my gratefulness to my supervisor Prof. M. Maheswaran, whose amazing patience, infinite help in all aspects, and non-stop support made me achieve what I did not know I had in me. I can never thank him enough. I am also thankful to the thesis committee members, Dr. J. Diamond, Prof. N. Arnason and Prof. P. Thulasiraman for giving me valuable suggestions, comments, and for evaluating this work. Also, I am very appreciative to TRLabs and the University of Manitoba whose financial support made this thesis possible.

All the love is to my mother (Fatima) and my father (Ahmed). They are always in my heart and mind, and whose my accomplishment in life is none but the result of Allah's answering to their prayers and supplication to Him for my success.

Finally, I express my heartfelt gratitude to my wife (Ameena), whom whatever words of thanks I say, they would not be enough to do justice to her. And to our children Muhammad, Ahmed, Hala, and Fatima whom just remembering them makes me know what I want to do in life.

Abstract

Organizing large-scale network computing systems in a *peer-to-peer* (P2P) fashion is a manifestation of one of the fundamental design principles on the Internet. Current research is focusing on improving P2P systems and one of the future directions is to combine P2P and Grid technologies. One of the key issues identified in the evolution of P2P technologies is the trust issue.

This thesis presents a trust model for P2P structured large-scale network computing systems. The most widely used trust modeling approach is to use a network of recommenders to obtain references and use these to predict the trust between two entities. This approach is known to suffer from drawbacks such as trustworthiness of the recommenders and scalability.

To address this problem, a solution is proposed where a recommender is independently evaluated using accuracy and honesty measures. This thesis explains using simulation results how the separation of accuracy and honesty helps in addressing the above issues. To demonstrate the utility of the trust model, a trust aware resource allocation model is developed such that it can be used to make trust cognizant resource allocations. To the best of our knowledge, this is the first study to integrate trust into resource management systems. The simulation results indicate that significant preferences gain can be obtained through this integration.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Security Versus Trust

As we move towards the continued growth of computing opportunities, the relationship between security and trust is becoming the focus of many researchers and businesses [1, 2, 3]. Trust is emerging as a fundamental part of the Internet of tomorrow [4]. Currently, security mechanisms do not say anything about trust [1, 5]. For example, a hostility behavior of a code can not be determined by any level of cryptography [6]. The term *soft security*, which we refer to as trust, is used in [6, 7] to describe a "social control" model which acknowledges that malicious entities may exist among harmless ones in online communities.

Online communities are facing increasing challenges. The network information systems are vulnerable to technical failures as well as malicious attacks [8]. If customers or clients refuse to engage in online activities because they are fearful that they will be cheated, have their confidential data stolen, or overcharged for online services; then online communities will not survive [9]. Similarly, if online businesses stay away fearing costly losses from such actions as customers or clients failing to pay bills, repudiating debts or

commitments, or clients running illegal or nefarious programs; then again online communities will not survive.

Can the solution to the above concerns be provided by suites of technical security mechanisms seeking to create "trusted" or rather trustworthy systems? In other words, can we suggest that trust is provided through security? It turns out that security does not give trust [1, 5, 3]. If your neighbor is a thief, you will not leave your car unlocked. But if there is a guard watching your car, you can leave your car unlocked. You feel certain and safe because of the presence of the guard, but still you do not **trust** your neighbor. **Security** (as provided by the guard in our example) gives certainty and safety, whereas trust gives vulnerability and risk [10, 11, 12]. For example, one feels safe and certain because the enemy is behind bars but still has no trust in that enemy.

We might think that as long as certainty and safety are provided, trust seems not to be needed. In the online world, neither certainty nor safety can be achieved . As stated in [10, 3], there is no such thing as absolute online certainty and safety. Even if we achieve a reasonable level of certainty and safety, the price of certainty and safety is limitation [3, 1] because we have yet to establish a method of assuring certainty and safety without: a) limiting the range, nature, and scope of on-line interaction, b) acceptance of greater surveillance, and c) the need to make prior judgments about whom we will or will not interact with. In general, we trade freedom and range of opportunity for this certainty and safety [1, 9].

In other words, security limits choice and allows retention; whereas trust creates opportunities and allows growth [10, 3]. Security systems such as surveillance that involve close monitoring and watching are not just costly for an organization, but also can undermine trust and elicit the very behaviors they are intended to prevent [1]. I quote from [9] "if heavy regulations is capable of eradicating overtures of trust, and of driving out opportunities for trusting relationships, then it is capable of doing great harm".

In addition, online security has draw backs because: a) betrayal comes from those allowed within our spheres of safety and within our safe zones [13, 1]. For example, while firewalls are used to protect assets from external attacks, it is still the insiders (e.g., company's employees) who can accidentally or intentionally do great amount of damage, b) security does not necessarily change the attitude of the "bad guys" [9]. Online security mechanisms such as firewalls, encryption, and access control would no more prevent online hackers than prison bars and surveillance cameras could achieve offline, and c) having achieved some modes of safety and certainty through security and relying on them, we might not notice a failure until considerable damage is done [14, 15]. For example, patches are released frequently to fix **bugs** in programs that we rely on to provide security [16, 17]. That is, on-line security is not bullet-proof and by solely relying on it, we might not notice a compromise until considerable damage has been done. For example, there are regular announcements of security flaws [18] and that several security flaws have been reported since Sun Microsystems [19] announced Java. Unfortunately, anti-virus software can not help much. If a virus can infect 1.2 million computers (one estimate of Melissa infections) in the hours before a fix is released, that is a lot of damage [15]. Further, Not all users are particularly diligent about installing security patches. A case study in [14] reports that 75% of users had failed to upgrade their Apache servers months after the Apache vulnerability was known.

Trust is an attitude and involves at least a trustor and a trustee [20]. In trusting, we are acknowledging the other as a **free** agent and this is part of the exhilaration both of trusting and being trusted. Where people are guaranteed safety, where they are protected from harm, and if all other persons act under coercion, then trust is redundant [9, 1]. If there is security, there is no place for trust and trust is squeezed out of the picture. The signs that give evidence of the reasonableness of trust *must* always fall short of certainty [10]. Trust is an attitude without guarantees and without complete warranty [5, 10].

1.2 Motivation

The *peer-to-peer* (P2P) computing is one of the technologies that is having a significant impact on the way Internet-scale systems are built. It is well established for applications such as file sharing (e.g., Gnutella [21] and KaZZa[22]) and parallel distributed computation (e.g., SETI@home [23]). The popularity of P2P computing has prompted the research community to examine several aspects of it. One aspect is to extend P2P computing to host a wider variety of applications. Several approaches including the following have been investigated to achieve this goal: (a) constructing generalized P2P overlays [24], (b) using P2P overlays as resource provisioning systems for resource management infrastructures such as Grid systems [25], and (c) hybrid systems that combine P2P and Grid computing techniques [26].

One of the issues that is common to all these approaches is trust [26, 27, 28, 21, 29, 30]. The manifestation of trust as a crucial issue can be understood by closely looking at traditional P2P applications. In file sharing and parallel computations, a massive redundancy approach is followed. In this approach, the objective is to make the hosted service (in the case of file sharing, the access to files) immune to individual resource failures or misbehavior. While this approach yields robust service for applications such as file sharing, it is not suitable for sensitive applications such as hosting databases or storing medical images. One of the causes of this situation is the massive redundancy approach itself. Because of massive redundancy, a peer by itself has very little value to the P2P system in which it is a member. As a result, the peer has little incentive to contribute towards the overall goal of P2P system and usually ends up pursuing its own agenda. This has been observed in real P2P systems such as Gnutella as the *free riding* phenomena [21] and also philosophically referred to as the "tragedy of the commons" [29]. The importance of trust can be further understood by examining the following scenarios: (a) accountability on the resource side [29] and (b) resource sharing in hostile versus friendly environments [8, 10].

As P2P systems are generalized, we need to manage the resources to deliver predetermined levels of service. One of the requirements to deliver acceptable levels of service is accountability on the resource side [29], For example, a resource should be accountable for promising services and not delivering them. One way of holding a resource accountable is to maintain a trust parameter dedicated to the resource and update it accordingly.

One of the objectives of resource sharing is to increase some measure of the overall work done by the collection of resources. In a hostile environment, resource sharing requires stringent security measures to protect resources as well as the consumers. The overhead of security provisioning can negate the performance advantages targeted by resource sharing. If the expected trust levels among the different entities are known, the sharing relationships can be confined to mutually trusting entities and consequently the security provisioning overhead can be reduced.

Being part in a P2P computing system, a client has the privilege of using pools of resources or services that would not be available to it otherwise. Unfortunately, the idea of having a virtual network framework is not attractive because of the risk associated with the notion of "sharing" resources or services [8, 9, 31, 27]. Because of the sensitivity and the vitality of data or information, such clients prefer to use their own "closed box" resources. This is not just costly but also an inefficient way to utilize resources. Clients and resources (also referred to as entities) perceive such an environment as offering both opportunities and threats [10]. Factors affecting trust for entities include security risks, privacy issues, and vulnerability to harm due to malicious attacks. It is in this environment of risk and uncertainty that mechanisms must develop strategies to establish trustworthiness. Consequently, systems should aid entities in assessing the level of trust they should place on online transactions.

1.3 Trust Taxonomy

The evolution of trust is one of the most profound and irreversible changes in online systems [4]. Users of the Internet consist of parties with different motivations. This implies that mechanisms that manage interactions on the basis of trust should be a fundamental and an integral part of online transactions [9, 4]. The notion of trust is more than creating, acquiring, and distributing credentials such as identities or certificates. A peer might be identified, authenticated, and authorized, but this does not ensure that it exercises its authorizations in a way that is expected [32, 31]. In general, trust is a complex concept that has been addressed at different levels by many researchers [33, 34, 5, 35]. This thesis classifies trust into two categories (a) identity trust and (b) behavior trust. Figure 1.1 shows a trust classification scheme. Identity trust can be further classified into the identity of an object or an entity. An object can be static as well as dynamic or executable data. Therefore, an object can be a stored object or an object in transit. An entity can be the originator or a recipient of an object. These two types of entities have different trust concerns as we will see later. On the other hand, behavior trust can be attached to the behavior of an entity or the behavior of an object.

Identity trust is concerned with verifying the authenticity of an entity and determining the authorizations that the entity is entitled to access [36]. Identity trust is based on techniques including encryption, data hiding, digital signatures, authentication protocols, and access control methods. An entity might be concerned with verifying the authenticity of the originator or the recipient of an object. For example, an entity might be concerned with delivering its object to the intended recipient(s) and not its competitors. An entity might also be concerned with the authenticity of the received object. Usernames, passwords, smart cards, digital signatures, and public key infrastructure [13] are techniques used for identifying and/or authenticating the originator or the recipient of an object.

An entity can also be concerned with the privacy or the integrity of the received object or

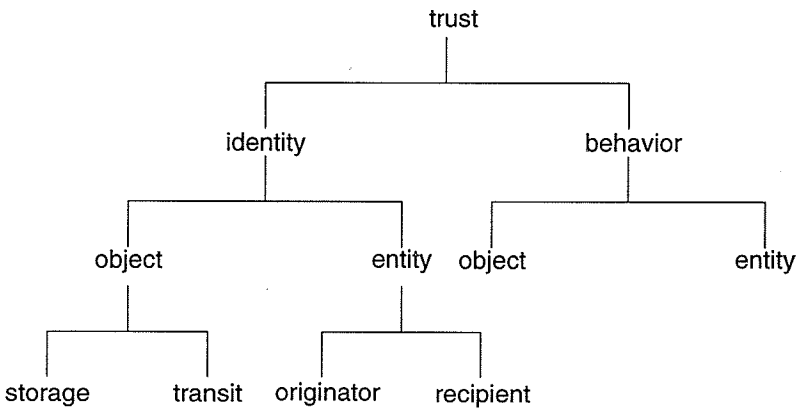


Figure 1.1: Trust taxonomy for online systems

its own object. Privacy is concerned with the secrecy of the object. An object stored on an entity's resources should only be accessible by authorized entities while an object in transit should not be accessible by any other entity as in man-in-the-middle attacks. Integrity means that the object stored in an entity's resources can not be accidentally or maliciously modified; whereas integrity for an object in transit means that the object received is the object that was sent. Access control, encryption methods, *IP Security* (IPSec), *Secure Sockets Layer* (SSL), checksums, and message integrity codes [13] are techniques used for ensuring the object's identity trust.

On the other hand, behavior trust deals with a wider notion of an object's or an entity's "trustworthiness". Behavior trust can be of concern because of the behavior of the entity. A malicious web server could accept to host Web document replicas and deliver modified versions to the user or refuse requests directed to these replicas [37]. In this case, the behavior trust of an entity (i.e. the web server) should be monitored to assure compliance with expected behavior. Behavior trust of an entity can be examined in different ways including: a) entity's accessibility: is the entity up most of the time, b) entity performance:

measures the response time of the entity, and c) honesty of the entity.

Behavior trust can also be of concern because of the behavior of the object itself. For example, a digitally signed certificate does not convey if the issuer is an industrial spy and a digitally signed code does not specify if the code is written by competent programmers [5]. The behavior of such digitally signed executable code when executed on the resources of entity x might not comply with the privacy or system regulations and policies of entity x. It should be noted that: if the object is static as in an e-mail text message, then there is no object behavior trust issue. An object behavior trust issue arises only when the object contains a dynamic or executable data.

Table 1.1 illustrates the difference between identity and behavior trust and shows why it is important to model behavior trust over the identity trust. Identity trust forms the basis for providing trust in any system. Without identity trust, we can not build behavior trust [32, 38, 3]. For example, in [27], trust is managed at two-levels: (a) the identity-level, where a "real world" identity is bound to an entity. This can be established by identification, authentication, and authorization techniques and (b) the behavior-level, where the behavior of an entity is monitored and managed. Knowing the identity of an entity enables recourse through the courts if deemed necessary [27].

The cost of evaluating the level of trust for identity trust is straight forward. The level of trust for identity trust is either 0 or 1. For example, the identification and authentication process (typically called "Login") starts with the user identifying himself by supplying his identity. The level of trust here is either successful Login (i.e., 1) or unsuccessful Login (i.e., 0). Because the identification and authentication process is based on the user's identity, the computational cost of the identification and authentication process is the same (i.e., fixed and does not change over time) whether it is the first or the tenth time the user is logging into the system. Also, identity trust does not change often. Entity x is known as x and object z is known an z unless they are given new identities. So, the changeability of identity trust is not controlled by the owner but rather it is imposed by an external entity

such as a system administrator. That is, identity trust is granted and it can not be developed over time. An entity does not develop its identity based on its behavior, but rather the identity is granted by some authoritative entity such as a system administrator.

If entity x requires a new identity by, for example, losing its identity, the identity can be easily replaced. As soon as an entity gets an identity, this identity establishes itself in the system. For example, if the entity x wants to access a remote data, the remote server checks its access list. If x is in the access list, the remote server grants x access to the data. Otherwise, x is denied access to the data. The establishment of x in the remote server's access list does not evolve, but rather is either 0 or 1 (i.e., either x is in the access list or not).

On the other hand, behavior trust is built as another layer to identity trust [32, 3, 27, 28]. However, the cost structure associated with providing behavior trust is more complex than the identity trust. For example, since behavior trust can be based on experience over time, the cost structure associated with establishing initial behavior trust is complex and challenging for newcomers in an online environment [10]. The cost of allowing entity x or entity y to access certain resources depends on the trustworthiness of these two entities [5, 39]. The more trustworthy an entity is, the lesser concern and the lesser cost that need to be employed to guard these resources from it. Behavior trust changes more often depending on the behavior of the entity. Further, behavior trust is gained and not given [10, 40]. An entity can approve or disapprove itself as being trustworthy to other entities. If entity x loses its trustworthiness because of its misbehavior, x's trustworthiness can not replaced [40]. An entity needs to gain its trustworthiness by interacting with other entities and correcting its reputation. This process is not as fast as replacing an identity. It takes more time to gain the trust of other entities from the behavior point of view. Other entities' perception of entity x to be trustworthy or not is learnt gradually through interaction with it [40].

Table 1.1: Comparison of identity and behavior trust.

Trust attribute	Identity trust	Behavior trust
Importance	basis (foundation)	layer
Cost	straight forward	complex
Changeability	very seldom	frequent
Creation	granted	developed
Replacement	yes	no
Propagation	immediate	with time
Perception	exists	learned

1.4 Thesis Overview

In this thesis, a trust model for network computing systems is presented. The trust model is defined and the schemes used in the model are described. Peer-reviews are one of the key mechanisms in this trust model. The model presents a concept of accuracy to enable peer review-based mechanisms to function with imprecise trust metrics, where different peers can evaluate the same situation differently. By introducing a concept of honesty, the trust model handles situations where peers intentionally lie about others for their own benefit. Simulation studies demonstrate that these two conditions can be handled by the trust model mechanisms. Further, the simulation studies show the importance of properly handling these conditions for trust modeling that uses P2P reviews.

To improve the overall scalability and efficiency, we use aggregation to segment the network computing system into domains. A domain assumes the responsibility of managing the trust of the set of resources within it by estimating the best trust levels for the resources, using the trust levels to group the resources into clusters, representing the resources to other

domains, and penalizing or rewarding the resources for their behavior. One of the advantages of the two-level trust model is the facility to include different initial trust levels into the model for new resources. In addition, the trust notion managed by this model is used in resource management systems to optimize the operation of a large-scale distributed system such as Grid [41]. Simulation studies indicate that the performance resource management systems can improve the overall quality of the schedules obtained by the allocation process.

In the rest of this thesis, unless explicitly stated, trust refers to behavior trust. The rest of the thesis is organized as follows. Chapter 2 examines the related literature and gives background information on identity as well as behavior trust models. Chapter 3 defines the notions of accuracy, honesty, trust and reputation. Mechanisms for computing these notions are outlined in Chapter 3 as well. Mapping the trust model onto a P2P system is presented in Chapter 4, while Chapter 6 discusses the scalability of the trust model. Chapter 5 outlines simulation studies performed to evaluate the trust model and discusses their results. Chapter 7 presents a case study designed to investigate the utility of the trust model. Simulation studies are also performed in Chapter 7 to investigate the effectiveness of the utility of the trust model. The thesis closes with future work and conclusions in Chapter 8.

Chapter 2

Literature Survey

2.1 Overview

This chapter provides a detailed description of literature related to trust modeling, examines each one of them, and highlights the unique aspects of the proposed trust model from the existing ones. Section 2.2 introduces and defines some terms used in the rest of the chapter. These terms are formally defined in Chapter 3 but included here for completeness purposes. The related work done pertaining to trust modeling is grouped into two main categories: (a) Models for identity trust (Section 2.3) and (b) Models for behavior trust (Section 2.4). We will use the term identity trust to mean security. Although identity trust is not directly related, the chapter includes a general survey of identity trust models to give a full picture of the "trust" notion. For the relevance and distinction between identity and behavior trust, please refer to Section 1.1.

The primary emphasis will be on modeling behavior trust which is the focus of the thesis. Behavior trust models can be categorized into three groups. First, reputation-based

models, where a reputation management system gathers, distributes, and aggregates reviews about participants' behavior. These mechanisms help participants in making decisions about whom to trust and also provide an incentive for honest behavior. Second, direct and reputation-based (hybrid) models, where a peer relies on its own experience as well as reviews from other peers to make a trust decision. In the hybrid models, a peer has the flexibility to weigh the two components (direct experience and reputation) differently. Third, incentive-based models which use market-based or payment mechanisms to give an incentive to the buyer or the seller in such a way that they do not just honestly reveal their trustworthiness but also benefit from the trade. Incentive-based models discussed here are applicable for E-commerce environments. Incentive-based models refer to the reputation and hybrid-based models as *threshold models* because reputation-based models and hybrid-based models assume that a peer will only engage in a trust relationship if the level of trust exceeds some particular threshold (i.e., an acceptable level of trustworthiness). Incentive-based models are proposed to ensure that it is in the best interest of a peer to honestly report information.

Section 2.5 highlights the limitations of the existing trust modeling approaches and how our proposed trust model overcomes these limitations, while Section 2.6 provides references to some trust working groups and the current status of trust modeling. Section 2.7 closes the chapter.

2.2 Definitions

For completeness and clarification purposes, we include the definitions of trust terms used in this chapter. Motivations of using these terms in our behavior trust model, their quantifications, and how to compute and use them are explained in Chapter 3. The following are definitions of identity trust, behavior trust, reputation, honesty, and accuracy, respectively.

Identity trust is concerned with verifying the authenticity of an entity and determining the authorizations it is entitled to access.

Behavior 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.

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.

Entity x is said to be honest if the information, pertaining to a specific entity within a specific context at a given time, received x is the same information that x believes in.

An entity 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 within a precision threshold.

2.3 Models for Identity Trust

Trust models such as PGP [33] and X.509 [34] as well as trust management applications such as PolicyMaker [42] and KeyNote [43] are concerned with *identity trust*. PGP and X.509 are two of the main trust models used for authentication using digital certificates based on public key cryptography. That is, these two models can be used to guarantee the identity of the originator or the recipient of the object. PGP's digital certificates are used primarily for privacy and authentication relating to e-mail type of applications between

human users. PGP trust model assumes no centralized or hierarchical relationship between certification authorities [44].

On the other hand, X.509 is a strictly hierarchical trust model used for authenticating web transactions (i.e., authenticating the user or the web server) by offering a digital certificate as a proof of identity. Digital certificates such that issued by PGP and X.509 do not bind access rights to the owner of the public key. PolicyMaker and KeyNote specify the access rights of a public key. Knowing what a public key is authorized to do, can be used to enforce secrecy and integrity of a stored object.

The *Secure Sockets Layer* (SSL) [45] is a trust model that provides application encryption for Web browsers and it protects the secrecy and integrity of an object sent from an application that uses SSL. That is, it is a protocol that protects data sent between Web browsers and Web servers by insuring that the data came from the Web site supposed to have originated from and that no one tampered with the data while it was being sent. For example, SSL is used in virtually all the encrypted e-commerce credit-card transactions today. Any Web site address that starts with "https" has been SSL-enabled.

IP *Security* (IPSec) [46] is a standard suite of protocols that ensure private and trusted communications over IP networks. In other words, while the object is in transit, IPSec ensures the secrecy and the integrity of the object. IPSec achieves this by implementing network layer encryption and authentication providing an end-to-end object identity trust solution.

In [47], a security architecture for a Grid system is designed and implemented in the context of the Globus system [48]. In [47], the security policy focuses on authentication and a framework to implement this policy has been proposed. Authentication is provided so that users and resources can identify and verify themselves when creating computational entities (i.e., processes).

A design and implementation of a secure *Service Discovery Service* (SDS) is presented in [49]. SDS can be used by service providers as well as clients. Service providers use

SDS to advertise their services that are available or already running while clients use SDS to discover these services. In SDS, privacy and integrity are maintained via encryption of all information sent between system entities.

Identity trust mechanisms do not consider *behavior trust* which changes over time and thus these approaches have no mechanisms to monitor behavior trust relationships. In addition, these trust models and trust management applications do not recognize the need for entities to learn from past experiences in order to dynamically update their trust levels [44, 31].

2.4 Models for Behavior Trust

Before x decides to have a transaction with y, it might want to determine the reputation of y to assess the risk involved with the transaction. Therefore, x might have to manage two different kinds of information. One resulting from its own experience (i.e., direct trust) and the other resulting from gathering recommendations about y (i.e., reputation of y). Therefore, trust models can be based on reputation or on direct trust and reputation. Trust models can be incentive-based as well. Incentive-based mechanisms discussed here are of two types:

- A solution proposed as an alternative to reputation-based models. The claim is that if incentives are made to y such that y honestly reveals its trustworthiness, then the collection of reputation information is not needed and thus trust management costs can be reduced.

- A solution to make reputation-based models incentive-compatible. That is, to ensure that it is in the best interest of an entity to honestly share reputation information.

Thus, this section classifies behavior trust models into: (a) reputation-based models, (b) direct trust and reputation-based (hybrid) models, and (c) incentive-based models.

2.4.1 Reputation-based Models

A Reputation-based Trust Model for Peer-to-Peer eCommerce Communities

A reputation-based trust model is presented in [50] for P2P electronic communities. In eCommerce P2P settings, peers often have to interact with unfamiliar peers and need to manage the risk that is involved with the interactions. One way to help minimize such risk is to use reputation-based reviews to help evaluate the trustworthiness and predict the future behavior of target peers. In [50], authors introduced a trust model called PeerTrust. In PeerTrust, the trustworthiness of a peer is evaluated in terms of reputation it receives in providing service to other peers in the past. After each transaction, the two participating peers give feedback about each other on the current transaction. For example, after peer x interacted with peer y, x gives feedback according to its satisfaction with interacting with y and y gives feedback according to its satisfaction in interacting with x. That is, each peer is bound to give feedback after each transaction. Feedback is collected by a distributed feedback system within PeerTrust. Each time when a source peer is interested in evaluating the trustworthiness of a target peer, the source peer has to retrieve the feedbacks of other peers which have interacted with the target peer.

For example, suppose that peer x wants to determine the trust level of peer y. Let $I(y)$ denote the total number of interactions peer y engaged in for a specific period of time, $p(y,i)$ denote other participating peer in y's ith interaction, $S(y,i)$ denote the satisfaction y receives from $p(y,i)$ in its ith interaction, and $Cr(p(y,i))$ denote the credibility of the feedback submitted by $p(y,i)$. The value $S(y,i)$ is between 0 and 1 and quantifies the amount of satisfaction y receives from $p(y,i)$ in its ith interaction. The value $Cr(p(y,i))$ can be quantified by using a function of the trust level of a peer. That is, feedback from trustworthy peers are considered more credible and thus weigh more than those from untrustworthy peers.

Then, x can quantify y's trust level as follows:

$$T(y) = \frac{\sum_{i=1}^{I(y)} S(y,i)\, Cr(p(y,i))}{I(y)} \tag{2.1}$$

After x collects the feedback from other peers that have interacted with y, x computes the trust level of y by averaging the credible amount of satisfaction y receives for each transaction performed. A simple rule of determining y's trustworthiness can use a threshold value. If $T(y)$ is greater than the threshold value, then x considers y to be trustworthy. Otherwise, y is considered to be untrustworthy.

One limitation of this model is that it has no mechanism for preventing a dishonest peer from inserting arbitrary bogus number of feedbacks and potentially causing a *denial of service* attack. Further, this mechanism does not prevent cheating via collusion, where a group of peers secretly agree or cooperate especially for an illegal or deceitful purpose. In addition, this approach has no mechanism for filtering out and isolating dishonest peers from the reputation network. A peer gathers feedbacks from other peers regardless of their honesty. This practice is not just inefficient, but also gives continued opportunities for dishonest peers to damage and influence the feedback-based reputation network. Because our model uses the *honesty* concept, such dishonest peers will be detected and isolated from the feedback-based reputation network. Further, our trust model prevents cheating via collusion by introducing the concept of *accuracy*, which enables our trust model to adjust each recommendation and ensures that the received information pertaining to a target peer is as close as possible to the trustworthiness of the target peer.

A Reputation-based Approach for Choosing Reliable Resources in Peer-to-Peer Networks

A reputation-based approach focusing on P2P applications for file exchange is proposed in [51]. This approach extends Gnutella-like [52] environments by guiding peers in their

decision making on the file to download as well as the peer offering the file. This approach is motivated by the fact that P2P file sharing applications introduce a whole new class of trust threats such as distributing viruses or Trojan horses.

In [51], each peer has a unique identifier (i.e., peer_id) being a digest of the peer's public key, obtained using a secure hash function. In addition, each file is associated with a digest computed by applying a secure hash function to the file's content. Each peer x maintains two datasets, a file dataset and a peer dataset. A file dataset maintains two attributes: peer_id and value. Each time x downloads a file from a target peer y, x keeps y's peer_id and the value which is either good (+) or bad (-) describing x's opinion in the file downloaded from y. On the other hand, A peer dataset has three attributes: peer_id, num_plus, and num_minus. Each time x downloads a file from y, x keeps y's peer_id, the number of successful (i.e., num_plus) and unsuccessful (i.e., num_minus) downloads.

The proposed approach in [51] uses a protocol that works as follows: First, the source peer, searching for a specific file to download, broadcasts a query message to all of its neighbors. Query replies contain the offerer's id and the digest of the file it is offering. The source peer has to select among the possibly different peers offering possibly different files. The selection process can be guided by the source peer's preference and/or by the number of offerers. In principle, the source peer selects a list of top files and the file to download is selected by polling votes. Vote polling is performed via broadcasting to all the source peer's neighbors. There are different ways a voter can cast its vote and it is up to the Gnutella configurator to decide the specific choice to adapt. For example, a peer could vote *yes* only if num_minus is zero or if num_plus is much higher than num_minus. Once the source peer receives the votes, it uses a majority voting scheme to determine the best offerer. To prevent overloading the best offerer, the source peer contacts the best offerer for the file digest. Based on the majority voting scheme, the source peer decides from which peer to download the file and it starts the download. After downloading, the file is checked against the digest to ensure the file's integrity. Finally, the source peer updates its datasets

according to its satisfaction with the file downloaded and the peer offering it.

This approach has no mechanism for filtering out and isolating dishonest peers from the reputation network. A peer broadcasts to all of its neighbors regardless of their honesty. This practice is not just inefficient, but also gives continued opportunities for dishonest peers to damage and influence the reputation network. Because this approach uses a voting scheme to determine the truth, a peer can be fooled into accepting a tampered file especially if the majority of recommenders are dishonest. Further, this model is based only on reputation and deals with file sharing or file exchange.

A Distributed Trust Model for Peer-to-Peer Networks

A P2P trust model called *Poblano* is presented in [53]. Poblano allows peers to communicate their "opinion" on both the information they have received and on the peers that are its source. Poblano coined the term "codat" which is a general term that covers static as well as dynamic or executable data. In Poblano, a peer is connected to at least one peer group, which is a dynamic set that have agreed upon a common set of policies and services. Peer group's membership is motivated by *keyword* interest. In Poblano, the "keyword" is used instead of "context" that is used in our trust model. A peer uses the values in Table 2.1 to quantify its confidence. These values are based on the work done in [5].

Table 2.1: Confidence Values.

Value	Meaning
-1	Distrust
0	Ignore
1	Minimum trust
2	Average trust
3	Good trust
4	Complete trust

Over time, each peer will acquire a local collection of codats. This local collection of codats will be kept in a local *CodatConf* table. A CodatConf table is categorized by keyword and has four attributes: $codat\_id$, $flag$ of making the codat either local or remote, $popularity$ and $relevance$. Popularity is monotonically increasing and is incremented each time the codat is requested. Relevance measures how relevant a codat is to the keyword and is quantified using the same values shown in Table 2.1. In addition, each peer group maintains two more tables, namely *PeerConf* and *PeerGroupConf*. PeerConf and PeerGroupConf tables are categorized by keyword and each has two attributes: $peer\_id$, and $confidence$. PeerConf table can be accessed by all those peers in a given peer group, whereas PeerGroupConf table can be accessed by all the peer groups to which peer x belongs.

One of the applications of Poblano is reputation guided search in JXTA [54], which is a P2P platform providing developers of P2P applications with a wide range of libraries and services like routing, peer discovery, and peer communication. Specifically, Poblano is used as the trust model for distributing signed certificates among peers in JXTA. When peer x does a search to find a given peer y's certificate (i.e., using the given peer's public key for securing a transaction), x will do the following. First, x looks up the keyword "signed certificates" in its own CodatConf table. If there is a local signed certificate (i.e., the flag is set to local) for y, then the search succeeds. Else, x looks in the PeerConf table to see if there are any peers highly associated with keyword "signed certificates". That is, if there are any peers in which x has a high confidence (i.e., a confidence value ≥ 2), then x forwards the request to those peers. Otherwise, x can resort to the PeerGroupConf table to forward the request to other peers highly associated with the keyword "signed certificates" across all the peer groups to which x belongs.

If the search is successful, x requests the target peer's certificate. The provider (i.e., the peer that provided the target peer's certificate) increments the signed certificate's popularity by one as soon as it is requested by x. After using the certificate, x adds an entry in its

CodatConf table and sets codat_id to the target peer's certificate id, flag to local, popularity to zero (since no peer has requested the certificate yet from x), and relevance to x's satisfaction of how relevant the certificate is to the search keyword . In addition, x generates or updates an entry in its peer group table (i.e. PeerConf) setting peer_id to the provider's id and confidence to x's confidence of the signed certificate. The same codat rating will be also fed back to the provider which may choose to update its own rating based on: (a) its confidence in x, (b) previous rating value of the signed certificate, and/or (c) the feedback value on the signed certificate.

A major drawback of Poblano is that it does not learn from the information available to it. For example, when peer x does a search to find a target peer's certificate, x looks up the keyword "signed certificates" in its own CodatConf table. If there is a local signed certificate matching what x is looking for, then the search succeeds. This practice can lead x to use a compromised given peer's certificate (i.e., using a compromised given peer's public key for securing a transaction). Because x's local information in its CodatConf table might be outdated, x will be using a compromised given peer's public key for securing its transactions. Because our model combines direct experience with reputation, a peer always probes its recommenders to get current information. Also, our model uses a decay function to distinguish between old and current information as explained in Chapter 3.

A Trust Model for Peer-to-Peer Content Distribution Networks

A trust model for P2P *behavior trust* in *Content Distribution Networks* (CDNs) is proposed in [37]. CDNs are systems that dynamically move content (e.g. web content) close to the clients and these clients are transparently redirected to the content nearest to them. The emphasis in this model is on web servers which potentially belong to different administrative peers and can cooperate to replicate their documents worldwide. The trust model represented in [37] requires that each peer provides recommendations to a number of other

peers and that these recommendations are publicly available to any peer in the system. A recommendation represents the level of trust that one peer has in another.

The motivation of this trust model is that once web servers start cooperating and replicating their documents worldwide, such a system presents obvious trust concerns. A dishonest web server can accept to host web document replicas and deliver modified versions to the users. A dishonest web server could accept to host replicas and refuse to answer any request directed to them. Further, a dishonest web server could allocate less resources to replicas than was negotiated.

This trust model uses a voting scheme to aggregate several recommendations into a single trust value. This trust model assumes that similar independent recommendations reinforce each other and if all recommendations but one agree, then the opposite recommendation is wrong. This is a major drawback of the trust model as it opens the door for cheating via collusion, where a group of peers secretly agree or cooperate especially for an illegal or deceitful purposes. Such a group of peers tend to be dishonest and give poor recommendations (e.g., when asked for recommendations regarding one of their opponents) in order to isolate their opponents and leave them out of the competition. Because our model uses the *honesty* concept, such recommenders will be detected and isolated from the recommenders network. Further, this model is based on recommendations and uses an *exponentially weighted moving average* (EWMA) algorithm [55], when updating the recommender set. EWMA algorithms are either able to detect true changes quickly or to mask observed noise and transients [55]. Hence, dishonest peers can cheat every n transactions and still be considered trustworthy.

Robustness of Reputation-based Trust

Assuming that less than 50% of a population of entities are dishonest, a simple reputation-based polling mechanism is presented in [56]. This mechanism assumes that an entity x has

identified one (entity y) of several entities that can provide a service that x needs. Since the performance of service providers varies significantly, x is interested in selecting a service provider with high performance. As x lacks prior knowledge about the performance of y, x will poll a group of recommenders who have knowledge of y's performance. The goal of entity x is to select the service provider who has the highest reputation.

The motivation of this trust mechanism is that in dynamic and open societies, autonomous agents interact with each other with little or no information. Therefore, there is always a chance that a rogue agent may take advantage of an unwitting agent by pretending to offer assistance for malicious hidden motives.

Because the percentage of the dishonest entities is assumed and this percentage should be less than 50% of the total population, this trust scheme uses a majority voting when combining recommendations. Majority voting is not a suitable way to combine recommendations because it opens the door for cheating via collusion, where a group of peers secretly agree or cooperate especially for an illegal or deceitful purposes. Such a group of peers tend to be dishonest and give poor recommendations (e.g., when asked for recommendations regarding one of their opponents) in order to isolate their opponents and leave them out of the competition. Because our trust model distinguishes between trustworthiness and honesty, it is able to perform well even though the dishonest entities exceed 50% of the total entities population. Further, assuming less than 50% of the entities population to dishonest, is not a reasonable assumption for autonomous agents which are software entities that are capable of independent action in dynamic and unpredictable environments. Hence, using majority voting, an entity can be fooled if the majority of recommenders are dishonest.

Managing Trust in a Peer-2-Peer Information System

A scheme for trust management in a P2P information system is proposed in [57], where the focus is on implementing a generic scalable infrastructure to deploy a trust model. This

model is based on binary trust meaning that an entity is either trustworthy or untrustworthy. This trust model assumes that usually trust exists and distrust is the exception. Hence, the model considers only the information on untrustworthy interactions as relevant. That is, the only time entities post information is when they have a complaint based on bad experiences they had while interacting with other entities. For example, assume that p and q interact with each other and r later wants to determine the trustworthiness of p and q. Let us assume that p is untrustworthy and q is trustworthy. After their interaction, q files a complaint, which is perfectly fair since p is untrustworthy. Now, p will also file a complaint about q in order to hide its misbehavior. Entity r can not distinguish whether p or q is untrustworthy. But, the trouble for p starts when it continues to cheat with another entity s. Then r observes that p complains about q and s, whereas both q and s complain about p. Hence, r concludes that p is untrustworthy. The model assumes that relatively few entities cheat.

One limitation of this model is that it is based on a binary trust scale (i.e., an entity is either trustworthy or not). Hence, once there is a complaint filed against entity q, q is considered untrustworthy even though it has been trustworthy for all previous interactions. Also, this approach has no mechanism for preventing a dishonest entity from inserting arbitrary number of complaints and potentially causing a *denial of service* attack. Further, this mechanism does not prevent a group of entities from cheating via collusion. For example, a group of entities, for deceitful purposes, can secretly agree to file complaints against a particular entity.

2.4.2 Hybrid-based Models

The Ponder Policy Specification Language

A survey of trust in Internet applications is presented in [44] and as part of this work a policy specification language called Ponder [35] is proposed for specifying security and management policies as well as allowing the specification of more abstract and complex

behavior trust relationships. Ponder recognizes the need for peers to learn from their past experience (i.e., direct trust) as well as for a trust network (reputation) to dynamically determine and adjust trust levels. Ponder can be used to define authorization and security management policies. Ponder is being extended to: (a) allow for more abstract and potentially complex trust relationships between entities across organizational domains and (b) have mechanisms for monitoring trust relationships and acknowledge that trust changes over time.

The same group in [44] is working on trust specification language [20] for e-commerce and hoping to use these specifications to generate Ponder trust management policies. The authors in [20] view the essential components of a trust relationship as: a trustor, a trustee, a specific context, and conditions under which the trust relationship can proceed. In [20] trust specification has two constructs: (a) trust construct and (b) recommend construct. Trust construct is used to specify a trust relationship and it is of the form:

```
PolicyName:  trust( Trustor, Trustee, ActionSet, Level)
ConstraintSet;
```

PolicyName is the name of the policy being defined. Trustor and Trustee refer to objects or people, whereas ActionSet is a list of semi-colon delimited actions which specify the context for the trust relationship. Level is an integer in the range of 1 to 100 for trust and -1 to -100 for distrust, which defines a particular trust/distrust level for the actions forming the part of the trust relationship. ConstraintSet is the set of conditions that must be met for the trust relationship to be established. Recommend construct is used to specify positive or negative recommendations and has a similar layout as the trust construct.

Trust Models such as the ones proposed in [5] and in this thesis can feed the trust attributes (e.g., trustor, trustee, context, and trust level) into this trust specification language (i.e., [20]), which then can be tailored for use at the management level. For example, a system administrator with a global view of the system resources and needs can utilize these specifications to augment the overall performance of the system.

Managing Trust and Reputation in the XenoServer Open Platform

The XenoServer project [27] is developing an infrastructure for general-purpose and public wide-area computing, capable of hosting tasks that span the full spectrum of distributed paradigms. The architecture of XenoServer Open Platform consists of XenoServers, XenoCorp, XenoServer Information Service (XIS), and clients. XenoServers host tasks that are submitted by clients and a XenoCorp act as a trusted third party. The job of a XenoCorp is to authenticate XenoServers and clients by signed credentials that can be tied to the issuing XenoCorp. That is, XenoCorp is the authority responsible for registering the clients and XenoServers that wish to participate in the XenoServer Open Platform. XenoCorp has the absolute authority for registering and ejecting participants.

In such a public open environment, XenoServers face some threats. Disreputable clients may try to: (a) place tasks on them and not pay and (b) run tasks that use XenoServers as the source of some nefarious activity on the network. On the other hand, clients may find that their tasks are not executed properly. A client may may also find that an untrustworthy XenoServer may over charge for resource consumption or the XenoServer might be using unreliable machines.

Fundamental to all of this is trust. XenoTrust [27, 58] is the trust management introduced for use in the XenoServer Open Platform. It takes a two-layer approach to managing trust by distinguishing *authoritative* and *reputation-based* trust. Authoritative trust is a boolean property established between a XenoCorp and clients/XenoServers that register with it. Authoritative trust is made possible by the credentials issued by the XenoCorp. These credentials can be validated by any of the XenoCorp's clients or XenoServers. The Authoritative trust makes XenoCorp aware of a "real world" identity bound to the participants. This enables recourse through the courts (i.e., judicial system), should it be necessary. The second layer of the XenoTrust model is the reputation-based trust, which

is a continuous property quantifying the trustworthiness that one client or XenoServer ascribes to another. In XenoTrust, this layer is called reputation-based trust but actually it is a hybrid-based trust because a XenoServer or a client can rely on reputation-based information as well as its own experience. This will become clear in the upcoming example. Individual clients and XenoServers will build up this reputation-based trust information locally as they interact with others. The participants' views are subjective. For example, some clients might favor a XenoServer correct pricing advertisements, while other clients might favor a XenoServer with a reliable service. It is the participant's responsibility to decide how to interpret the scores in its reputation database (i.e., to decide at what point a counterpart is deemed untrustworthy of further interactions). Participants will also tell others about their experiences and it is up to participants to decide how to deal with the reports that they hear.

Reputation-based trust involves three steps: statement advertisement, rule-based deployment, and reputation retrieval. We explain these three steps by giving the following example. Let us assume that two Xenoservers (A and B) have been hosting tasks on behalf of client X. All of these XenoServers advertise their experience with client X. The following are the advertisements held in the XenoTrust system and are available for public view.

$$(A, X, payment, 1, 50)$$

$$(B, X, payment, 0.27, 10)$$

$$(D, X, payment, 0.40, 25)$$

$$(D, A, belief, 1)$$

$$(D, B, belief, 0)$$

$$(X, X, payment, 1, 100)$$

The first three advertisements above show three XenoServers (A, B, and D), which have hosted tasks on behalf of a client X. The first three advertisements labeled "payment" refer to the XenoServer's view of whether client X is reputable when it comes to settling their bills and thus, indicating how likely a XenoServer believes (a value between 0 and 1) an interaction with X is to proceed without problems in this regards. The advertisement also shows how many times a XenoServer has interacted with X. For example, XenoServer A assigns X the maximum score (i.e., 1) stating that it has a complete trust in X paying its bills based on A's 50 direct interactions with X. While XenoServer B assigns X a low score of 0.27 stating that it has 0.27 trust level in X paying its bills based on B's 10 direct interactions with X. The third "payment" advertisement made by XenoServer D assigning X a trust level 0f 0.40 regarding X's bills payment based on D's 25 direct interactions with X.

The second two advertisements labeled "belief" are advertisements made by XenoServer D indicating the value (between 0 and 1) it places on statements made by A and B. For example, D places 1 indicating a complete trust on statements made by A. On the other hand, D places 0 indicating a complete distrust on statements made by B. The last advertisement is made by X about itself, claiming to have interacted 100 times and always successfully.

Different users (Xenoservers and clients) of XenoTrust have installed different rule-sets to quantify their trust in clients or XenoServers on the basis of the advertisements that have been made. For example, if D wants to evaluate the trustworthiness of X, D can use a rule set "scaled", which combines advertisements and scale them according to D's "belief" statements. In this case, D will ignore B's advertisement and consider only its own and A's advertisements. Other rule sets, which D can use, include discarding X's self-referential advertisement before combing the other advertisements according to various weighed averages.

One limitation of this model is that it has no mechanism for preventing a dishonest users from inserting arbitrary bogus number of advertisements and potentially causing a *denial*

of service attack. Further, this mechanism does not prevent cheating via collusion, where a group of users secretly agree or cooperate especially for an illegal or deceitful purposes. Because our model uses the *honesty* concept, such dishonest users will be detected and isolated from the recommenders network. Further, our model uses the *accuracy* concept to adjust each recommendation and hence prevents cheating via collusion.

Supporting Trust in Virtual Communities

A model for supporting *behavior trust* based on experience and reputation is proposed in [5]. This theoretical 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. This trust model describes direct trust as an entity x's belief in another entity y's trustworthiness within a certain context c to a certain trust level. These trust levels range from *very untrustworthy* to *very trustworthy* and represented by 1 to 4, respectively. The set of entities that x directly interacted with are kept in set Q_x. For each target entity directly trusted, x keeps the following three attributes in Q_x: the target entity's id, the context, and outcome. Outcome $T = (n_4, n_3, n_2, n_1)$ is a four-tuple (very trustworthy, trustworthy, untrustworthy, very untrustworthy). For example, Let us assume that entity x has directly interacted with entities y and z. Therefore, $Q_x = \{(y, c_2, (2, 0, 3, 1)), (z, c_1, (1, 3, 0, 1))\}$ to mean that:

- Entity x has directly interacted 6 times with entity y for a specific context c_2. The outcome is very trustworthy for 2 of these interactions, untrustworthy for 3 of these interactions, and very untrustworthy for 1 of these interactions.

- Entity x has directly interacted 5 times with entity z for a specific context c_1. The outcome is very trustworthy for 1 of these interactions, trustworthy for 3 of these interactions, and very untrustworthy for 1 of these interactions.

Further, let us assume that x directly interacted once more with z and that x found the outcome of its interaction with z to be untrustworthy. Then x will update its Q_x as follows: $Q_x = \{(y, c_2, (2, 0, 3, 1)), (z, c_1, (1, 3, 1, 1))\}$. Now, if x wants to determine the direct trust in z for c_1, then x will return the $max = modaln_i$ value. For example, if $T = (1, 3, 1, 1)$, x's direct trust in z for c_1 is trustworthy. If max returns more than one value, x's direct trust in z is assigned uncertainty value according to Table 2.2 (the symbol ? indicates zero or one other value). *Semantic distance* (sd) quantifies the gap between the recommendation given

Table 2.2: Uncertainty values.

combination	outcome	meaning
(Number of very trustworthy equal number of trustworthy) AND ?	u^+	Mostly trustworthy and some untrustworthy
(Number of very untrustworthy equal number of untrustworthy) AND ?	u^-	Mostly untrustworthy and some trustworthy
All other combinations	u^0	Equal amount of trustworthy and untrustworthy

by recommender r (rec_r) and x's own perception of the recommendation (per_x). Semantic distance value are given by the set $S = \{-3, -2, -1, 0, 1, 2, 3\}$ and can be calculated by the following equation:

$$sd = rec_r - per_x \tag{2.2}$$

Consider the following two examples, where x asks r_1 for recommendation for a specific context c_2 regarding a target entity:

1. Recommender r_1 gave very trustworthy as a recommendation for the target entity. After x has interacted with the target entity, x found the target entity to be very trustworthy. Since very trustworthy is represented by the numeric value of 4, then

r_1's semantic distance is $sd_{r_1} = 4 - 4 = 0$.

2. Recommender r_1 gave very untrustworthy as a recommendation for the target entity. After x has interacted with the target entity, x found the target entity to be very trustworthy. Since very untrustworthy and very trustworthy are represented by the numeric values of 1 and 4, respectively, then r_2's semantic distance is $sd_{r_1} = 1 - 4 = -3$.

The trust model also describes recommender trust as an entity x's belief that another entity y is trustworthy to a certain trust level for giving recommendations about other entities with respect to c. Entity x keeps its recommenders in set R_x. For each recommender, x keeps the following three attributes in R_x: the recommender's id, the context, and outcome. Here, outcome is four sets = ({very trustworthy}, {trustworthy}, {untrustworthy}, {very untrustworthy}). For example, Let us assume that entity x has 2 recommenders r_1 and r_2. Therefore, $R_x = \{(r_1, c_1, (\{0\}, \{\}, \{\}, \{\})), (r_2, c_1, (\{\}, \{-2\}, \{\}, \{\}))\}$ to mean that:

- Entity x has asked r_1 for recommendations for a specific context c_1 regarding a target entity. Recommender r_1 gave very trustworthy as a recommendation for the target entity. After x has interacted with the target entity, x found the target entity to be very trustworthy. Therefore, r_1's *semantic distance* (sd_{r_1}) is 0.

- Entity x has asked r_2 for recommendations for a specific context c_1 regarding a target entity. Recommender r_2 gave trustworthy as a recommendation for the target entity. After x has interacted with the target entity, x found the target entity to be very trustworthy. Therefore, r_2's *semantic distance* (sd_{r_2}) is -2.

Further, let us assume that x asked once more for recommendations for a specific context c_1 from r_1 and r_2 regarding a target entity. Let r_1 give very trustworthy and r_2 give untrustworthy as their recommendations. After x has interacted with the target entity, x found out that the target entity to be very trustworthy. Therefore, r_1's sd_{r_1} is

0 and r_2's sd_{r_2} is 2. When x updates its recommender set, R_x will look as follows: $R_x = \{(r_1, c_1, (\{0,0\}, \{\}, \{\}, \{\}), (r_2, c_1, \{\}, \{-2\}, \{\}, \{2\}))\}$

The trust model presented in [5] suggests that in order to reduce the uncertainty, new entities are initialized and equipped with a number of "trusted" entities in their Q and R sets so that initial transactions can be made with already trusted parties. From [5], some issues not discussed as: (a) how are Q and R equipped, (b) who equips them, and (c) with how many "trusted" entities Q and R are equipped with.

One of the drawbacks of this model is its use of an EWMA algorithm for updating Q and R. The two sets Q and R are updated such that observed noise and transients are masked. For example, let $Q_x = \{(z, c_1, \{3,0,1,2\})\}$. That is, over the 6 transactions that x has directly interacted with z, z has been untrustworthy in 1 transaction and very untrustworthy in 2 transactions. In spite of that, x determines the direct trust in z for c_1 to be very trustworthy. In other words, a recommender can give intentionally false recommendations about a few domains and maintain high overall *accuracy*. Because our model uses the *honesty* concept, such recommenders will be detected and isolated from R. In addition, because of this trust model's inability to filter out and isolate dishonest entities from the reputation network, an entity asks all of its neighbors regardless of their honesty. This practice is not just inefficient, but also gives continued opportunities for dishonest entities to damage and influence the reputation network. Further, the scalability of the model is not explicitly addressed in this study. For example, there is no limit on the number of recommenders (i.e. size of R) that each entity has; nor there is a discussion on how well the model will scale as the number of entities grow. In our trust model, the scalability issue is addressed by the aggregation scheme.

Some concepts in our trust model are influenced by the work done in [5]. For example, the direct trust and recommender trust tables, the context, and the different trust levels. These concepts are detailed in Chapter 3.

An Evidential Model for Distributed Reputation Management

A reputation management model for a P2P multiagent system is proposed in [31]. An entity determines the trustworthiness of a correspondent by combining its local experience with the testimonies of other entities regarding the same correspondent. Using this model, entities find trustworthy correspondents by collaborating with others to identify those whose past behavior has been untrustworthy. In this model, each entity x has a set of *acquaintances*, which is a set of neighboring entities that x would contact to get testimonies of other entities behavior. Entity x maintains two attributes for each of its acquaintances, namely *expertise* and *sociability*. Expertise is the acquaintance's ability to act in a trustworthy manner, whereas sociability is the acquaintance's ability to refer to other trustworthy entities. Each entity modifies its set of acquaintances based on the entity's direct interaction with the acquaintance as well as the entity's interaction with entities referred to by the acquaintance.

This approach does not prevent dishonest entities from generating spurious ratings and assumes that the majority of the entities offer honest ratings to cancel the effect of dishonest entities.

2.4.3 Incentives-based Models

These models [59, 40] are proposed and focus on giving *incentives* to peers such that it is in the best interest for each peer to truthfully reveal its trustworthiness. Peers might decline to reveal the truth because: (a) truthfully reporting any reputation information, a peer might decrease its own reputation with respect to the average of other peers, (b) truthfully reporting reputation information, a peer provides competitive advantage to other peers, and (c) Untruthfully reporting reputation information, a peer can increase its own reputation with respect to other peers.

In [59], the proposal is to make a reputation mechanism incentive-compatible. A payment system is introduced where special agents called *broker agents* buy and sell reputation

information and assume that no other payments occur between the agents themselves. That is, all payments go through broker agents. Before a peer x engages in a transaction with another peer y, it must buy reputation information about y at cost F. After the transaction, the peer can sell its reputation report to any broker agent at cost C. A broker agent will pay for the reputation report only if it matches the next report about y filled by another peer. If x truthfully reports the result of its transaction with y, then $F \leq C$. That is, x will not lose any money. Otherwise, $F \geq C$ and gradually x loses money. Some of the drawbacks of this approach are that: (a) it assumes that the majority of the peers report honest reputation information to cancel the effect of dishonest peers and (b) it goes against the notion of P2P systems by having central points (i.e., broker agents) that control the payment system.

In [40], the authors proposed a solution applicable to entities in E-Commerce that can be used as an alternative to reputation-based systems. Compared to reputation-based systems, this mechanism provides several advantages. First, the proposed solution does not require the estimation of other entities' trustworthiness. For example, a seller does not need to estimate the trustworthiness of a buyer from the buyer's previous interactions with the seller. Also, the seller does not need to estimate the trustworthiness of a buyer from reputation databases. Second, the proposed solution reduces the cost of trust management. Since entities are honestly reporting their level of trustworthiness, there is no need to store and keep large reputation databases.

This approach assumes the following. The seller produces the commodity and sells it to the buyer. The buyer always pays (i.e., he is completely trustworthy). However, the seller's trustworthiness varies from 0 (completely untrustworthy) to 1 (completely trustworthy) and measures the probability β of the seller to deliver the commodity to the buyer. The buyer does not know β and believes that the commodity will be received with probability α. Probability α can be obtained from: (a) the history of previous interactions of the buyer with the seller, or (b) from the buyer seeking the reputation of the seller. The proposed approach also assumes that the quantities of the commodity produced and sold is known,

the seller's cost as a function of the quantities produced and sold is known, and the value the buyer places on the quantity is known.

[40] shows analytically that with all the above assumptions, the transaction between the seller and the buyer can be made such that the seller not just honestly reveals its level of trustworthiness β, but also benefits from the trade. Also, the buyer benefits from the trade by knowing the trustworthiness of the seller (i.e., β) instead of estimating the seller's trustworthiness by using α. In other words, the need to speculate about the seller's trustworthiness is eliminated.

One of the drawbacks of this approach is the assumption that the information about a seller's cost function is known, which is a strong assumption and is not suitable in such a P2P business competitive environment. Also, this approach is specific to E-Commerce transactions where at least one of the trading entities (i.e., the buyer) is completely trustworthy.

2.5 Limitations of Current Trust Models

The papers discussed above examine issues that are related to the notion of trust, but these papers come short of addressing the following:

- The distinction between accuracy and honesty and the importance in treating them differently. By separating accuracy and honesty, our trust model maintains its effectiveness even when the dishonest peers exceed 50% of the population. In addition, knowing the availability of honest peers, gives a confidence level of how reliable the trust model is. For example, [56] assumes that dishonest entities form less than 50% of the total population. Also, in [57, 31] and others where a majority voting scheme is used in determining an entity's trustworthiness, honest entities are assumed to be the majority to cancel the effect of dishonest entities.

- A mechanism to function with imprecise metrics, where different entities can evaluate the same situation differently. The mechanism introduced by the proposed trust model to accomplish this is the *accuracy* concept. Our model uses the *accuracy* concept to adjust each recommendation and hence prevents cheating via collusion. In the trust model presented in [27], an entity scales a recommender's advertisement according to the entity's belief. This scaling process takes into account how accurate the recommender's advertisement is. Further, referring to the trust model outlined in [50], before entity x collects feedback from recommenders to compute the trustworthiness of entity y, x scales the feedback by the respective recommenders' credibility. However, in these trust models [27, 50] accuracy is approximated by the trustworthiness of an entity. That is, feedback from trustworthy entities are considered more accurate and thus weigh more than feedback from untrustworthy entities. In our trust model, accuracy is an independent notion from an entity's trustworthiness. In other trust models, entities post complaints [57] and advertisements [27] with no mechanisms to check for the credibility of these posts. This can lead to a group of entities to secretly agree to post bogus complaints or advertisements against a particular entity for deceitful purposes.

- A mechanism for building and maintaining an honest recommender network. The mechanism introduced by the proposed trust model is the *honesty* concept. The aim of our trust model is to end-up with honest set of recommenders and thus prevent dishonest entities from influencing the recommendations network. The current practice pursued, for example in [5, 51], is that an entity asks all of its neighbors regardless of their honesty. This practice is not just inefficient, but also gives continued opportunities for dishonest entities to damage and influence the reputation network.

- The flexibility and the importance to weigh direct trust and reputation differently. Having the flexibility of combining direct trust and reputation gives the trust model

the leeway to choose the strategy that best fits it. For example, in Poblano [53] when peer x does a search to find a target peer's certificate, x looks up the keyword "signed certificates" in its own CodatConf table. If there is a local signed certificate matching what x is looking for, then the search succeeds. This practice can lead x to use a compromised or a stale given peer's certificate (i.e., using a compromised given peer's public key for securing a transaction). Because x's local information in its CodatConf table might be outdated, x will be using a compromised given peer's public key for securing its transactions. Because our model combines direct experience with reputation, a peer always probes its recommenders to get the most recent information.

- The applicability of modeling trust by integrating it into resource management systems. To the best of our knowledge, no existing literature directly addresses the issues of trust aware resource management systems. A resource management system manages the resources that comprise the computing environment [60]. Hence a resource management system is able to allocate resources to tasks, schedule tasks, control and monitor the status of resources. Resource management systems such as Globus [48] make the allocation decisions oblivious of the security and trust implications. Our study in investigating the utility of the trust model by integrating trust into resource management systems shows that: (a) the overall performance increases when the resource management algorithm is trust aware, and (b) high levels of "robustness" can be attained by considering trust while allocating the resources. These ideas are developed in Chapter 7.

Our behavior trust model and the mechanisms used to overcome the above limitations of the current trust models are published in 5 refereed conferences [61, 62, 36, 63, 39].

2.6 Trust: Trends and Current Status

Since 1950s, trust and trust relationships have been the subject in the offline world in many disciplines including philosophy, sociology, psychology, and management [1, 64, 2]. While trust has been identified as a critical factor in many offline non-technical human endeavors, researchers are just beginning to study it in the context of technology [10, 11, 64, 65]. Trust in online communities is in its infancy [10, 65] and there is a lack of standards towards and about online trust [2, 64]. In addition, trust management is a relatively young and a complex research field with many different emerging ideas and solutions [2, 64]. While the importance of trust in on-line interactions is accepted, there is a limited theoretical support for its role in on-line communities [10, 66, 12]. Fortunately, there is a lot of research that has been done on trust in the offline communities and it is on this body that the theory of on-line trust can be built. Some of the offline communities, where trust has been researched and identified as a key factor in relationships include philosophy, sociology, psychology, and management. For example, in the field of philosophy, Baier [67] known as the grandmother of trust, defines trust as *letting other persons take care of something the truster cares about*. Baier's focus is on *interpersonal trust* (i.e. trust between people) and says that one leaves others an opportunity to harm when one trusts. Sociologists [68, 69] view trust as a social lubricant that social interactions are jeopardized without it. Most sociologists agree that trust is very important and that trust is hard to build and easy to destroy. Others such as [70] studies the conditions in which trust declines and states that trust declines with: (a) decreased interaction frequency, (b) increased interactions of "outsiders" (i.e., increased interaction with people whom we do not know), and (c) rapid change in the social community.

The field of trust online has spawned the interest of a number of scholars in technical as non-technical fields. For example, some of the projects and directions represented

at the first conference on trust management [71] held in May, 2003 include: (a) facilitating the cross-disciplinary notion of trust by bringing together expertise from technology, philosophy, and social sciences, (b) facilitating the development of new paradigms in the area of dynamic open systems which effectively utilize computational trust models, and (c) helping to incorporate trust management into existing security standards. The conference [71] is held by ITrust [72], which is an European-funded trust working group that brings together researchers and practitioners across a range of disciplines to develop models and techniques for dealing with trust in open dynamic systems. The group aims are to explore the role of trust and its interactions with security. Effective trust modeling is believed to be an enabler for a range of new computing services including enhanced e-commerce, ubiquitous computing, Grid computing, P2P computing, and probably a variety of collaborative/cooperative online activities that we haven't begun to imagine [2, 12, 3].

2.7 Summary

In this chapter, we presented the related work done in the area of behavior trust modeling. First, we presented model for identity trust. These models are not directly related to behavior trust, but discussed them to show that these models are not suitable to monitor and manage behavior trust between peer in an online community. Second, we viewed the current behavior trust models and divided them into three categories: (a) reputation-based trust models, (b) hybrid-based (direct trust and reputation) trust models, and (c) incentive-based trust models. Third, we outlined the limitations of theses behavior trust models and how our proposed trust model overcomes these limitations. To date, research based on this thesis has generated 5 publications, which are listed in Section 2.5. Last, we discussed the the current issues and mentioned recent trust management conferences and working groups whose aim is to develop models and techniques for dealing with trust in open dynamic systems.

To conclude this chapter, the existing behavior trust model schemes that tackle the trust notion are summarized in Table 2.3. When computing trust, models can be classified according to: (a) the components used to compute trust and (b) the mechanism used to aggregate recommendations. Trust models compute trust based on two components: direct trust and/or reputation. If reputation is used to compute trust, then recommendations can be aggregated by taking a weighted average or a majority voting approach.

Table 2.3: Summary of existing behavior trust models.

Behavior trust model (Authors names) [paper reference #]	Components		Recommenders aggregation scheme	
	Direct trust	Reputation	Weighted averaging	Majority voting
(Li Xiong et al.) [50]		x	x	
(Ernesto Damiani et al.) [51]		x		x
(Rita Chen et al.) [53]		x	x	
(Guillaume Pierre et al.) [37]		x	x	
(Sandip Sen et al.) [56]		x		x
(Karl Abere et al.) [57]	x	x	x	
(Boris Dragovic et al.) [27]	x	x	x	
(Tyrone Grandison et al.) [35]	x	x	x	
(Alfarez Abdul-Rahman et al.) [5]	x	x	x	
(Bin Yu et al.) [31]	x	x		x
(Radu Jurca et al.) [59]	x	x		x
(Sviatoslav Brainov et al.) [40]	x	x		x

Chapter 3

Trust Terminology

3.1 Fundamental Trust Model Concepts

3.1.1 Identity Trust

Identity trust forms the basis for providing trust in any system. Without identity trust, we can not build behavior trust [32, 38, 3]. As stated in [27], identity trust is used to bind an entity to one identity. This prevents an entity from creating multiple false identities. Establishing this link enables holding an entity responsible for its behavior. The objective of applying identity trust is to keep entities from performing undesirable activities: (a) to your data or resources and (b) with your data or resources. Identity trust is based on techniques including encryption, data hiding, digital signatures, authentication protocols, and access control methods. The definition of identity trust used in this thesis is as follows:

> *Identity trust is concerned with verifying the authenticity of an entity and determining the authorizations it is entitled to access.*

3.1.2 Behavior Trust

Behavior trust is identified as a vital component in any Internet-based transaction and lack of behavior trust is a major obstacle for the potential growth of Internet communities [1, 10, 65]. Operating in open and dynamic environments, an entity encounters unfamiliar and possibly hostile entities. For example, consider entity y transferring an executable application to execute on entity x's resources. Even though authentication and authorization mechanisms allow y to use x's resources, they do not guarantee that y's application will not perform illegal operations on x's resources. Knowledge of y's prior behavior can avoid suspicion or x's hesitance to share its resources with y.

Behavior trust deals with a broad notion of an entity's "trustworthiness". There is a lack of consensus in the literature on the definition of behavior trust and on what constitutes behavior trust management [20, 73, 5, 69]. Trust is a multi-dimensional notion that is suitable for a wide range of relationships [20]. Researchers have defined trust in different ways, which often reflects the research's background. Psychologists, such as [74], view trust as a belief or feeling rooted deeply in the personality which has its origins in the person's early psychological development. This is referred to in the literature as *dispositional trust* [11, 5]. Dispositional trust does not focus on contextual or personal factors. That is, it is cross-contextual and cross-personal. A person is said to have dispositional trust if a person has a consistent tendency to trust across a broad spectrum of contexts and people. For example, when asked whether he trusts the new boss, an employee says she trusts new people, both at work and elsewhere. The philosopher and grandmother of trust, Baier [67], and others like [75, 76] focus on a context-specific *interpersonal trust* involving a trustor and a trustee. Interpersonal trust is a perception the trustor may have, that the trustee will not intentionally or unintentionally do anything that harms the trustor's interests. For example, if you hire a babysitter to care for your newborn while you and your spouse go for an outing, your interest is the safety of your newborn. Going for an outing with your spouse

is not nearly as valuable as what you may lose if your trust is violated. Your babysitter may abuse or harm your newborn. In sociology, researchers such as [77, 70] focus on *system trust* or *impersonal trust*. Impersonal trust is not based on the trustee's personal attributes but rather on a social or institutional structure. For example, people have impersonal trust in systems such as the monetary or the judicial systems. The lack of consensus in the literature on the definition of trust has led researchers to use the terms trust and security interchangeably [73, 20, 5]. For an overview on delineating the boundaries of security and trust, please refer to Section 1.1. The definition of behavior trust used in this thesis is as follows:

> *Behavior trust is the firmness of 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.*

Behavior trust is quantified by a dynamic parameter called *trust level* (TL) that ranges from *very untrustworthy* to *very trustworthy*, and which is represented by a numeric range [1..5] as shown in Table 3.1. The TL is computed based on past experiences for a specific context. For example, based on trust, entity y might allow entity x to use its resources to store data files but not executable files. As we saw in Chapter 2, other researchers such as [5, 53] and the grandmother of trust, Baier [67], focused on a context-specific trust. Also in [53], peer groups' membership is motivated by *keyword*, which is very similar to the context concept. In our trust model, we borrow the context-specific idea when measuring the trust notion between entities. The TL is also specified for a given time frame because the TL today between two entities is not necessarily the same as the TL was a year ago.

Table 3.1: Description of the different trust levels.

Equivalent numerical value	Description
1	very untrustworthy
2	untrustworthy
3	medium trustworthy
4	trustworthy
5	very trustworthy

3.1.3 Reputation

In a dynamic setting, entities need to manage risks involved with interacting with other entities. In a dynamic environment, entities are vulnerable to risks because of unknown, incomplete, or distorted information about each other. One way to address this problem is to establish trust through reputation. The reputation concept is already used in other trust models, for example [5, 50, 51]. 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 about whether to engage in 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 thesis 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.*

Forming the reputation of an entity so that it is effective, informed, and reflects the entity's "trustworthiness" depends on two factors: (a) the honesty of the information source (i.e., the entity that sent the information) and (b) the accuracy of the information received. Therefore, the objective is to rely on entities that are honest as well as to rely on information that is accurate. Honesty and accuracy are defined next.

3.1.4 Honesty

Relying on *other entities* for information when seeking the reputation of entity y, entity x might be misinformed and form the wrong perception about y. This is due to dishonest recommenders that try to pollute the environment by intentionally giving bogus reputation reports. Ideally, we want to prevent dishonest entities from contributing to the computation of reputation. The definition of honesty used in this thesis is as follows:

> *A recommender x is said to be honest if the information, pertaining to a specific entity within a specific context at a given time, received from from entity x is the same information that entity x believes in.*

When entity x gives out information, x fetches this information from a data structure that x maintains. Hence, by "believes in", we mean the information that is stored in x's data structure. Honesty is a critical factor in any reputation-based trust model. For example, current trust models [56, 57, 31] assume that the majority of the entities are honest and therefore cancel the effect of dishonest ones on the recommendation network. Also in [5] Section 2.4.2 , it is assumed that an entity is equipped with honest entities in its Q and R, and entities are assumed to be robust and resilient to dishonest entities and risky environments. In [50], please refer to Section 2.4.1, an entity's credibility is used to offset the risk of dishonest feedback. In these approaches, no mechanism is used to identify and prevent dishonest entities from polluting the recommendation network. In our trust model, the goal is to come up with a mechanism to compute honesty and use this measure to weed out and prevent dishonest recommenders from influencing the recommendation network. The mechanism of achieving this goal is explained in Section 3.4.

3.1.5 Accuracy

The objective here is to ensure that the received information pertaining to entity y is as close as possible to the trustworthiness of entity y. The definition of accuracy used in this thesis 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 within a precision threshold.*

Other trust models use similar concepts to the accuracy notion used here. For example, as discussed in Chapter 2, the trust model in [5] uses the semantic distance as a measure that is applied to **the information received from a recommender** to infer **what the recommender really means**. In the trust model presented in [27], an entity scales a recommender's advertisement according to the entity's belief. This scaling process takes into account how accurate the recommender's advertisement is. Further, referring to the trust model outlined in [50], before entity x collects feedback from recommenders to compute the trustworthiness of entity y, x scales the feedback by the respective recommenders' credibility. In [50], the trustworthiness of an entity is used to approximate its credibility. That is, feedback from trustworthy entities are considered more credible and thus weigh more than feedback from untrustworthy entities. In our trust model, accuracy is an independent notion from an entity's trustworthiness and in Section 3.4, we present the mechanism of computing and quantifying accuracy.

3.2 Trust Model Elements

In our trust model, entity x maintains a *set of recommenders* (R_x) and a *set of trusted allies* (T_x). Entity x completely trusts the members of T_x that are chosen based on off-line

relationships. Trusted allies are used by an entity as part of a mechanism to determine the honesty of its recommenders. In general, members of T_x do not have the complete knowledge of all other entities needed to provide the recommendations themselves and so use recommenders to fill this gap. In addition, x maintains a *recommender trust table* (RTT_x) that associates a two-tuple (honesty, accuracy) with each recommender in R_x. Initial membership of R_x is randomly chosen and it evolves as described below. Similar to the RTT_x, x maintains another table called *direct trust table* (DTT_x) that includes trust levels for entities with which x had prior direct transactions. These two tables RTT_x and DTT_x are maintained by x's *trust agent* (TA_x). The data structures DTT and RTT, their initializations, and their usage are explained in more detail in Section 4.3.1.

3.3 Assumptions of the Trust Model

The following are the assumptions of the trust model. First, behavior trust is a slowly varying parameter, therefore, the update associated with the trust level is computed based on a *significant* amount of transactional data. Second, an entity has at least a few friendly entities that are perfectly trusted and this set is generated by off-line mechanisms. Third, transactions between entities are secure. In other words, data or information that was received is the data or information that was sent. Well known techniques such as encryption methods, checksums, or SSL can be used to ensure the integrity of data while in transit. Fourth, the source and destination are properly authenticated. For example, each entity periodically re-authenticates its set of trusted allies since if a trusted ally is usurped by a dishonest entity, this will compromise the trustworthiness of the whole system. Fifth, trustworthiness and honesty are independent notions, (i.e., an entity can be trustworthy and dishonest at the same time). It should be mentioned that the notion of independence between trustworthiness and honesty is not fully studied in this thesis.

3.4 Computing Honesty and Accuracy

Entity x randomly chooses its recommender z and initializes it as follows. First, z is considered to have maximum accuracy regardless of the target entity. That is, $A_x(z,t,c) = 1$ meaning that x considers z to have maximum accuracy for context c at time t. Second, since z is a new recommender and has given no recommendations so far, z's *recommendation error* as observed by x for context c at time t is set to zero (i.e., $\Psi_{RE_x}(z,t,c) = 0$). Third, z is considered consistent regardless of the target entity. That is, $C_x(z,t,c) = 1$ meaning that x considers z consistent for context c at time t. Entity x's objective is to keep only honest and accurate recommenders in R_x.

Honesty is used to continually adjust the membership of the recommender set such that it consists of the most honest recommenders. The aim here is to determine the honesty of a recommender z for a single (source,target) entity pair regarding a transaction between the source entity x and the target entity y. The honesty of an entity is evaluated using the trusted allies as follows. To determine the honesty of z, x instructs the entities in T_x to request recommendations from z regarding y for a specific context. These requests are launched such that they arrive at z as closely spaced in time as possible. Since we consider behavior trust to be slowly varying and since requests launched from the entities in T_x arrive at z as closely spaced in time as possible, we eliminate the possibility of obtaining different responses from z due to the evolution over time of z's belief. If z gives away largely different answers to the different entities in T_x, then z is considered to be inconsistent. Otherwise, z is consistent. If z is inconsistent, then z must be dishonest. Now, let us discuss the consistent case.

If z is consistent, then there are two possible cases. In the first case; z can give the same trust level it stores in its data structure (i.e. being honest) to the entities in T_x, in which case z is considered consistent and honest. In the second case; z can modify the trust level it stores in its data structure (i.e. being dishonest) but gives the same modified trust level to

the entities in T_x, in which case z is considered consistent and dishonest.

Let the consistency of recommender z as observed by x in giving recommendation regarding y for context c at time t be denoted by $C_x(z, y, t, c)$. Let $RE_k(z, y, t, c)$ denote the recommendation for entity y given by z to entity k for context c and time t, where $k \in T_x$. The recommendation $RE_k(z, y, t, c)$ is given from DTT_z and that is what z believes in as the reputation of y. The data structure DTT_z is explained in more detail in Section 4.3.1. Let $TL_{min}(x, z, y, t, c) = \min_{k \in T_x}\{RE_k(z, y, t, c)\}$ and $TL_{max}(x, z, y, t, c) = \max_{k \in T_x}\{RE_k(z, y, t, c)\}$. Let $\Delta_{RE_x}(z, y, t, c)$ denote the difference and be given by:

$$\Delta_{RE_x}(z, y, t, c) = TL_{max}(x, z, y, t, c) - TL_{min}(x, z, y, t, c) \tag{3.1}$$

The value of $\Delta_{RE_x}(z, y, t, c)$ will be less than a small value ϵ_{RE} if recommender z is consistent. Consequently, $C_x(z, y, t, c)$ is computed as follows: (It should be noted that after $C_x(z, y, t, c)$ is computed, it will be used to update z overall consistency regardless of the target entity (i.e., $C_x(z, t, c)$). Please, see Section 4.3.3).

$$C_x(z, y, t, c) = \begin{cases} 0 & \text{if } \Delta_{RE_x}(z, y, t, c) > \epsilon_{RE} \\ 1 & \text{otherwise} \end{cases}$$

If $C_x(z, y, t, c) = 0$, then z is dishonest and will be filtered out and prevented from influencing the recommendation network. If $C_x(z, y, t, c) = 1$, then z is consistent but may be dishonest. Another filter called the accuracy measure is used by our trust model to capture these consistent but dishonest recommenders and adjust their recommendations before using them to compute the reputation of y.

Seeking the reputation of y, x will ask z for recommendation regarding y for c at time t. Once z sends its recommendation (i.e., $RE_x(z, y, t, c)$) and before x can use $RE_x(z, y, t, c)$ to calculate the reputation of y, $RE_x(z, y, t, c)$ must be adjusted to reflect recommender z's accuracy. To achieve this objective, a *shift function* (S) is introduced that uses the

overall accuracy $A_x(z,t,c)$ (i.e., z's accuracy regardless of the target entity) to correct $RE_x(z,y,t,c)$. The shift function is defined as:

$$S(x,y,z,t,c) = \begin{cases} RE_x(z,y,t,c) + 4(1 - A_x(z,t,c)) & \text{if } \Psi_{RE_x}(z,t,c) < 0 \\ RE_x(z,y,t,c) - 4(1 - A_x(z,t,c)) & \text{if } \Psi_{RE_x}(z,t,c) \geq 0 \end{cases}$$

Let $ITL_x(y,t,c)$ denote the *instantaneous trust level* (ITL) of y obtained by x as a result of monitoring its current transaction with y for context c at time t. The ITL is determined based on a single transaction between x and y. In practice, the monitoring process can be done offline, online, or combining offline and online mechanisms. The *transaction monitor* (TM) proxy of x (i.e. TM_x proxy) determines the $ITL_x(y,t,c)$ of the transaction. Because a TM proxy is controlled by the associated entity, a transaction can be rated to have different trust levels by different TM proxies. TM proxies observe the transaction or the transaction records to determine whether any abuses have taken place. While the particular entities should be able to configure the TM proxies to define what conditions exactly cause a breach in the transaction contract, some example breaches include: (a) holding the resources for longer periods that initially requested, (b) trying to access protected local data, (c) instantiating illegal tasks on the resources, (d) reneging on promises to provide resources, and (e) going down during periods of peak usage [29]. One way of monitoring the transaction is to analyze the data gathered from *audit data* [78] generated by the operating system. Audit trails are particularly useful because they can be used to establish guilt of attackers. An audit trail is a record of activities on a system that are logged to a file in chronologically sorted order. Since almost all activities are logged on a system, it is possible to manually inspect these logs and detect untrustworthy entities. However, the incredibly large sizes of audit data generated make manual analysis undesirable. The audit data generated can be of the order of 100 Megabytes daily.

Monitoring the transactions in a real-time *Intrusion Detection Systems* (IDSs) [79, 80]

automate the cumbersome task of going through the rather jungle-like audit data trails. However, monitoring the transactions in a real-time fashion can cause significant overhead for trust computation. One way to reduce the overhead is to combine online and offline mechanisms in the monitoring process. For example, we can log the events and use an offline post mortem analysis to determine the trust level. Alternatively, the data for analysis can be gathered from *audit data* [78] generated by the operating system or a post-mortem analysis tool such as IDSs. Monitoring each transaction is an onerous task. Therefore, the monitoring process (i.e., obtaining $ITL_x(y, c, t)$ is done every nth transaction. After the transaction is over and if x carried out the monitoring process (i.e., $ITL_x(y, c, t)$ is obtained), then $\Psi_{RE_x}(z, y, t, c)$ can be calculated as shown in Equation 3.2, where $\Psi_{RE_x}(z, y, t, c)$ is z's recommendation error as observed by x when z gives recommendation regarding y for context c at time t.

$$\Psi_{RE_x}(z, y, t, c) = RE_x(z, y, t, c) - ITL_x(y, t, c) \tag{3.2}$$

The value of $|\Psi_{RE_x}(z, y, t, c)|$ is an integer value ranging from 0 to 4 because $RE_x(z, y, t, c)$ and $ITL_x(y, t, c)$ are in [1..5]. Notice, that $\Psi_{RE_x}(z, y, t, c)$ is computed if and only if $ITL_x(y, t, c)$ is obtained. Hence, the accuracy of z when giving recommendation regarding y for context c at time t as far as x is concerned (i.e., $A_x(z, y, t, c)$) can be defined by:

$$A_x(z, y, t, c) = -\frac{1}{4} |\Psi_{RE_x}(z, y, t, c)| + 1 \tag{3.3}$$

Note that, $A_x(z, y, t, c)$ is a real number in the interval $[0, 1]$. If $|\Psi_{RE_x}(z, y, t, c)| = 0$, $A_x(z, y, t, c) = 1$ implying that z has maximum accuracy as far as x is concerned. Inversely, if $|\Psi_{RE_x}(z, y, t, c)| = 4$, $A_x(z, y, t, c) = 0$ meaning that z is completely inaccurate to x about y as far as x is concerned. Note that $A_x(z, y, t, c) = 0$ if and only if $|\Psi_{RE_x}(z, y, t, c)| = 4$ meaning that: (a) $ITL_x(y, t, c) = 1$ and $RE_x(z, y, t, c) = 5$ or (b)

$ITL_x(y,t,c) = 5$ and $RE_x(z,y,t,c) = 1$. This is because the largest recommendation error results in the lowest accuracy and the smallest recommendation error results in the maximum accuracy

Note also that, $A_x(z,y,t,c)$ is the accuracy of z pertaining to the current particular transaction between x and y. Whereas, $A_x(z,t,c)$ is the overall accuracy of z, regardless of the target entity, as far as x is concerned. Finally: (a) $A_x(z,y,t,c)$ will be used to update z's overall accuracy (i.e. $A_x(z,t,c)$), and (b) $\Psi_{RE_x}(z,y,t,c)$ is used to update z's overall recommendation error (i.e., $\Psi_{RE_x}(z,t,c)$). The update procedures are explained in more detail in Section 4.3.3.

3.5 Computing Trust and Reputation

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 may apply 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 trust level 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 flexible to weigh the two components differently.

Let the behavior trust for a given context c and time t between two entities x and y be $\Gamma(x,y,t,c)$, direct trust between the entities for the same context and time be $\Theta(x,y,t,c)$, and the reputation of y for the same context and time be $\Omega(y,t,c)$.

The computation of $\Omega(y,t,c)$ is calculated by x asking all entities (excluding y) about

the reputation of y as illustrated in Equation 3.4, where U is the universe set of all entities.

$$\Omega(y,t,c) = \frac{\sum_{z \in U-\{x,y\}} \Upsilon(t-\tau_{zy}(t),c)\ S(x,y,z,t,c)}{|U-\{x,y\}|}, z \neq y \tag{3.4}$$

where U is the universe set of entities, Υ is the decay function (defined below), S is the shift function, and τ denotes the time of the last transaction between z and y (defined below in more detail). In practice no entity is omniscient and thus, the reputation of y is estimated by x based on the recommendations x receives from its recommenders. In general, this estimate will not be the same as $\Omega(y,t,c)$. However, as a simplification measure, we assume the estimate to be sufficiently accurate. Let the weights given to direct trust and reputation be α and β, respectively such that $\alpha + \beta = 1$ and $\alpha, \beta \geq 0$. If the trustworthiness of y, as far as x is concerned, is based more on direct trust than the reputation of y, then α should be larger than β.

In the trust model, trust is allowed to change with time. For example, if x trusted y at a given level five years ago, x's trust in y now is likely to be lower unless x and y have continued to interact since then. To model this aspect, we multiply the trust levels in DTT_x by a *decay function* ($\Upsilon(t-\tau_{xy}(t),c)$), where c is the context, t the current time, and $\tau_{xy}(t)$ is the time of the last transaction between x and y.

$$\Gamma(x,y,t,c) = \alpha\ \Theta(x,y,t,c) + \beta\ \Omega_x(y,t,c) \tag{3.5}$$

$$\Theta(x,y,t,c) = \Upsilon(t-\tau_{xy}(t),c)\ DTT_x(y,t,c) \tag{3.6}$$

The reputation of y is computed as the average of the product of the trust level in the DTT shifted by the *shift function* S and the *decay function* ($\Upsilon(t-\tau_{zy}(t),c)$), for all recommenders $z \in R$ and $z \neq y$. In practical systems, entities will use the same information to

evaluate direct trust and give recommendations, i.e., DTT will be used to give recommendations as well as for obtaining the direct trust level.

$$\Omega_x(y,t,c) = \frac{\sum_{z \in R_x} \Upsilon(t - \tau_{zy}(t), c)\, S(x,y,z,t,c)}{|R_x|}, z \neq y \quad (3.7)$$

3.6 Trust Transaction Example

3.6.1 Overview

Before presenting the numerical example to illustrate how the behavior trust operates, we provide the following pseudo-code segments to show the sequence of the steps a source entity x takes to determine the trustworthiness of a target entity y. For the sake of the reader and for an easy reference, table 3.2 summarizes the behavior trust terms used by x.

Before x engages in any behavior trust relationships with any target entity, x performs an initialization steps to choose its set of trusted allies as well as its set of recommenders. Entity x also performs further steps to initialize its set of recommenders as illustrated in Figure 3.1. In lines **(1)** and **(2)**, x chooses its set of trusted allies (i.e., T_x) based on off-line relationships and randomly chooses its set of recommenders (i.e., R_x). In lines **(3)** through **(6)**, x initializes accuracy and consistency of its recommenders.

Now, to determine the trustworthiness of y, x uses the steps illustrated in Figure 3.2. It should be mentioned that x performs the consistency check every mth transaction and the accuracy check every nth transaction. Also, trans_num means the transaction number (i.e.,1st transaction, 2nd transaction, etc.). In line **(1)**, x computes its direct trust in y and the details of this step are illustrated in Figure 3.3. In line **(2)**, x requests and gets recommendations from all of its recommenders in R_x. As illustrated in line **(3)**, if the consistency check needs to be performed, x carries that in line **(4)** and detailed in Figure 3.4. Before using the recommendations to compute the reputation of y, x adjusts the recommendations

Table 3.2: The behavior trust terms used by entity x.

Behavior trust term	meaning
$A_x(z, y, t, c)$	Accuracy of recommender z as observed by x in giving recommendation regarding y at time t for context c
$A_x(z, t, c)$	Accuracy of recommender z as observed by x at time t for context c
$C_x(z, y, t, c)$	Consistency of recommender z as observed by x in giving recommendation regarding y at time t for context c
$C_x(z, t, c)$	Consistency of recommender z as observed by x at time t for context c
$\Psi_{RE_x}(z, y, t, c)$	Recommendation error of recommender z as observed by x in giving recommendation regarding y at time t for context c
$\Psi_{RE_x}(z, t, c)$	Recommendation error of recommender z as observed by x at time t for context c
$RE_x(z, y, t, c)$	The recommendation for y given by recommender z to x at time t for context c
$ITL_x(y, t, c)$	The instantaneous trust level of y obtained by x as a result of monitoring its current transaction with y at time t for context c
$S(x, z, t, c)$	The shift function used by entity x to adjust the recommendation given by recommender z at time t
$\Gamma(x, y, t, c)$	Behavior trust between x and y at time t for context c for context c
$\Theta(x, y, t, c)$	Direct trust between x and y at time t for context c
$\Omega_x(y, t, c)$	Reputation of y as observed by x at time t for context c

received from its recommenders in R_x. This step is shown in line **(5)** and detailed in Figure 3.5. In lines **(6)** and **(7)**, x computes the reputation of y and the behavior trust of y, respectively. Lines **(8)** through **(12)** are self explanatory. The update process carried out by x after the transaction is over is shown in Line **(13)** and detailed in Figure 3.6.

The loop, Figure 3.4, in line **(1)** through line **(8)** carries out the consistency check. As a consequence, the inconsistent and hence must be dishonest recommenders are identified

The initialization steps performed by entity x.

(1) ;; Based on off-line relationships, x chooses its set of trusted allies T_x
(2) ;; x randomly chooses its set of recommenders R_x
(3) **for** each recommender $z \in R_x$ **do**
(4) $A_x(z,t,c) \leftarrow 1$;; z is considered to have maximum accuracy
(5) $C_x(z,t,c) \leftarrow 1$;; z is considered to be consistent
(6) $\Psi_{RE_x}(z,t,c) \leftarrow 0$;; since z is considered to have maximum accuracy, then recommendation error of z is set to 0

Figure 3.1: Source entity x initializes its recommenders in R_x.

Source entity x computes the behavior trust in target entity y.

(1) **compute** $\Theta(x,y,t,c)$;; as detailed in Figure 3.3
(2) **get** $RE_x(z,y,t,c)$, $\forall z \in R_x$;; x gets the recommendations regarding y
(3) **if** ((trans_num **mod** m) $= 0$)
(4) **consistency** check ;; as detailed in Figure 3.4
(5) **adjust** recommendations ;; as detailed in Figure 3.5
(6) $\Omega_x(y,t,c) \leftarrow \frac{\sum_{z \in R_x} \Upsilon(t-\tau_{zy}(t),c)\ S(x,y,z,t,c)}{|R_x|}, z \neq y$;; as illustrated in Equation 3.7
(7) $\Gamma(x,y,t,c) \leftarrow \alpha\ \Theta(x,y,t,c) + \beta\ \Omega_x(y,t,c)$;; as illustrated in Equation 3.5
(8) **if** x decides to proceed with the transaction
(9) **if** ((trans_num **mod** n) $==$ 0) ;; accuracy check
(10) **obtain** $ITL_x(y,t,c)$
(11) **else**
(12) x rejects the transaction with y
(13) **update** process ;; as detailed in Figure 3.6

Figure 3.2: Behavior trust steps.

and removed from T_x by setting their consistency to zero. The consistent recommenders have their consistency set to one.

Source entity x computes direct trust in target entity y.

(1) ;; get $DTT_x(y,t,c)$
(2) ;; decay $DTT_x(y,t,c)$ if necessary
(3) ;; compute the direct trust $\Theta(x,y,t,c)$ as illustrated in Equation 3.6
(4) $\Theta(x,y,t,c) \leftarrow \Upsilon(t-\tau_{xy}(t),c)\ DTT_x(y,t,c)$

Figure 3.3: computation of $\Theta(x,y,t,c)$.

Source entity x performs the consistency check to filter dishonest recommenders.

(1) **forall** $z \in R_x$ **do**
(2) **get** $RE_k(z,y,t,c),\ \forall k \in T_x$
(3) $\Delta_{RE_x}(z,y,t,c) \leftarrow \max_{k\in T_x}\{RE_k(z,y,t,c)\} - \min_{k\in T_x}\{RE_k(z,y,t,c)\},\ \forall k \in T_x$
(4) **if** $\Delta_{RE_x}(z,y,t,c) > \epsilon_{RE}$;; z_i inconsistent and hence must be dishonest
(5) $C_x(z,y,t,c) \leftarrow 0$
(6) $|R_x| = |R_x| - 1$;; remove z from R_x
(7) **else** ;; z consistent but may be dishonest
(8) $C_x(z,y,t,c) \leftarrow 1$

Figure 3.4: Consistency check performed by x.

Figure 3.5 illustrates how x adjusts the recommendations received from consistent recommenders. Lines **(2)** through **(5)** details the explicit steps carried out by the shift function as explained in Section 3.4. Figure 3.6 shows the update steps that x performs after the transaction with y is over. The algorithms used to perform the various updates are discussed in more details in Section 4.3.3. Line **(1)** starts a loops that goes through all the recommenders in R_x. In line **(2)**, if the consistency check was performed by x, then x updates the overall consistency of its recommender z by using z's current consistency in recommending y. Similarly, if the accuracy check was performed by x, then x updates the

Source entity x adjusts z's recommendation to reflect z's accuracy.

(1) **forall** $z \in R_x$ **do**
(2) **if** ($\Psi_{RE_x}(z,t,c) < 0$) ;; computed in line **(0)**, Figure 3.1
(3) $S(x,y,z,t,c) \leftarrow RE_x(z,y,t,c) + 4(1 - A_x(z,t,c))$
(4) **else**
(5) $S(x,y,z,t,c) \leftarrow RE_x(z,y,t,c) - 4(1 - A_x(z,t,c))$

Figure 3.5: Adjusting the recommendations made by x's recommenders.

overall accuracy of z using z's current accuracy in recommending y.

Source entity x updates the consistency and accuracy of its recommenders.

(1) **forall** $z \in R_x$ **do**
(2) **if** ((trans_num **mod** m) $= 0$) ;; consistency check
(3) **update** $C_x(z,t,c)$ using $C_x(z,y,t,c)$
(4) **if** ((trans_num **mod** n) $= 0$) ;; accuracy check
(5) **update** $DTT_x(y,t,c)$ using $ITL_x(y,t,c)$
(6) $\Psi_{RE_x}(z,y,t,c) \leftarrow RE_x(z,y,t,c) - ITL_x(y,t,c)$
(7) **update** $\Psi_{RE_x}(z,t,c)$ using $\Psi_{RE_x}(z,y,t,c)$
(8) $A_x(z,y,t,c) \leftarrow -\frac{1}{4}\ |\Psi_{RE_x}(z,y,t,c)| + 1$
(9) **update** $A_x(z,t,c)$ using $A_x(z,y,t,c)$
(10) trans_num++

Figure 3.6: Update process.

3.6.2 Example

Figure 3.7 shows a block diagram of the different components of the trust model. The figure shows an example of a source entity x wanting to engage in a transaction with a target entity y. The example, provided in this section, illustrates the evaluation of direct as well as reputation trust relationships in the context c of a "printing service" where x provides some of its resources to other entities for printing purposes. In such a transaction, there is a trust concern from the source entity as well as the target entity. As shown in the figure, each entity has its local elements. For example, x has its own T_x, R_x, DTT_x, RTT_x, TM_x proxy, and TA_x. Each entity should have its own trusted allies set and set of recommenders. For the sake of simplicity, we provide a walk through example of x determining the trustworthiness of y. Hence, the figure shows only the trusted allies set and set of recommenders of x, namely T_x and R_x.

Let us assume that entity y wants to print its annual report using x's resources. Entity x is concerned about y's trustworthiness and hesitant to provide its service and allow y to use its resources. Some breaches that y can commit while using x's resources include: (a) holding the resources for longer periods that initially requested, (b) trying to access protected local data, and (c) instantiating illegal tasks on the resources. To make a decision about whether to have a transaction with y, x can rely on its own past experience with y as well as on y's reputation. This is expressed in Equation 3.5.

As stated in Section 3.2 each entity maintains a set of trusted allies as well as a set of recommenders. Hence, x maintains T_x and R_x. Suppose that x has two trusted allies a_1 and a_2 in T_x as well as three recommenders r_1, r_2, and r_3 in R_x. The trusted allies in T_x are completely trusted by x and are chosen based on off-line relationships. However, the recommenders r_1, r_2, and r_3 in R_x are randomly chosen and initially these recommenders are considered by x to be honest and completely accurate. That is, once r_1, r_2, and r_3 are randomly chosen by x, their consistency and accuracy are initialized as follows:

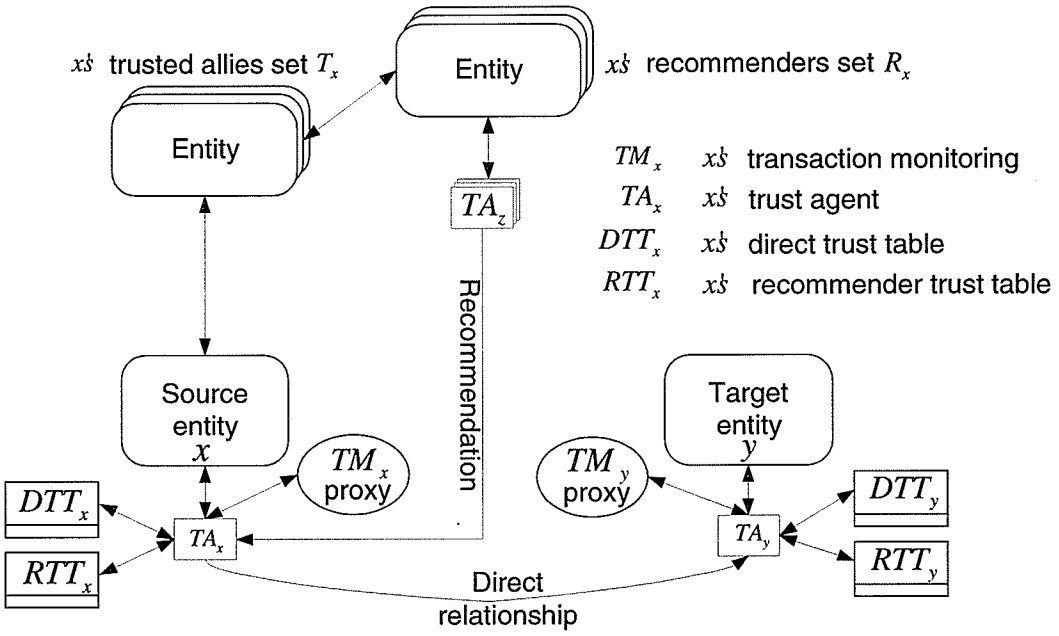


Figure 3.7: The different components of the trust model.

$C_x(r_1, t, c) = 1$ because since r_1 is honest, r_1 is consistent, since z is a new recommender and gave no recommendations to x so far, z's recommendation error as observed by x for context c at time t is set to zero (i.e., $\Psi_{RE_x}(r_1, t, c) = 0$), and $A_x(r_1, t, c) = 1$. Similarly, $C_x(r_2, t, c) = 1$, $A_x(r_2, t, c) = 1$, $\Psi_{RE_x}(r_2, t, c) = 0$, $C_x(r_3, t, c) = 1$, $A_x(r_3, t, c) = 1$, $\Psi_{RE_x}(r_3, t, c) = 0$. We also assume that r_1, r_2, and r_3 have previous interactions with y.

First, Let us illustrate how can x discover and quantify y's reputation. That is, let us show how to compute $\Omega_x(y, t, c)$. To determine the reputation of y, x consults its recommenders in R_x to give recommendations regarding y for context c. Since $C_x(r_1, t, c) = 1$ and $C_x(r_2, t, c) = 1$, x decides to keep both of r_1 and r_2 in R_x. Note that, as long as the recommenders are consistent, the first measure used by our trust model, namely consistency, will filter out the inconsistent and hence dishonest recommenders. Then, the second

measure used by our trust model, namely accuracy, assures that if there is any bias in the recommendations received about y's trustworthiness, this bias is minimized and brought as close as possible to the trustworthiness of y. Please refer to Section 3.4 for clarification.

Now, x sends recommendation requests to r_1, r_2, and r_3 to obtain y's reputation. After consulting DTT_{r_1}, r_1 finds that it has a trust level of 3 based on recent interactions with y for context c at time t. Consulting DTT_{r_2}, r_2 finds that it has a trust level of 5 based on interactions done two years ago with y for context c. Consulting DTT_{r_3}, r_3 finds that it has a trust level of 5 based on interactions done recently with y for context c. Assume that x carries out the consistency check and instructs the entities in T_x to request recommendations from r_1, r_2, and r_3 regarding y for c. Suppose that: (a) r_1 gives 3 to a_1 and a_2, (b) r_2 gives 5 to a_1 and a_2, and (c) r_3 gives 3 and 5 to a_1 and a_2, respectively. Calculating $\Delta_{RE_x}(r_1, y, t, c)$ as illustrated in Equation 3.1 yields $\Delta_{RE_x}(r_1, y, t, c) = 0$. Similarly, $\Delta_{RE_x}(r_2, y, t, c) = 0$, whereas $\Delta_{RE_x}(r_3, y, t, c) = 2$. Obviously, $\Delta_{RE_x}(r_3, y, t, c) > \epsilon_{RE}$ and hence $C_x(r_3, y, t, c) = 1$. Therefore, r_3 is considered inconsistent and hence dishonest. As a result, r_3 is removed from R_x.

Before x uses $RE_x(r_1, y, t, c)$ and $RE_x(r_2, y, t, c)$ to compute the reputation of y, x assures that if there is any bias in the recommendations received about y's trustworthiness, this bias is minimized and brought as close as possible to the trustworthiness of y (i.e., x has to use the recommender's accuracy to adjust or shift the recommendation). After receiving the recommendation from r_1 and before the transaction between x and y takes place, x has the following about r_1: $RE_x(r_1, y, t, c) = 3$ and $A_x(r_1, t, c) = 1$. Keep in mind that $ITL_x(y, t, c)$ will be determined during or after the transaction. In other words, before the transaction with y, x does not know the value of y's ITL. Since $A_x(r_1, y, t, c)$ and $\Psi_{RE_x}(r_1, y, t, c)$ depend on y's ITL, they will be computed if and only if y's ITL is obtained. Similarly, after receiving the recommendation from r_2 and before the transaction between x and y takes place, x has the following about r_2: $RE_x(r_2, y, t, c) = 5$, $A_x(r_2, t, c) = 1$. Similarly, $\Psi_{RE_x}(r_2, y, t, c)$ and $A_x(r_2, y, t, c)$ will be computed if and only if y's ITL is

obtained. Therefore, $RE_x(r_1, y, t, c)$ and $RE_x(r_2, y, t, c)$ are shifted as follows:

$$S(x, y, r_1, t, c) = 3 - 4(1 - 1) = 3 \quad (3.8)$$

$$S(x, y, r_2, t, c) = 5 - 4(1 - 1) = 5 \quad (3.9)$$

Since, r_1 has maximum accuracy as far as x is concerned, then the recommendation given by r_1 will not be shifted and will be taken as is by x. The recommendation given to x by r_1 regarding y is 3. After applying the shift function to the recommendation to reflect r_1's accuracy, indeed x takes exactly what r_1 recommends and this is shown in Equation 3.8. Similar observation can be drawn on the recommendation received from r_2. Now, x is ready to compute y's reputation according to Equation 3.7. Entity x needs to decay the recommendations obtained from r_1 and r_2 to reflect the time elapsed since these recommendations were obtained. One implementation of the decay function is to make $\Upsilon(t - \tau_{r_1 y}(t), c)$ a real number in the interval [0, 1] such that 0 means extreme decay and 1 means no decay. For example, $\Upsilon(t - \tau_{r_1 y}(t), c)$because the recommendation given by r_1 regarding y is based on recent interactions. Hence, x does not decay the recommendation from r_1 and takes it as is. However, since the recommendation given by r_2 regarding y is based on interactions done two years ago, then x needs to decay this recommendation. Assume that x set the decay function $\Upsilon(t - \tau_{r_2 y}(t), c)$ to 0.4. Therefore, the reputation of y $\Omega_x(y, t, c)$ as illustrated in Equation 3.7, is computed as:

$$\Omega_x(y, t, c) = \frac{(1 \times 3) + (0.4 \times 5)}{2} = 2.5 \quad (3.10)$$

To compute the direct trust x has in y, x consults DTT_x through TA_x as shown in Figure 3.7. Let us assume that x has directly interacted with y a year ago and the trust level is 5. Since this direct trust level was based on interactions one year ago, $\Upsilon(t - \tau_{r_1 x}(t), c)$ might

be assigned 0.7. According to Equation 3.6:

$$\Theta(x, y, t, c) = 0.7 \times 5 = 3.5 \tag{3.11}$$

Assume that $\alpha = 0.8$ and $\beta = 0.2$ meaning that x relies more on its direct trust than reputation, then finally x is ready to calculate the trust in y,

$$\Gamma(x, y, t, c) = 0.8 \times 3.5 + 0.2 \times 2.5 = 3.3 \tag{3.12}$$

According to Table 3.1, the behavior trust evaluation of y at trust level 3.3 says that y is medium trustworthy. Suppose that x decides to have the transaction with y and the transaction is monitored by TM_x proxy as shown in Figure 3.7. Suppose that $ITL_x(y, t, c)$ is found by TM_x proxy to be equal to 3.

After the transaction and since x monitored its transaction with y, the accuracy check can be performed. The accuracy of the recommenders based on the current transaction are computed as shown in Table 3.3. Based on the current transaction and after computing the accuracy and the recommendation error, x does the following:

- Updates its direct trust in y by using $ITL_x(y, t, c)$ to update the corresponding entry in DTT_x (i.e., $DTT_x(y, t, c)$).

- Uses $C_x(r_1, y, t, c)$ and $C_x(r_2, y, t, c)$ to update $C_x(r_1, t, c)$ and $C_x(r_2, t, c)$, respectively.

- Uses $\Psi_{RE_x}(r_1, y, t, c)$ and $\Psi_{RE_x}(r_2, y, t, c)$ to update $\Psi_{RE_x}(r_1, t, c)$ and $\Psi_{RE_x}(r_2, t, c)$, respectively.

- Uses $A_x(r_1, y, t, c)$ and $A_x(r_2, y, t, c)$ to update $A_x(r_1, t, c)$ and $A_x(r_2, t, c)$, respectively.

The next chapter will have more details on the update algorithms used to update direct trust, recommenders' honesty, recommendation error, and recommenders' accuracy.

Table 3.3: Accuracy computation for recommenders in T_x.

Recommender	Recommendation error	Recommender's accuracy
r_1	$\Psi_{RE_x}(r_1, y, t, c) \leftarrow RE_x(r_1, y, t, c) - ITL_x(y, t, c)$ $\Psi_{RE_x}(r_1, y, t, c) \leftarrow 3 - 3 = 0$	$A_x(r_1, y, t, c) \leftarrow -\frac{1}{4} \lvert\Psi_{RE_x}(, r_1, y, t, c)\rvert + 1$ $A_x(r_1, y, t, c) \leftarrow 1$
r_2	$\Psi_{RE_x}(r_2, y, t, c) \leftarrow RE_x(r_2, y, t, c) - ITL_x(y, t, c)$ $\Psi_{RE_x}(r_2, y, t, c) \leftarrow 5 - 3 = 2$	$A_x(r_2, y, t, c) \leftarrow -\frac{1}{4} \lvert\Psi_{RE_x}(z, y, t, c)\rvert + 1$ $A_x(r_2, y, t, c) \leftarrow 0.5$

3.7 Summary

In this chapter, we introduced the fundamental concepts used in our behavior trust model and presented the basic rules for quantifying them. The fundamental concepts and rules have been presented and published at some conferences [63, 39, 61].

Behavior trust is quantified by trust level, which is a dynamic parameter ranging from very trustworthy to very untrustworthy. The trust level is computed based on direct trust as well as reputation for a specific context at a given time. Direct trust is the behavior trust level resulting from an entity's own past experience with the target entity. Ideally, reputation of an entity is the behavior trust level reached by global consensus. In practice, it is estimated by polling recommenders regarding the behavior trust of the target entity.

Because recommenders can be dishonest and distort the reputation estimates, we introduced a consistency concept that tracks the truthfulness of a recommender. A recommender is considered to be truthful if it says what it actually knows. We track the truthfulness or the honesty of a recommender by applying a consistency measure. If a recommender gives away largely different answers to the different entities in T_x, then the recommender is considered to be inconsistent and hence dishonest. Otherwise, the recommender is consistent. If the recommender is consistent, then the recommender might be: (a) honest and that is what we want to achieve, or (b) dishonest by modifying the truth but giving the same modified truth to all the entities in T_x.

In our trust model, we introduced an accuracy measure to reflect a recommender's accuracy: (a) to further prevent these consistent but dishonest recommenders from influencing the recommendation network, and (b) to adjust the recommendations received from honest recommenders. This is done before the recommendations received are used to compute the reputation of the target entity. In a nutshell, the accuracy measure is an important concept that tracks how correctly a recommender estimates the underlying trust level of the target entity.

Further, Our trust model has the flexibility to weigh direct trust and reputation differently. Having the flexibility of combining direct trust and reputation gives the trust model the leeway to choose the strategy that best fits it. Finally, to illustrate the usage of concepts used in our trust model and to explain how the behavior trust operates, we provided an application example. The example depicts how behavior trust is determined between a source entity and a target entity.

Chapter 4

Mapping the Trust Model onto Network Computing Systems

4.1 Overview of Network Computing Systems

The deployment of faster networking infrastructures and the availability of powerful microprocessors have positioned *Network Computing* (NC) systems as a cost-effective alternative to traditional computing approaches. The NC system can be grouped into various categories depending on the extent of the system and the performance of the interconnection media. For example, clusters of workstations are NC systems that use commodity networks to create dedicated very tightly coupled systems. Another example of NC systems is the meta-computing initiatives on the Internet that attempt to harness the available resources to perform complex parallel applications such as prime number sieves. Motivated by the successes of NC systems, researchers have started examining more generalized resource/information sharing and integration infrastructures such as Grid [41, 81, 60] and P2P [21, 22, 26] systems.

A Grid system is defined as a generalized, NC and data handling virtual system that

is formed by aggregating the services provided by several distributed resources [82, 48, 83, 84, 85]. A Grid can potentially provide pervasive, dependable, consistent, and cost-effective access to the diverse services provided by the distributed resources and support problem solving environments that may be constructed using such resources.

On the other hand, P2P systems are a way of organizing NC systems and they are a manifestation of one of the fundamental design principles of the Internet [86]. Recently, popular file-sharing applications such as KaZZa [22] and Gnutella [28, 21] have rekindled interest in this approach to large-scale system design. The P2P approach has significant benefits including scalability, low cost of ownership, robustness, and ability to provide site autonomy. Although current P2P systems are mainly concerned with file swapping applications, the concept can be generalized to build different large-scale NC systems [24, 25, 26]. To achieve this generalization, NC systems' scalability becomes a vital concern.

For scalability, large scale NC systems can be considered as a set of interconnected domains. These domains can interact in a P2P fashion to share resources and services amongst themselves. One primary goal of NC 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) ensuring 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.

Dividing NC systems into domains or regions that could be diverse along one or more dimensions such as addressing, performance, or trust is suggested in [87]. This division introduces a new building block to connect domains and adds an additional layer called *trust enforcement layer* [87]. Integrating trust into NC systems introduces trust awareness that enables the isolation of different resource pools and client pools into "trusted

domains." Such trusted domains increase and encourage more business-to-business applications which in turn can create new forms of service models.

Another goal behind mapping trust onto NC systems is the capability of addressing the needs of a wide area networked system where different nodes (clients or resources) interact by sharing resources and/or services. Building a trust model for these distributed heterogeneous domains is a complex task. The distribution nature of such an environment plays a primary factor in the design complexity for such systems in various ways: (a) the trust model needs to be designed in a distributed architecture fashion and (b) the scalability becomes a major issue for the success of such a trust model.

In wide area networked systems, we can not assume that every node knows about every other node nor can we assume that all the nodes are trustworthy. Therefore, another goal of the trust model is the ability to:

- Evolve and establish trust relationships between nodes prior to any cooperation.
- Manage and maintain existing trust relationships.
- Be resilient to dishonest nodes that for malicious reasons give incorrect reviews.

4.2 Aggregating Network Computing Systems

A straightforward approach to mapping the trust model would be to consider each node (resource or client) of the networked system as an entity of the trust model. Such an approach is less desirable from the scaling point-of-view because a networked system can have a large number of nodes. To reduce the number of nodes, the networked system is aggregated into *network computing domains* (NCDs) and the trust model is required to maintain an entity for each NCD as shown in Figure 4.1. The aggregation, however, has some interesting side-effects beyond improving scalability. Before examining these effects, we present the assumptions and methodologies used for the aggregation.

The aggregation process elects one or more leaders for a given NCD. Because the leader is representing the whole NCD in the trust modeling process, it is held responsible for any violations by the NCD members. The leader is expected to manage the member nodes to maintain a high reputation for its NCD within the global community.

For example, consider an NCD formed by five physical nodes. Let one of the nodes be the leader. The leader is assumed to have the complete trust of the rest of the nodes within the NCD. Although the leader in general, does not trust the other members at a uniform level, in this example, we assume that the leader trusts the nodes at a uniform high level. Suppose a new node is interested in joining the NCD; it would negotiate with the leader regarding the trust level that would be bestowed upon it. Ideally, the joining node would like to be trusted at the highest possible level and may lay claim based on references from prior associations. The leader has conflicting requirements: (a) it wants to present its resources as highly trustworthy to the outside NCDs because the value of the highly trustworthy resources will be much higher than the resources that are less trustworthy and (b) it does not want to overestimate the trustworthiness of a resource because its own reputation will suffer if the resource turns out to be less trustworthy. This conflict drives the leader to make choices that are as close to optimal as possible.

As shown in Figure 4.1, each NCD is a collection of nodes (clients and resources). These nodes collectively contribute to the trustworthiness of their NCD. Each NCD has a TM_{NCD} proxy that monitors the NCD-level transactions with other NCDs. The trust levels that an NCD believes of other NCDs directly interacted with are kept in DTT_{NCD}, while consistency and accuracy of the NCD's recommenders are kept in RTT_{NCD}. These two data structures, associated with each NCD, are maintained by TA_{NCD}. For a transaction between two NCDs, the actual interaction occurs at the node level as illustrated in Figure 4.1.

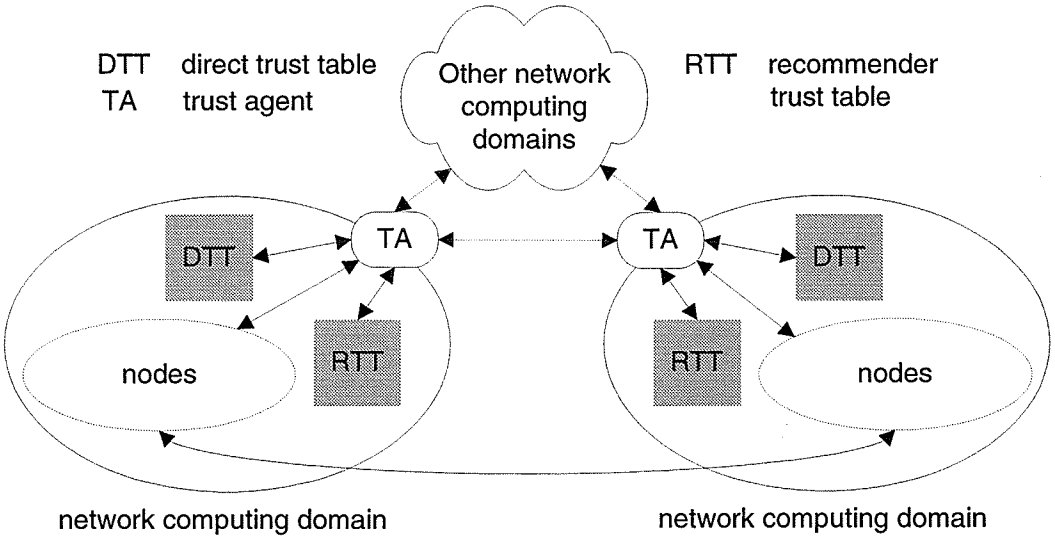


Figure 4.1: Block diagram of the overall network computing system trust model.

4.3 Mechanisms for Mapping Trust

4.3.1 Trust Representation and Usage

One important aspect of mapping trust is the mechanism for representing trust that exists among the NCDs. Table 4.1 shows an example DTT_{NCD_s} that depicts how a given NCD_s trusts other NCDs. For a specific context c_i, NCD_s trusts NCD_j at trust level $TL_{sj}^{c_i}$ and this trust level is based on direct experience with NCD_j. If NCD_j is *unknown* to NCD_s, the trust level $TL_{sj}^{c_i}$ is set to -1 and this denotes that NCD_s did not have a prior direct relationship or transaction with NCD_j. The trust levels in the DTT_{NCD_s} are time stamped to indicate the time of the last update. For example, $T_{sj}^{c_i}$ denotes the timestamp of the last update of the direct trust level between NCD_s and NCD_j for context c_i.

An NCD_s has a set of recommenders (i.e R_{NCD_s}) providing it with recommendations. This information is kept in RTT_{NCD_s} as shown in Table 4.2. For each recommender in R_{NCD_s}, NCD_s maintains the recommender's consistency as well as the recommender's accuracy. That is, NCD_s's objective is to filter out inconsistent and hence dishonest recommenders in its R_{NCD_s} and also to know how accurate is the recommendations given by the consistent recommenders. Let $C_{RTT_{NCD_s}}(NCD_z, t, c)$ denote the consistency of NCD_z kept in RTT_{NCD_s} for context c at time t. Let $A_{RTT_{NCD_s}}(NCD_z, t, c)$ denote the accuracy of NCD_z kept in RTT_{NCD_s} for context c at time t. Initially, NCD_s randomly selects its recommenders and considers them to be consistent and have maximum accuracy. Let $TL^{c_i}_{sj,c}$ denote the consistency that NCD_s assigns to recommender NCD_j for a specific context c_i and let $T^{c_i}_{sj,c}$ denote the timestamp of the last update of consistency that NCD_s assigns to NCD_j for specific context c_i. Similarly, let $TL^{c_i}_{sj,a}$ denote the accuracy that NCD_s assigns to recommender NCD_j for a specific context c_i and let $T^{c_i}_{sj,a}$ denote the timestamp of the last update of accuracy that NCD_s assigns to NCD_j for specific context c_i. For example, assume that NCD_s chooses NCD_j as one of its recommenders for context c_i. As explained in Section 3.4, since NCD_j is considered consistent by NCD_s, then, $TL^{c_i}_{sj,c}$ is set to 1 and $T^{c_i}_{sj,c}$ is set to the time of the last update. Similarly, since NCD_j is considered by NCD_s to have maximum accuracy, then $TL^{c_i}_{sj,a}$ is set to 1 and $T^{c_i}_{sj,a}$ is set to the time of the last update. These two data structures, namely DTT_{NCD_s} and RTT_{NCD_s}, are maintained by TA_{NCD_s}.

Table 4.1: An example of a direct trust table maintained by NCD_s.

Context	Network Computing Domains				
	NCD_1		...	NCD_j	
	Trust Level	Timestamp	...	Trust Level	Timestamp
c_1	$TL^{c_1}_{s1}$	$T^{c_1}_{s1}$	...	$TL^{c_1}_{sj}$	$T^{c_1}_{sj}$
⋮	⋮	⋮	⋮	⋮	⋮
c_i	$TL^{c_i}_{s1}$	$T^{c_i}_{s1}$	...	$TL^{c_i}_{sj}$	$T^{c_i}_{sj}$

Table 4.2: An example of a recommender trust table maintained by NCD_s.

C	Set of Recommenders									
	NCD_1				...	NCD_j				
	Consistency		Accuracy		...	Consistency		Accuracy		
	Trust Level	Timestamp	Trust Level	Timestamp	...	Trust Level	Timestamp	Trust Level	Timestamp
c_1	$TL^{c_1}_{s1,c}$	$T^{c_1}_{s1,c}$	$TL^{c_1}_{s1,a}$	$T^{c_1}_{s1,a}$	...	$TL^{c_1}_{sj,c}$	$T^{c_1}_{sj,c}$	$TL^{c_1}_{sj,a}$	$T^{c_1}_{sj,a}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
c_i	$TL^{c_i}_{s1,c}$	$T^{c_i}_{s1,c}$	$TL^{c_i}_{s1,a}$	$T^{c_i}_{s1,a}$	...	$TL^{c_i}_{sj,c}$	$T^{c_i}_{sj,c}$	$TL^{c_i}_{sj,a}$	$T^{c_i}_{sj,a}$

Suppose a resource belonging to NCD_s is interested in engaging in a transaction with a remote resource in NCD_t. To determine the suitability of the candidate remote resource, the NCD_s consults: (a) DTT_{NCD_s} to obtain its direct trust level with NCD_t and (b) its recommenders in RTT_{NCD_s} to obtain the reputation of NCD_t. Let NCD_r be one of the recommenders contacted by NCD_s. NCD_r will consult DTT_{NCD_r} to find out whether it had any prior transactions with NCD_t. If NCD_r had no prior transactions with NCD_t, then NCD_r will request its set of recommenders from RTT_{NCD_r} to determine NCD_t's reputation. Therefore, the recommendation requests can form a *recommender tree* as illustrated in Figure 4.2. Because we assume that the trust network among the NCDs are connected, a recommendation tree exists for any given source and target NCDs. Leaf nodes in a recommendation tree reply to their parent nodes with the trust level in their DTTs concerning NCD_t. A recommendation tree has DTT lookups at its leaf nodes and DTT plus RTT lookups at the intermediate nodes. To avoid cycles in the recommender tree, a recommendation request carries the list of visited NCDs. Further, the recommendation request carries a *time to live* field to determine the maximum number of NCDs it is allowed to visit before it is discarded.

Table 4.3: An example of a global direct trust table.

Network Computing Domains	Network Computing Domains				
	NCD_1		...	NCD_j	
	Trust Level	Timestamp	...	Trust Level	Timestamp
NCD_1	TL_{11}	T_{11}	...	TL_{1j}	T_{1j}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
NCD_i	TL_{i1}	T_{i1}	...	TL_{ij}	T_{ij}

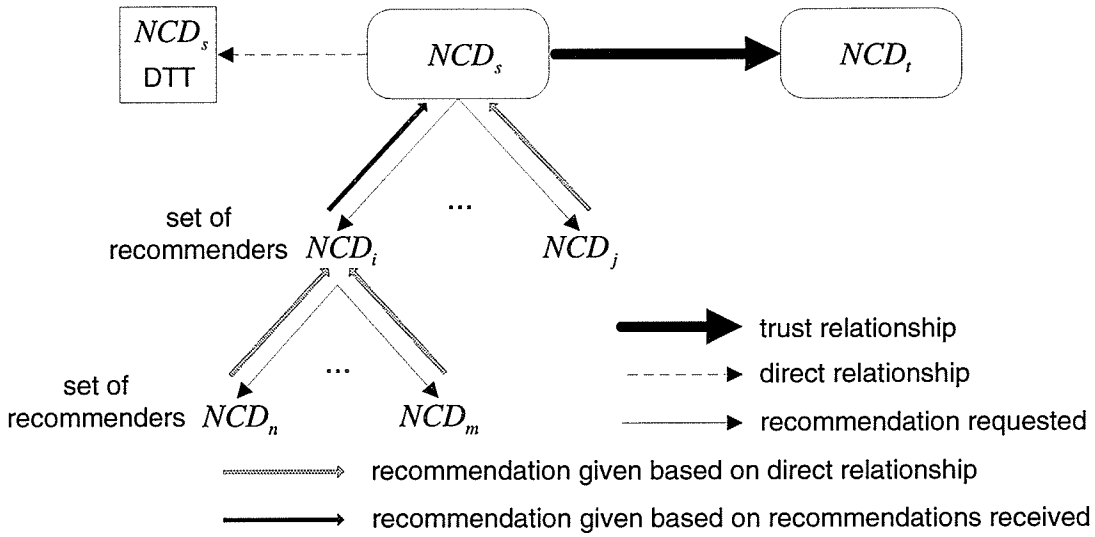


Figure 4.2: An example of a recommendation tree existing in a trust relationship.

4.3.2 Coherent versus Incoherent Trust Models

Suppose we merge all the local DTTs together to obtain a *global* DTT, that will depict how any given NCD trusts other NCDs as shown in Table 4.3. The global DTT will have equal number of rows and columns entries if we disregard the multiple entries made due to the different contexts. The variation across a single column of the global DTT shows how different NCDs trust a given NCD.

Since NCDs give recommendations using their DTTs, the structure of DTT used by the trust model is essential for recommendations to be useful (i.e., the structure of DTT is essential for the trust model to learn from recommendations). The effectiveness of the accuracy measure is a good indicator of whether the recommendations are useful to the

learning process of the trust model. This is because before an NCD can use the recommendation received from a recommender, the NCD must adjust the recommendation to reflect the recommender's accuracy (i.e. this process determines if the recommendation is useful or not). Since the accuracy measure works by comparing the ITL to the recommendation and then shifting up or down (i.e. adjusting) the recommendation accordingly to narrow the gap between ITL and the recommendation, the accuracy measure will be effective as long as the recommendation is consistently low or consistently high in relation to the ITL. Hence in this case, the trust model will learn from recommendations.

That is, a trust model is considered to be coherent if the recommendation is consistently low or consistently high in relation to the ITL. Otherwise, the trust model is considered incoherent.

In the thesis and for the rest of the simulation, we adopted the following definition of coherent and incoherent trust models. Suppose NCD_j is highly trustworthy in the global sense, then we can expect the global DTT to have very high trust levels along the j-th column. We refer to the trust model as *coherent* if the variation along any given column of the DTT is below a given threshold. Otherwise, the trust model is considered to be *incoherent*. In practice, a coherent DTT means that when NCD_s is considered trustworthy by NCD_t, it is very highly likely that other NCDs will also find NCD_s trustworthy. As we will see in Chapter 5, Section 5.3.2, this property of the DTT is essential for recommendations to be useful. For example, assume that, in the global sense, NCD_j is very trustworthy and that NCD_i is very untrustworthy. Table 4.4 is considered coherent because other NCDs (i.e., NCD_s, NCD_t, NCD_p, NCD_q) agree on the trustworthiness of NCD_i and NCD_j. Whereas, Table 4.5 is considered incoherent because there is no global consensus on the trustworthiness of NCD_i and NCD_j. For simplicity purposes, we ignored the timestamps in these two DTT tables (i.e., Tables 4.4 and 4.5).

Table 4.4: A coherent global direct trust table.

Network Computing Domains (Trustors)	Network Computing Domains Domains (Trustees)	
	NCD_i	NCD_j
NCD_s	1	5
NCD_t	2	3
NCD_p	1	4
NCD_q	1	3

Table 4.5: An incoherent global direct trust table.

Network Computing Domains (Trustors)	Network Computing Domains Domains (Trustees)	
	NCD_i	NCD_j
NCD_s	1	1
NCD_t	5	3
NCD_p	3	1
NCD_q	1	5

4.3.3 Trust Evolution

Suppose that based on the trust evaluations NCD_s decides to go ahead with the transaction, the transaction can be monitored by the TM_{NCD_s} and the TM_{NCD_t} proxies. The TM_{NCD_s} and the TM_{NCD_t} proxies determine the instantaneous trust levels $ITL_{NCD_s}(NCD_t, t, c)$ and $ITL_{NCD_t}(NCD_s, t, c)$, respectively. Because a TM proxy is controlled by the associated NCD, TM proxies of NCD_s and NCD_t might evaluate the same transaction differently. For how a transaction is monitored and examples of conditions that can cause a breach in the transaction between NCD_s and NCD_t, please refer to Section 3.4.

In the remainder of this section, we detail how the updates mentioned in Figure 3.6

are carried out by NCD_s. If ITLs are obtained by the TM proxies, then they are used to evaluate the two sources of information regarding trust between NCD_s and NCD_t. For example, if ITL_{NCD_s} is obtained, the following two sources of information are updated by TA_{NCD_s}: (a) direct trust between NCD_s and NCD_t (i.e., $DTT_{NCD_s}(NCD_t, t, c)$) and (b) the accuracy of NCD_z in making recommendations for context c at time t (i.e., $A_{RTT_{NCD_s}}(NCD_z, t, c)$

The two sources are evaluated differently. Let $ITL_{NCD_s}(NCD_t, t, c)$ denote NCD_t's ITL for context c at time t as observed by NCD_s and $DTT_{NCD_s}(NCD_t, t, c)$ be the trust level of the DTT_{NCD_s} entry that corresponds to the level NCD_s trusts NCD_t for context c at time t based on direct interaction that NCD_s had with NCD_t. Let δ be a real number between 0 and 1.

$$\begin{aligned} DTT_{NCD_s}(NCD_t, t, c) &= \delta\ DTT_{NCD_s}(NCD_t, t, c) \\ &+ (1-\delta)\ ITL_{NCD_s}(NCD_t, t, c) \end{aligned} \tag{4.1}$$

If $\delta > 0.5$, preference is given to the ITLs determined through the analysis of the previous transactions between NCD_s and NCD_t.

To evaluate the set of recommenders, NCD_s needs to compute the consistency as well as the accuracy measures as shown in Section 3.4. Suppose $\Psi_{RE_{NCD_s}}(NCD_z, NCD_t, t, c)$ is the recommendation error for recommender NCD_z based on the recommendation regarding NCD_t that NCD_z gave to NCD_s for the current transaction and $\Psi_{RE_{NCD_s}}(NCD_z, t, c)$ is the accuracy of recommender NCD_z maintained based on all previous recommendations made by NCD_z. The following formula is used to update NCD_z's recommendation error.

$$\begin{aligned} \Psi_{RE_{NCD_s}}(NCD_z, t, c) &= \delta\ \Psi_{RE_{NCD_s}}(NCD_z, t, c) \\ &+ (1-\delta)\ \Psi_{RE_{NCD_s}}(NCD_z, NCD_t, t, c) \end{aligned} \tag{4.2}$$

Suppose $A_{NCD_s}(NCD_z, NCD_t, t, c)$ is the accuracy of recommender NCD_z based on the recommendation regarding NCD_t that NCD_z gave to NCD_s for the current transaction and $A_{RTT_{NCD_s}}(NCD_z, t, c)$ is the accuracy of recommender NCD_z maintained at RTT_{NCD_s} based on all previous recommendations provided by the average accuracy measure. The following formula is used to update the average accuracy measure.

$$\begin{aligned} A_{RTT_{NCD_s}}(NCD_z, t, c) &= \delta\, A_{RTT_{NCD_s}}(NCD_z, t, c) \\ &+ (1-\delta)\, A_{NCD_s}(NCD_z, NCD_t, t, c) \end{aligned} \tag{4.3}$$

The above equations, use a weighted averaging scheme to determine the update for the parameters. In Section 5.5, we show alternative schemes for updating the parameters and examine their properties. The consistency for the recommenders is updated differently. Suppose $C_{NCD_s}(NCD_z, NCD_t, t, c)$ is the consistency of recommender NCD_z based on the recommendation NCD_z gave for the current transaction to NCD_s regarding NCD_t for context c at time t.

Let $C_{RTT_{NCD_s}}(NCD_z, t, c)$ denote the consistency of recommender NCD_z maintained at RTT_{NCD_s} based on all previous recommendations provided by NCD_z. The following simple formula updates $C_{RTT_{NCD_s}}(NCD_z, t, c)$.

$$\begin{aligned} &C_{RTT_{NCD_s}}(NCD_z, t, c) = \\ &\min(C_{RTT_{NCD_s}}(NCD_z, t, c), C_{NCD_s}(NCD_z, NCD_t, t, c)) \end{aligned} \tag{4.4}$$

This formula penalizes NCD_z for lying even once. When the consistency value reaches 0, the NCD_z is removed from the recommendation set for a random amount of time that is sampled from a predefined interval.

4.4 Behavior Trust Illustration

The following Figure 4.3 illustrates how the behavior trust operates. The figure shows a source NCD (i.e. NCD_s) interested in determining the trustworthiness of a target NCD (i.e., NCD_t) for context c at time t. The NCD_s determines NCD_t's trustworthiness by combining its own experience with NCD_t and the reputation of NCD_t. The NCD_s checks DTT_{NCD_s} to obtain its direct trust in NCD_t as shown in the figure. Determining the reputation of NCD_t is more involved. The reputation of NCD_t is obtained by NCD_s requesting its recommenders in R_{NCD_s} for recommendations regarding NCD_t for context c at time t. If NCD_s decides to check the honesty of its recommenders, then NCD_s will do that by the help of its trusted allies in T_{NCD_s} as explained in Section 3.4. If the honesty check was done by NCD_s through its trusted allies in T_{NCD_s}, then NCD_s can evaluate its recommenders' honesty. This is done by applying the consistency test as shown in Equation 3.4. After the recommendations regarding NCD_t are received by NCD_s, NCD_s shifts every recommendation by its respective recommender's accuracy as well as decaying the recommendation if needed. For example, if the recommendation received from NCD_z is based on recent direct interaction between NCD_z and NCD_t, then there no need of decaying the recommendation. Now, NCD_s is ready to compute the reputation of NCD_t by summing the recommendations and taking the average as defined in Equation 3.7.

After that and as shown in Figure 4.3, NCD_s computes the behavior trust in NCD_t by combining its direct trust level in NCD_t and reputation trust level of NCD_t. Based on the computed NCD_t's trustworthiness, NCD_s can decide to go ahead or reject the transaction with NCD_t. If NCD_s decides to proceed with the transaction and if the monitoring process takes place, the transaction monitor proxy (TM_{NCD_s} proxy) obtains $ITL_{NCD_s}(NCD_t, c, t)$ as explained in Section 3.4. After the transaction is finished, the following can be updated: (a) if $ITL_{NCD_s}(NCD_t, c, t)$ was obtained, NCD_s can evaluate its recommenders' accuracy and (b) NCD_s can update its direct trust level of NCD_t using

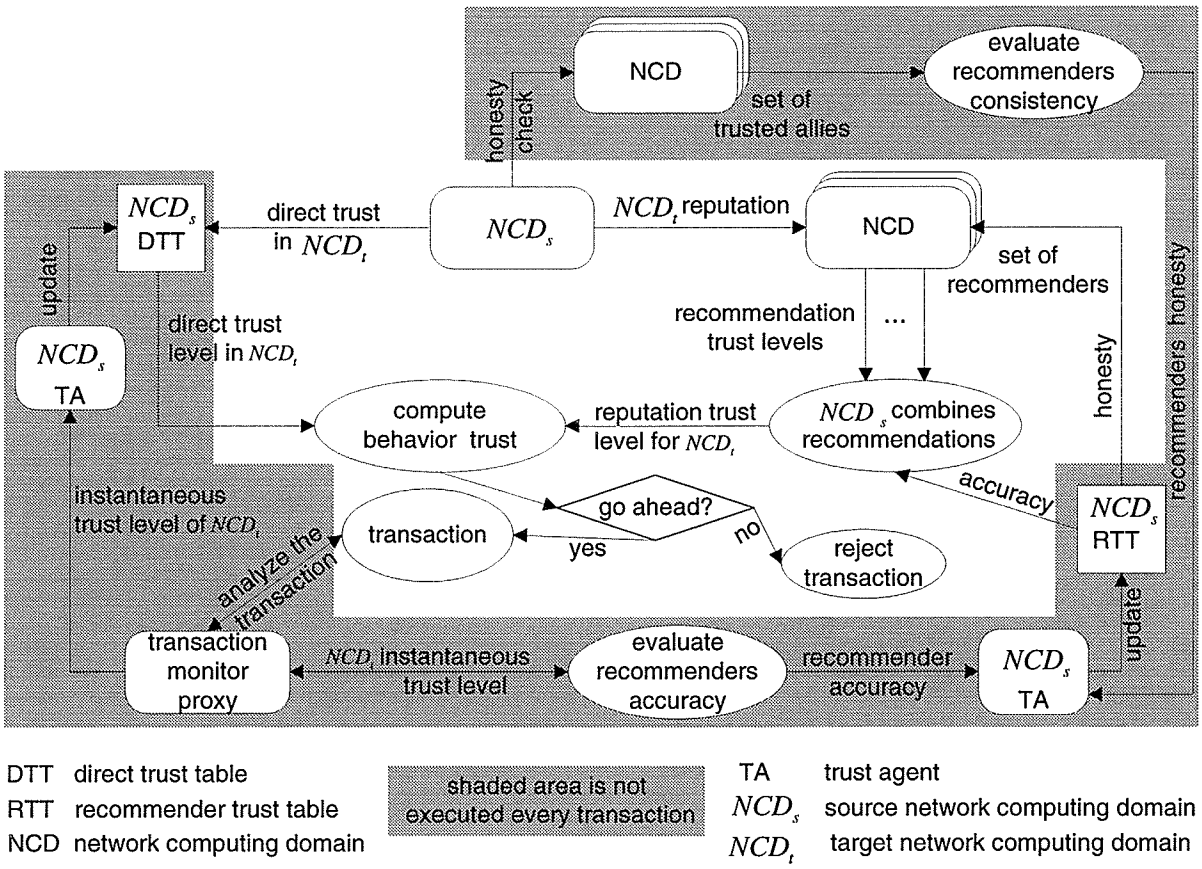


Figure 4.3: Trust Development Cycle.

$ITL_{NCD_s}(NCD_t, c, t)$. In addition, if the consistency check was performed, then NCD_s can update its recommenders' consistency. In Figure 4.3, the shaded area illustrates these update operations. Since monitoring each transaction and checking the recommenders' honesty for each transaction are onerous tasks, the operations shown in the shaded area will be carried out periodically by NCD_s.

4.5 Trust Transaction Example

We present an example to illustrate: (a) the updating process of the recommenders' consistency, (b) the updating process of the recommenders' accuracy, (c) the updating process of the recommenders' recommendation error, and (d) the updating process of NCD_x's direct trust in NCD_t. In other words, we will show how DTT_{NCD_s} and RTT_{NCD_s} are updated. It should be mentioned here that this example is a continuation of the example presented in Section 3.6.2. For the sake of continuation, we introduce the following presentation adjustments: (a) we use NCD_s instead of x, (b) we use NCD_t instead of y, and (c) we use NCD_{r_1}, NCD_{r_2}, and NCD_{r_3} instead of r_1, r_2, and r_3, respectively.

In the example presented in Section 3.6.2, NCD_s performed the consistency check, went a head with the transaction with NCD_t, monitored its transaction with NCD_t, and performed the accuracy check. Table 4.6 summarizes the outcome of these checks. In

Table 4.6: Outcome of tests performed in Example 3.6.2 by NCD_s.

Recommender	Outcome of consistency check performed by NCD_s	Outcome of accuracy check performed by NCD_t
NCD_{r_1}	$C_{NCD_s}(NCD_{r_2}, NCD_t, t, c) = 1$	$\Psi_{RE_{NCD_s}}(NCD_{r_1}, NCD_t, t, c) = 0$
		$A_{NCD_s}(NCD_{r_1}, NCD_t, t, c) = 1$
NCD_{r_2}	$C_{NCD_s}(NCD_{r_2}, NCD_t, t, c) = 1$	$\Psi_{RE_{NCD_s}}(NCD_{r_2}, NCD_t, t, c) = 2$
		$A_{NCD_s}(NCD_{r_2}, NCD_t, t, c) = 0.5$
NCD_{r_3}	$C_{NCD_s}(NCD_{r_3}, NCD_t, t, c) = 0$	$\Psi_{RE_{NCD_s}}(NCD_{r_3}, NCD_t, t, c)$ (Not computed)
		$A_{NCD_s}(NCD_{r_3}, NCD_t, t, c)$ (Not computed)

Table 4.6, recommender NCD_{r_3}'s accuracy and recommendation error were not computed

since NCD_{r_3}'s consistency for the current transaction is zero. That is, NCD_s will consider NCD_{r_3} inconsistent and hence dishonest and so it will be removed from R_{NCD_s}.

Now, NCD_s will update its direct trust in NCD_t. Also, NCD_s will update the consistency, the recommendation error, and the accuracy of its recommenders. These update mechanisms are based on Equations 4.1 to 4.4. The updates are basically two kinds, namely updating DTT_x and updating RTT_x. For the following updates, we choose the value of δ to be 0.9. We start with updating DTT_x and following Equation 4.1. From Example 3.6.2 we have $DTT_{NCD_s}(NCD_t, t, c) = 5$ and $ITL_{NCD_s}(NCD_t, t, c) = 3$ is obtained by monitoring the transaction. Therefore, the new $DTT_{NCD_s}(NCD_t, t, c)$ is calculated as:

$$DTT_{NCD_s}(NCD_t, t, c) = 0.9 \times 5 + 0.1 \times 3 = 4.8 \tag{4.5}$$

The RTT_x updates are shown in Table 4.7. The next time NCD_s engages in a transaction with a target NCD, NCD_s will use the updated values shown in Table 4.7. These updated values will be used by NCD_s as the initial values, illustrated in Figure 3.1, in its next interaction with a target NCD.

Table 4.7: Updates process performed by NCD_s.

Recommender	recommender's consistency	recommendation error	recommender's accuracy
NCD_{r_1}	$C_{NCD_s}(NCD_{r_1}, NCD_t, t, c)$ $= \min(1, 1)$ $= 1$	$\Psi_{RE_{NCD_s}}(NCD_{r_1}, t, c)$ $= 0.9 \times 0 + 0.1 \times 0$ $= 0$	$A_{NCD_s}(NCD_{r_1}, t, c)$ $= 0.9 \times 1 + 0.1 \times 1$ $= 1$
NCD_{r_2}	$C_{NCD_s}(NCD_{r_2}, NCD_t, t, c)$ $= \min(1, 1)$ $= 1$	$\Psi_{RE_{NCD_s}}(NCD_{r_2}, t, c)$ $= 0.9 \times 0 + 0.1 \times 2$ $= 0.2$	$A_{NCD_s}(NCD_{r_2}, t, c)$ $= 0.9 \times 1 + 0.1 \times 0.5$ $= 0.95$

4.6 Trust Model realism and limitations

We showed how the trust model is applied to a large-scale network computing system. For scalability purposes, a network computing system is aggregated into network computing domains (NCDs) and the trust model is required to maintain an entity for each NCD. These NCDs are considered interconnected domains that interact in a P2P fashion to share resources and services amongst themselves. The trust model is deployed in each NCD as shown in Figure 4.1. Implementing the trust model on NCDs that peer with each other is feasible due to the following: Since the goal is to map the trust model onto a large-scale network computing system, the trust model was designed with the scalability factor in mind. The trust model elements such as DTT_{NCD}, RTT_{NCD}, TM_{NCD} proxy, TA_{NCD}, and T_{NCD} are designed to operate and evolve trust in a purely distributed manner. There is no NCD that is omniscient. Rather each NCD: (a) has its own view of how trustworthy other NCDs are and keeps this information in DTT_{NCD}, (b) controls the monitoring process of its own transactions using TM_{NCD} proxy, (c) has its own set of recommenders and set of trusted allies, and (d) maintains DTT_{NCD} and RTT_{NCD} using its own TA_{NCD}. Each NCD cooperates with other NCDs by sharing information in the form of recommendations. Further, each NCD is required to maintain two data structures, namely DTT_{NCD} and RTT_{NCD}. The size of RTT_{NCD} is small since it contains information about R_{NCD}, which is a small set of the total number of NCDs. On the other hand, DTT_{NCD} contains entities for each of the NCDs that this particular NCD directly interacted with. Hence, as the direct interaction with different NCDs increase, the size of DTT_{NCD} increases in a linear fashion.

A potential implementation problem with our trust model is ensuring that the identities of NCDs cannot be created trivially. This is crucial since reputation can get erased if an NCD changes its identity. Hence, an untrustworthy NCD can use this trick to start fresh every time it builds up a bad reputation history.

Further, P2P systems are well known to suffer from *free loader* problem [88, 89], where an individual may refuse to devote resources to external requests and still get full benefit from the P2P system. Taking this further to our trust model, an NCD may expect its recommendation requests to be fulfilled by others. But, when it comes to others asking for recommendations, this particular NCD chooses not to give recommendations by simply returning -1 (i.e., do not know) or basically ignoring the request. That is, NCDs may refuse to give recommendations for various reasons. Further, by isolating dishonest recommenders and routing the recommendation requests to only honest recommenders, a tedious task can be potentially created by bombarding these honest recommenders with a rather huge volume of recommendation requests. As a result, an honest recommender might be inclined to refuse giving recommendations in order to avoid the extra work.

To remedy this problem, our trust model should provide some incentives to encourage and reward these cooperative NCDs. As it is, our model implicitly provides two levels of incentives for NCDs to be cooperative: (a) by modeling honesty, the trust model provides a mechanism for giving an incentive to recommenders to truthfully give recommendations and hence be cooperative in the P2P environment. If a recommender is dishonest, it will be isolated and the rest of the P2P environment will not ask recommendations from it and (b) by modeling trust, the model provides another level of incentive to NCDs to be trustworthy and behave as expected. By enabling trust-aware resource management systems, best behavior is motivated and that in turn improves the overall system performance (please see Chapter 7. Although the above mentioned incentives are well taken, we argue that more explicit incentives should be provided by our trust model. For example, cooperative NCDs should be given reduced cost when using resources or should have a higher priority when submitting tasks.

Another limitation of our trust model that we foresee is the potential creation of performance bottlenecks. Because every NCD wants to keep honest recommenders, the majority, if not all, of the recommendation requests will be routed to the honest recommenders. This

not only creates a tedious task that can be bothersome for these honest recommenders, but also can create a potential performance bottleneck. As a consequence, responses to recommendation requests can experience a longer delay.

4.7 Summary

In this chapter, we presented an architecture of how our trust model can be mapped onto network computing systems. This architecture was presented in various conferences including [36, 61]. Scalability, that is associated with large-scale network computing systems, becomes a vital concern. To address this concern, we considered an architecture where the overall network computing system is partitioned into autonomous entities called network computing domains (NCDs). Considering each node (resource or client) of the networked system as an NCD of the trust model is undesirable from the scaling point-of-view because a networked system can have large number of nodes. Therefore, an NCD is considered as a collection of nodes and the various NCDs can belong to different administrative domains and be managed by different policies. The overall system is organized as a collection of NCDs. One purpose of trust modeling is to facilitate the NCDs to make informed decision when a client wants to access resources that belong to an NCD other than its own NCD. Therefore, the NCDs should be organized such that they are at the same level (i.e., they are all peers) and have the freedom to choose a set of remote NCDs to engage in remote transactions. Naturally, a peer-to-peer organization at the NCD-level satisfies this requirement. Issues such as peer discovery and negotiation that are important for NCDs to operate in a peer-to-peer configuration are beyond the scope of this thesis. Each NCD elects one or more nodes as leaders. A leader of a NCD is assumed to have the complete trust of the nodes within the NCD. However, the leader may not trust all the nodes within the NCD at the same level.

Each NCD has a TM proxy that monitors the NCD-level transactions with other NCDs.

An NCD computes its trust level of other NCDs based on its own experience as well as reputation provided by its set of recommenders. Further, each NCD has a TA_{NCD} that maintains the NCD's DTT and RTT. In their DTTs, NCDs keep track of the trust level of other NCDs that they directly interacted with. While in their RTTs, NCDs keep track of the honesty and accuracy of their recommenders. Essential for recommendations to be useful, the trust model should be coherent. A trust model is considered to be coherent if the variation along any given column of its DTT is below a certain threshold. Otherwise, the trust model is considered to be incoherent. We provided an example in Section 4.3.2 to clarify this concept.

In this chapter, we also discussed how trust evolves by showing how NCD_s updates its direct trust table, its recommenders' consistency, recommendation error of its recommenders, and its recommenders' accuracy. We presented a walkthrough example to show how behavior trust operates as well as a numerical example to illustrate the trust evolution. Finally, we included a section discussing the limitations and potential implementation problems with our trust model.

Chapter 5

Performance Evaluation

5.1 Overview

A series of simulation studies was conducted to examine various properties that affect the performance of the proposed trust model. In these simulations, the trust model was abstracted to keep the complexity manageable and at the same time provide sufficient detail as explained below. In Section 5.2, we examined the performance of our trust model. We accomplished that by investigating two properties. First, since modeling behavior trust is a learning process, we carried out an experiment to examine under what conditions can our trust model learn the trust levels. Second, we investigated our trust model's ability to correctly predict the trust that exists between the NCDs.

In Section 5.3.4, we point out other parameters that might have been examined in the performance evaluation study. We give justifications for not including them in the simulation. As illustrated through simulation experiments in Sections 5.4 and 5.5, we also point out the importance of some of these parameters in estimating the trust levels.

5.2 Simulating Trust Model Performance

5.2.1 Goals of the Simulation

There are two goals of the simulation experiments performed in section 5.2. First and as explained in Section 4.3.2, the trust model can be referred to as *coherent* or *incoherent.* We also claimed that this property of the trust model is essential for trust estimation to be useful. In this section, through simulation experiments, we show that our claim holds.

Second, P2P reputation-based systems rely on cross-ratings. Because these systems are based on a community which may include untrustworthy nodes, correctly predicting the trust level between nodes is difficult to do [28, 5]. In trust models such as [5, 57, 37], the main goal is to identify trustworthy and untrustworthy nodes. Hence, a node will be able to engage in transactions with trustworthy nodes and avoid untrustworthy ones. In our trust model and through simulations, we examined this property by measuring our trust model's ability to correctly predict the trust that exists between NCDs. We accomplished this by varying the number of dishonest NCDs, the frequency of the monitoring process, and the weight of direct trust versus reputation. We also varied the number of transactions that happen between NCDs to examine the agility of the trust model in detecting the untrustworthy NCDs.

5.2.2 Overview

In Chapters 3 and 4, we explained in detail how our physical (i.e. real world) behavior trust model operates. In this section, we give an overview of the simulation (i.e. conceptual) behavior trust model including any necessary simplifications with justifications of why these simplifications do not limit the validity and the applicability of our physical behavior trust model.

In the simulation model, the physical system that consists of a collection of NCDs that

peer with each other is represented by a collection of peering simulated NCDs. Therefore, in the simulation, there is no representation of the nodes within NCDs and this does not impact the relevance in presenting the real world model for the following reasons. First, the transactions that originate and terminate at the resources within NCDs are modeled as happening between the NCDs. This aggregation does not introduce any errors because in the physical behavior trust model, the NCDs are assumed to represent the client and resource sides. Second, clients' and resources' trustworthiness is represented globally by their NCD. Hence, as far as the trust model is concerned, the objective is to: (a) model behavior trust by evolving and updating trust levels that exist between these peering simulated NCDs and (b) predict the trust levels that exist between these peering simulated NCDs to be as close as possible to the real world trust levels that exist between the real world peering NCDs. Having said that, an NCD has to manage its members (i.e., clients and resources) to maintain a high global reputation and this incurs additional costs and raises issues such as how resources join, are managed, and leave an NCD. One of the costs and issues that needs to be addressed when managing clients and resources within an NCD is scalability. Scalability is discussed in detail in Chapter 6.

Through simulation, we would like to capture the following. Assume that we know the real world picture as far as behavior trust is concerned. That is, we assume that we have a data structure that contains the real trustworthiness that exist between the real world peering NCDs. Since one of the underlying assumptions of this study is that naturally there exists a network of trust relationships among different NCDs, we model the natural trust relationships that exist among the NCDs by an *Actual Direct Trust Table* (ADTT). This table contains the absolute trustworthiness that exists among different NCDs. For simplicity, we assume that these trust relationships are constants for the duration of the simulation time. The ADTT has the same structure as Table 4.1.

The objective of trust modeling is to discover these trust relationships via observations of ongoing transactions. We do this by running two different simulation experiments: (a)

under what conditions can our trust model learn the true trust levels in ADTT. We examine two types of conditions, namely a coherent versus an incoherent trust model and (b) examine how successfully can we predict the true trust levels in ADTT.

In an actual system, the TM proxies periodically examine the transactions among the NCDs to determine the trust level that exists between source and target NCDs. As explained in Section 3.4, this is referred to as the instantaneous trust level or ITL. Through outside observation this is the closest we can get to the actual trust expressed in ADTT. The ITLs obtained by a TM proxy are used to update a *computed direct trust table* (CDTT). The CDTTs keep track of trust levels revealed by the post mortem analysis carried out as the transactions take place among the different NCDs. The revealing of the ITLs 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 ITLs by equating the CDTT entries to the corresponding ADTT entries plus a small value that is chosen uniformly at random in the range [0,2]). The CDTT is initially created by adding a random "noise" generated from [0,4] to the ADTT.

In addition to the CDTTs, we maintain *predicted direct trust tables* (PDTTs) to track the evolution of the trust relationship among the NCDs. The PDTTs are updated using the trust values that are predicted by Equation 3.5. It should be noted that the prediction process uses the current entries of the CDTT to determine the new values for PDTT. The CDTT and PDTT have the same structure as Table 4.1. The PDTT entries are initialized to -1 meaning that all NCDs are unknown to each other.

Furthermore, RTT has the same structure as Table 4.2 and it keeps track of the consistency and accuracy of NCD_s's recommenders. Initially, NCD_s selects its recommenders uniformly at random and considers them to be consistent (i.e., the recommenders' consistency is set to 1) and have maximum accuracy (i.e., the recommenders' accuracy is set to 1). Assume that NCD_s has 2 recommenders, namely NCD_j and NCD_k. Initially, NCD_s considers its recommenders to be consistent and have maximum accuracy, as illustrated in

Table 5.1: Initial recommender trust table maintained by NCD_s.

	Consistency	Accuracy
NCD_j	1	1
NCD_k	1	1

Table 5.1. For the trusted allies table (T), NCD_s chooses the trusted allies (i.e., members of T) based on off-line relationships. The structure of T is just a list of these NCDs chosen by NCD_s based on off-line relationships. The trusted allies are assumed to be consistent and have maximum accuracy. TM proxy and TA are agents whose duties are monitoring transactions and maintaining trust tables (i.e., CDTT, PDTT, RTT, and T), respectively.

Figure 5.1 shows the simulation model and the simulation entities used in each NCD. Each NCD is simulated as having its own CDTT, PDTT, RTT, T, TM proxy, and TA. Each TA maintains its own CDTT, PDTT, RTT, and T. TAs also communicate with their respective TM proxy, instructing it to monitor a transaction if needed. Further, TAs send transaction requests, reply to the transaction requests, and initiate the transaction.

5.2.3 Design and Exogenous Parameters

Tables 5.2 and 5.3 show the design and exogenous parameters used in the simulation. The NCDs' transactions process was simulated using a discrete event simulator. The term randomly generated over a range [a, b] means that the number is generated using a discrete (integer-valued) uniform distribution over a, a+1, ..., b inclusive. That is written as U[a, b]. The transactions that take place among the NCDs arrive at the NCDs based on a Poisson process. The design parameter $reps$ denotes how many times the simulation run is repeated. In simulating our trust model, $reps$ is set deterministically at 10. That is, each point in Tables 5.16 to 5.18 and Figures 5.4 to 5.16 is the result of 10 simulation runs.

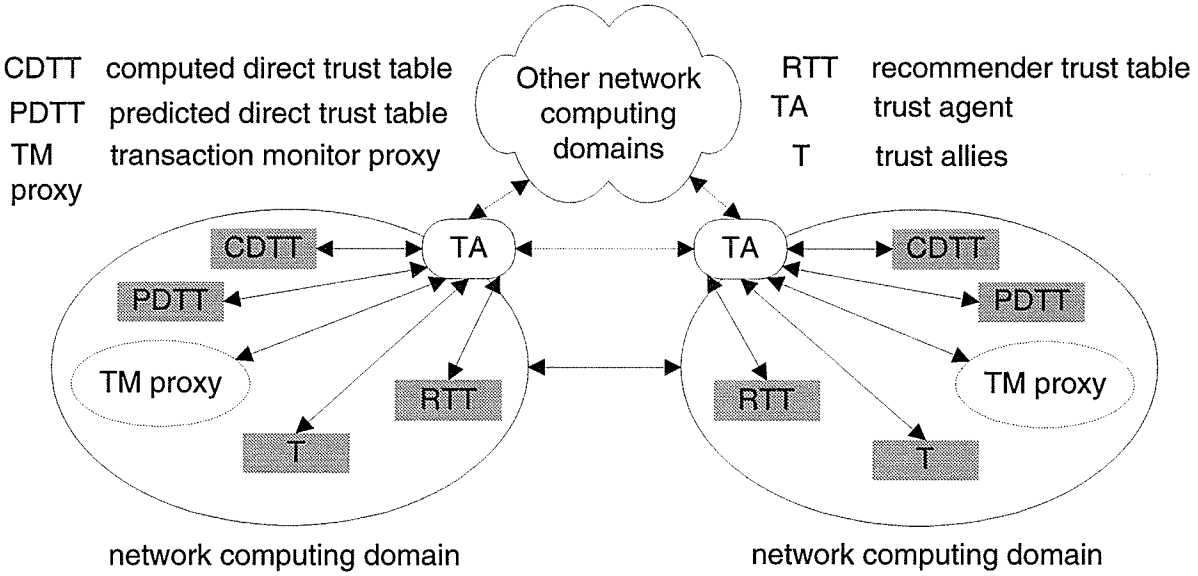


Figure 5.1: Simulation model.

The trust model topology used in the simulation consists of 30 NCDs (i.e., $NCDs\_num = 30$). The number of recommenders ($recs\_num$) for each NCD was fixed and set to 4, and the number of trusted allies ($allies\_num$) for each NCD was fixed at 4. It should be noted that recommenders are dropped if their consistency becomes 0. That is, an NCD drops out inconsistent, and hence dishonest, recommenders from its recommendation set and replaces them with new recommenders. But the recommender's set size stays fixed at 4. In Section 5.4, we will discuss varying the recommendation set to examine its effect on the performance of the trust model. For $allies\_num$, it turns out that as long as $allies\_num > 1$, it is sufficient for the consistency check. That is, as long as the size of the trusted allies set is > 1, the consistency check can be carried out correctly.

The source and the target NCDs for each transaction were randomly generated over a range [0, $NCDs\_num - 1$]. The frequency of the monitoring process (mon_freq) is set

Table 5.2: Design parameters used in the simulation.

Symbol	Definition	Design parameter values
$reps$	How many times the simulation is repeated for each point in the graphs and tables	$reps = 10$
$NCDs\_num$	Number of NCDs	$NCDs\_num = 30$
$recs\_num$	Number of recommenders	$recs\_num = 4$
$allies\_num$	Number of trusted allies	$allies\_num = 4$
$NCDs\_dishonest$	Number of dishonest NCDs	$NCDs\_dishonest = [0, 15, 20]$
$transactions\_num$	Number of transactions per relation	$transactions\_num = [5, 10, 25, 50, 100, 150]$
α	Direct trust weight	$\alpha = [0.0, 0.5, 1.0]$
$cons\_check$	Consistency check	Boolean value taking 0 or 1
$accu\_check$	Accuracy check	Boolean value taking 0 or 1
mon_freq	Transaction monitoring frequency	$mon\_freq = [1, 5, 10, 20]$
$cons\_freq$	Consistency frequency	$cons\_freq = [1, 5, 10, 20]$
$accu\_freq$	Accuracy frequency	$accu\_freq = [1, 5, 10, 20]$
ϵ_{RE}	Consistency determination	$\epsilon_{RE} = 0$

deterministically at [1, 5, 10, 20] meaning that the TM proxy is monitoring every, every fifth, every tenth, or every twentieth transaction, respectively. The objective of varying the frequency of the monitoring interval is to examine the dependence of the trust model on the instantaneous trust levels obtained by the TM proxies.

The frequency of carrying out the the consistency check ($cons\_freq$) and the accuracy check ($accu\_freq$) is set deterministically at [1, 5, 10, 20]. The consistency determination parameter ϵ_{RE} is set to 0. The reason behind setting ϵ_{RE} to 0 is as follows. As explained in Section 3.4, the value of $\Delta_{RE_{NCD_s}}(NCD_z, NCD_t, t, c)$ will be computed. If

Table 5.3: Exogenous parameters used in the simulation.

Symbol	Definition	Design parameter values
$NCDs\_untrust$	Number of untrustworthy NCDs	$NCDs\_untrust = U[NCDs\_num - 25, NCDs\_num - 5]$
$rels\_exist$	existing trust relationships	$rels\_exist = U[NCDs\_num - 25, NCD\_num - 20]$

$\Delta_{RE_{NCD_s}}(NCD_z, NCD_t, t, c) \neq 0$, it means that there is a difference in the trust levels that NCD_z gave out to NCD_s's trusted allies. Now, there is no reason why NCD_z gives out inconsistent trust levels except if NCD_z has some malicious motives. Hence NCD_z should be considered inconsistent and dishonest.

As explained in Figure 3.6, the consistency check is carried out every m transaction, while the accuracy check is carried out every n transaction. We chose to carry out the accuracy and consistency checks simultaneously (i.e., $m = n$) because of the following reason. The purpose of the consistency check is to filter out the inconsistent and hence dishonest recommenders. But after carrying out the consistency check, some of these recommenders may be dishonest. If these dishonest recommenders are given the chance, they will pollute the recommendation network. Carrying out the accuracy check right away enables our trust model to capture these consistent but dishonest recommenders and adjust their recommendations before using them to compute the reputation of the target NCD. We also, relaxed the consistency frequency, monitoring frequency, and accuracy frequency because of the following reasons: (a) if we often carry out these mechanisms (consistency, monitoring, and accuracy), then there will be a significant overhead for trust computation, and (b) it is not realistic to carry the consistency check, monitoring process, and accuracy check every transaction. If an NCD can perform these checks every transaction, then there is no need to model behavior trust!

The value of α determines how the trust model is using the direct and reputation components in computing the final trust levels. For example, $\alpha = 1$ means only the direct trust is used, while $\alpha = 0$ means only the reputation is used. By varying α, we can examine the dependence of the model on the different trust components. The value α is set deterministically at [0.0, 0.5, 1.0]. We also varied the number of transactions per trust relationship ($transactions\_num$) to evaluate the speed of the convergence of the trust model to the actual trust among the different NCDs. The value of $transactions\_num$ is set deterministically at [5, 10, 25, 50, 100, 150]. In section 5.2.5, we illustrate how the $transactions\_num$ parameter is implemented as a mechanism in our trust model. The number of dishonest NCDs ($NCDs\_dishonest$) is set deterministically at [0, 15, 20]. By varying $NCDs\_dishonest$, we can examine the effect of the number of dishonest NCDs on the trust model's ability to identify the untrustworthy NCDs.

The number of untrustworthy NCDs ($NCDs\_untrust$) is randomly generated over a range [$NCDs\_num - 25$, $NCDs\_num - 5$]. An NCD is considered untrustworthy if its trust level is ≤ 2. Otherwise, the NCD is considered trustworthy. The number of existing trust relationships ($rels\_exist$) is randomly generated over a range [$NCDs\_num - 25$, $NCDs\_num - 20$]. The number $rels\_exist$ determines the number of trust relationships that pre-exist before the simulation takes place. At the end of Section 5.2.4, an example is given to illustrate how some of these parameters are implemented as mechanisms.

5.2.4 Conceptual Model

Figure 5.2 shows the simulation control flow for a single point (i.e., for one value of $NCDs\_dishonest$, one value of mon_freq, one value of α, and one value of $trans$) in Tables 5.16 to 5.18 and Figures 5.4 to 5.16. First, we prepare an array called *results* to store the *success rates* resulting from running the simulation $reps$ times. The *success rate* measure the convergence rate of PDTT to ADTT. We also initialize the variable *rep_num* to

0 and the k entries of the array *results* to 0 as well. The design parameter $reps$ is included in this phase to be used as the size of the array *results*. The settings phase is explained in Section 5.2.3, where the design and exogenous parameters are set. Then, the simulation starts the initialization phase, which is detailed in Section 5.2.5. After that, the simulation enters a loop. This loop repeats for $trans$ times, where $trans$ is the number of transactions to be simulated as explained in Section 5.2.5. The data analysis phase outputs the *success rate*. The *success rate*, which is a real number, will be stored in the *results* array. The variable *rep_num* is incremented by 1 and if $rep\_num < reps$, then another round of the simulation takes place. Otherwise, the simulation stops.

First, we want to explain how the simulated entities such as ADTT, CDTT, PDTT, RTT, and T fit into the schemes explained in Section 3.6. Suppose we merge all the local DTTs together to obtain a *global* DTT, that will depict how any given NCD trusts other NCDs as shown in Tables 5.4, 5.5, and 5.6. These DTTs will have an equal number of row and column entries if we disregard the multiple entries made due to the different contexts. Also for illustration purposes, we ignore the time stamps assigned to the different trust levels. Table 5.4 shows a coherent ADTT that is assumed known in our simulation. It was created for 6 NCDs and $NCDs\_untrust = 2$, where NCD_0, NCD_1, NCD_2, and NCD_4 are trustworthy. On the other hand, NCD_3 and NCD_5 are untrustworthy. Now, for the simulation to take place, we have to generate a CDTT. Remember that one of the exogenous parameters is $rels\_exist$. In this example, $rels\_exist$ is set to 3 meaning that each NCD has previously interacted with another 3 NCDs. For example, the first row in Table 5.5 denotes that NCD_0 has previous interaction with NCD_1, NCD_4, and NCD_5. For NCD_2 and NCD_3, NCD_0's direct trust level is set to -1 indicating that it had no previous interaction with them. The entries in Table 5.6 (i.e., PDTT) are initialized to -1. In section 3.6, DTT is simulated by CDTT. The PDTTs are updated using the trust values that are predicted by Equation 3.5.

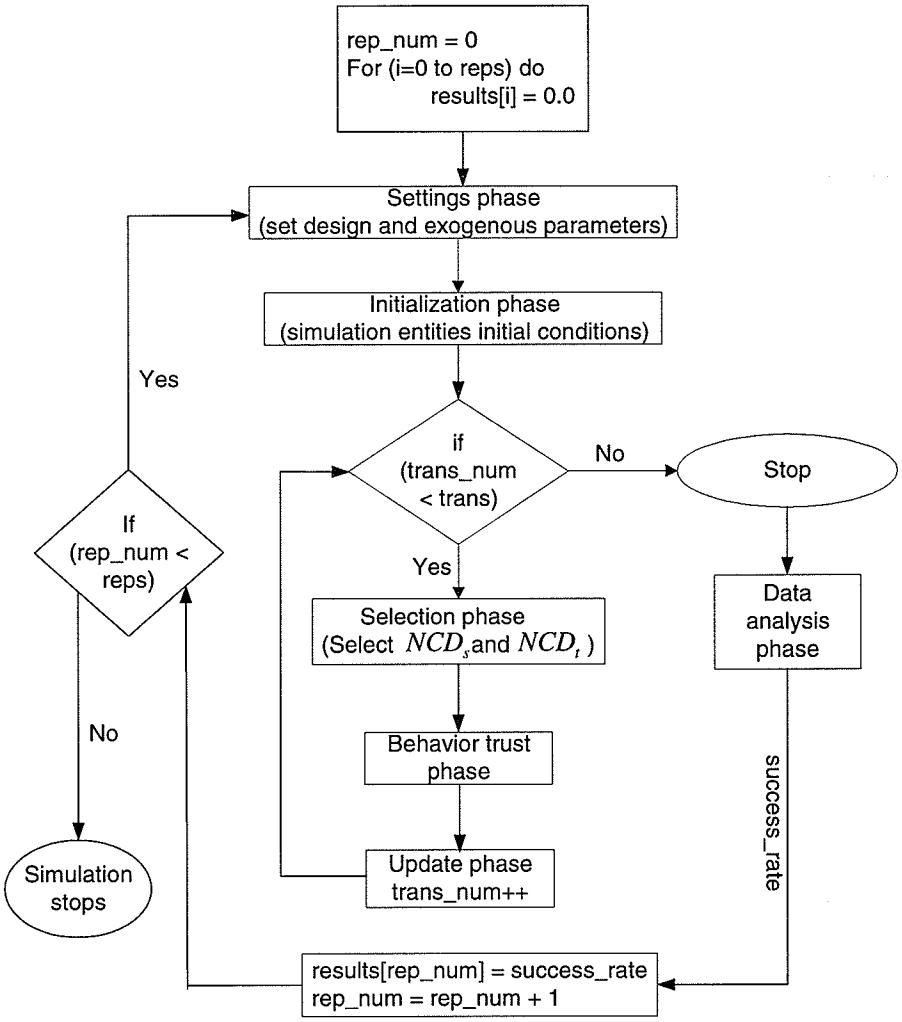


Figure 5.2: Simulation control flow.

5.2.5 Initialization Phase

In the initialization phase, we start by setting up the variable $trans\_num$ to 0. The variable $trans\_num$ is basically a counter that keeps track of the number of transaction initiated during the simulation. We also set up the variable $trans$, which indicates the number of

Table 5.4: An actual direct trust table example. Element in row i column j, TL_{ij} = direct trust by NCD_i in NCD_j.

i/j	0	1	2	3	4	5
0	5.00	3.00	4.00	2.00	4.00	1.00
1	3.00	5.00	5.00	2.00	4.00	2.00
2	3.00	3.00	5.00	2.00	4.00	2.00
3	3.00	5.00	4.00	5.00	3.00	1.00
4	5.00	5.00	4.00	2.00	5.00	1.00
5	4.00	4.00	5.00	1.00	5.00	5.00

Table 5.5: A computed direct trust table example. Element in row i column j, TL_{ij} = direct trust by NCD_i in NCD_j.

i/j	0	1	2	3	4	5
0	5.00	1.00	-1.00	-1.00	3.00	5.00
1	3.00	5.00	3.00	-1.00	-1.00	3.00
2	4.00	5.00	5.00	3.00	-1.00	-1.00
3	-1.00	5.00	-1.00	5.00	2.00	3.00
4	-1.00	-1.00	2.00	3.00	5.00	5.00
5	4.00	-1.00	3.00	2.00	-1.00	5.00

transactions in each simulation run. Each entry (in Tables 5.4, 5.5, and 5.6) denotes the trust level resulting from a relationship between 2 NCDs (except the diagonal entries). For example in Table 5.5, the second entry in the first row indicates that NCD_0 trusts NCD_1 at trust level 1.0. The design parameter $transactions\_num$ denotes the number of times that each entry in Table 5.6 is updated. Remember that ADTT is not updated, CDTT is updated only periodically by the TM proxies when monitoring takes place. But each time a transaction takes place, Equation 3.5 is computed and PDTT (i.e., Table 5.6) is updated. That is, on average, each entry in PDTT is updated $transactions\_num$ times. Now, the

Table 5.6: A predicted direct trust table example. Element in row i column j, TL_{ij} = direct trust by NCD_i in NCD_j.

i/j	0	1	2	3	4	5
0	5.00	-1.00	-1.00	-1.00	-1.00	-1.00
1	-1.00	5.00	-1.00	-1.00	-1.00	-1.00
2	-1.00	-1.00	5.00	-1.00	-1.00	-1.00
3	-1.00	-1.00	-1.00	5.00	-1.00	-1.00
4	-1.00	-1.00	-1.00	-1.00	5.00	-1.00
5	-1.00	-1.00	-1.00	-1.00	-1.00	5.00

diagonal entries of PDTT do not denote any relation and can be ignored. The same can be said regarding the diagonal entries of ADTT and CDTT. Therefore, the number of entries that simulate relations between the NCDs is $NCDs\_num \times (NCDs\_num - 1)$. Since $transactions\_num$ is the number of times that each entry in Table 5.6 is updated, then the number of transactions ($trans$) in each simulation run can be calculated as:

$$trans = transactions\_num \times (NCDs\_num \times (NCDs\_num - 1)) \qquad (5.1)$$

Since $NCDs\_num = 30$ and $transactions\_num$ is set deterministically at $[5, 10, 25, 50, 100, 150]$, then $trans$ is set deterministically at $[4350, 8700, 21750, 43500, 87000, 130500]$. Also, in the initialization phase, each NCD chooses its initial recommenders uniformly at random such that each recommender is considered to be consistent and have maximum accuracy.

5.2.6 Performance Metric

The goal of behavior trust modeling is to predict the trustworthiness of others [10]. The success of a trust model is determined by how correctly it predicts the natural trustworthiness. As explained in Section 5.2.4: (a) we model the natural trust relationships that exist among the NCDs by ADTT and (b) PDTT contains the predicted trust relationships among the simulated NCDs that our trust model maintains through the simulation process. Now, the goal is to examine how close the trust levels maintained in PDTT are to their respective natural trust levels in ADTT.

We define *success rate* (SR) as the ability to correctly predict the trust that exists between the NCDs. A prediction is considered successful when: (a) a trustworthy NCD is predicted as trustworthy, or (b) an untrustworthy NCD is predicted as untrustworthy. An NCD is considered to be trustworthy if its trust level is in [3, 5] and considered to be untrustworthy if its trust level is in [1, 2]. For $i \neq j$, let the value of the prediction function, $\Phi(NCD_i, NCD_j)$, be 1 if NCD_i correctly predicts NCD_j's trustworthiness and 0 otherwise. Hence, SR is computed for n NCDs at time t as follows: (The pseudo-code for calculating the success rate is included in appendix B.)

$$SR(t) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{n} \Phi(NCD_i, NCD_j)}{n \times (n-1)} \times 100 \quad , i \neq j \tag{5.2}$$

5.2.7 Event Generation

In this section, we discuss the type of events that change the state of the system and present a flow chart to show how each event is being generated in our system. There are 4 event types, namely *relation*, *recommendation*, *reply*, and *complete*. A relation event is created to simulate NCD_s's interest to engage in a behavior trust relationship with NCD_t. A recommendation event is created to simulate the desire of NCD_s to request recommendation

from its recommender z regarding the trustworthiness of NCD_t, while a reply event is created to simulate the response of z to the recommendation request sent by NCD_s. Finally, a *complete* event is created by NCD_s to simulate the readiness of NCD_s to compute the trustworthiness of NCD_t. Readiness means that NCD_s got all the responses from its recommenders regarding the trustworthiness of NCD_t and hence NCD_s is ready to compute NCD_t's reputation. Table 5.7 shows the filled structure of an event.

Table 5.7: The structure of an event.

field type	field name	Comments
int	type	An integer field having the following values: 1 if the event type is RELATION 2 if the event type is RECOMMENDATION 3 if the event type is REPLY 4 if the event type is COMPLETE
double	time	The time of the event
int	parent	The parent NCD in a recommender tree.
int	initiator	The NCD initiating the event (i.e., x or NCD_s).
int	receiver	The NCD receiving the event (i.e., z or NCD_r).
int	recommendee	The NCD whose reputation is sought (i.e., y or NCD_t).
double	TL	The trust level that conveys the trustworthiness of the recommendee

We should mention that a numerical example will be given in Section 5.2.8 to illustrate and clarify: (a) how and why events are created and generated and (b) how the event structure shown in Table 5.7 is filled and used with each of the four event types. Now and as shown in Figure 5.3, we will discuss in more detail the event generation flow control. For simplicity reasons, we used the following notation in the figure: x instead of NCD_s, y instead of NCD_t, and z instead of NCD_r. The event generation process starts by creating a relation event and inserting it into the calendar (i.e. event list). This step is taken to

initialize the calendar. Then, the event-generation process enters the main body that loops as long as the calendar is not empty. An event is picked up from the calendar and based on the event type, the system state changes as explained below. After event processing control returns to the main body to select the next event, the standard event-oriented simulation algorithm [90] is followed. We now detail the state changes occurring for each event type.

If the event type is RELATION, the event initiator (i.e., NCD_s) will request recommendations from each $z \in RTT_{NCD_s}$. NCD_s simulates this by creating a recommendation event for each of its recommenders and inserting in into the calendar. Then, while the calendar is not empty, a new event is picked up from the calendar.

If the event type is RECOMMENDATION, then the event receiver z consults DTT_z to find out whether it had any prior transactions with NCD_t. Remember from Section 4.3.1 if $DTT_z(NCD_t, t, c) = -1$, it denotes that z did not have a prior direct relationship or transaction with NCD_t. Here there are two outcomes. First: If $DTT_z(NCD_t, t, c) \neq -1$, then z creates a reply event and inserts in into the calendar. Second: If $DTT_z(NCD_t, t, c) = -1$, then z will request its set of recommenders to determine NCD_t's reputation. Recommender z simulates that by creating a recommendation event for each of its recommenders and inserting in into the calendar. It should be mentioned that before z inserts the recommendation event into the calendar, it does the following: (a) assigns the event parent to be the event initiator and (b) assigns the event initiator to be itself. It does that because z, as a child in the recommendation tree, is requesting recommendations from its own recommenders. So, basically we will have a recommendation tree consisting of the parent, child (i.e., z), and the children of z (i.e., z's recommenders).

If the event type is REPLY, the event initiator checks if it has received replies from all of its recommenders. If some recommenders have not responded to the event initiator's recommendation requests, then a new event is picked from the calendar. Otherwise, the event parent is checked. There are two possibilities. First: If the event parent $= -1$, it means that the event receiver is NCD_s (i.e., the root of the recommendation tree). If this

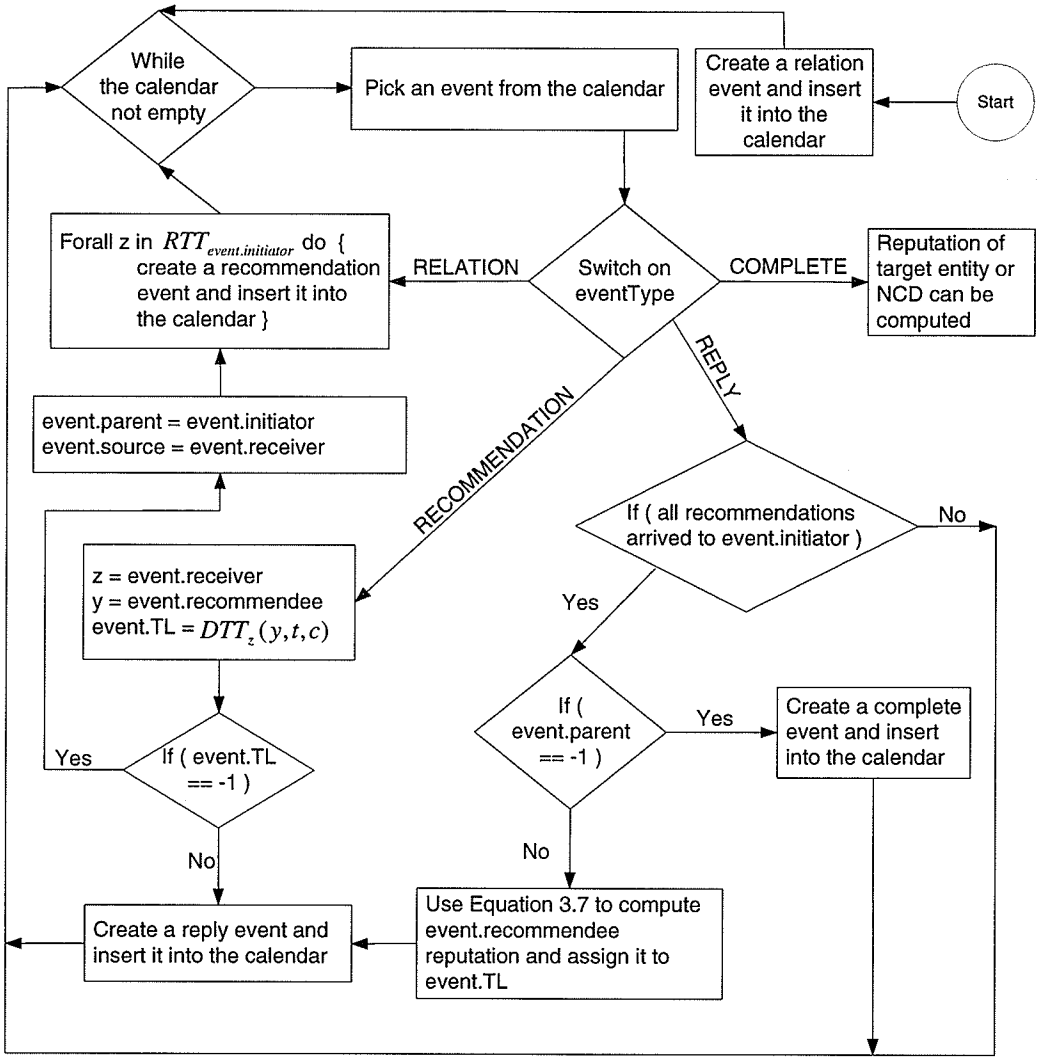


Figure 5.3: Event generation flow control.

is the case, NCD_s creates a complete event and inserts it into the calender. Second: If the event parent $\neq -1$, the event receiver, which must be a recommender, uses Equation 3.7 to compute NCD_t's reputation, which is assigned to the event TL. Then the event receiver creates a reply event to its parent and inserts it into the calender.

If the event type is COMPLETE, the event initiator NCD_s knows that all of its recommenders have responded to its recommendation requests regarding the trustworthiness of NCD_t. Therefore, NCD_s is ready to compute NCD_t's reputation using Equation 3.7.

Figure 5.3 illustrates the event generation for one behavior trust relationship. Figure 5.3 can be generalized, where k number of trust behavior relationships can be generated. In that case, a variable called *relation_num* is included in the event structure shown in Table 5.7. The purpose of the variable *relation_num* is to keep track of events generated and match these events to their corresponding trust behavior relationship. For example, an event might be generated as a result of a trust behavior relationship between NCD_s and NCD_t and another event might be generated as a result of a trust behavior relationship between NCD_p and NCD_q.

5.2.8 Event Generation Example

Let us assume that NCD_s wants to engage in a trust behavior relationship with NCD_t and that NCD_s has 2 recommenders, namely r_1 and r_2. Also assume that r_1 has previous interactions with NCD_t and r_2 has no previous interactions with NCD_t. Let r_3 be the only recommender that r_2 has. Also, let us assume that r_3 has previous interactions with NCD_t. To initialize the calendar, NCD_s creates a relation event as shown in Table 5.8. The event type is 1 (i.e., RELATION) and time of event is generated based on Poisson process with mean 1.0. Since the event was generated by NCD_s (i.e., the root of the recommendation tree), the parent is set to -1 to denote that there is no parent for this event. Since there are no recommenders requested yet, the recommendee is set to -1. The relation event is inserted into the calendar. Following the event flow control, Figure 5.3, an event is picked from the calendar. Since there is only one event in the calendar, the relation event created by NCD_s is picked. At this time, NCD_s creates recommendation events, shown in Table 5.9, for each of its recommenders and insert these events in the calendar. Since

Table 5.8: A relation event created by NCD_s.

type	time	parent	initiator	receiver	recommendee	TL
1	0.075532	-1	NCD_s	NCD_t	-1	-1

Table 5.9: Recommendation events created by NCD_s.

type	time	parent	initiator	receiver	recommendee	TL
2	0.455785	-1	NCD_s	r_1	NCD_t	-1
2	1.103377	-1	NCD_s	r_2	NCD_t	-1

events are sorted in ascending order of their time, the next event to be picked from the calendar is the recommendation event created by NCD_s for recommender r_1. Because r_1 has previous interaction with NCD_t, assume that it finds $DTT_{r_1}(NCD_t, t, c) = 3$, which is assigned to the event TL as illustrated in Table 5.10. Then r_1 responds right away to NCD_s as illustrated by Table 5.10. This response is simulated by creating a reply event and inserting it into the calendar. The next event to be picked from the calendar is the the

Table 5.10: A reply event created by r_1.

type	time	parent	initiator	receiver	recommendee	TL
3	1.187279	-1	r_1	NCD_s	NCD_t	3

recommendation event created by NCD_s for recommender r_2. Since r_2 has no previous interaction with NCD_t, it does the following (according to Figure 5.3): (a) it sets event initiator to event parent and (b) it sets event receiver to event initiator. Since r_2 has only

one recommender (i.e., r_3), it creates one recommendation event and inserts it into the calendar. These steps are illustrates in Table 5.11. The next event to be picked from the

Table 5.11: Recommendation events created by r_2.

type	time	parent	initiator	receiver	recommendee	TL
2	2.031448	NCD_s	r_2	r_3	NCD_t	-1

calendar is the the reply event created by r_1 to NCD_s. At this time NCD_s is still waiting for the reply from r_2, so the next step is to pick an event from the calendar. The next event to be picked from the calendar is the the recommendation event created by r_2 to r_3. Because r_3 has previous interaction with NCD_t, assume that it finds $DTT_{r_3}(NCD_t, t, c) = 2$, which is assigned to event TL as illustrated in Table 5.12. Then r_3 responds right away to r_2 as illustrated by Table 5.12. This response is simulated by creating a reply event and inserting it into the calendar. The next event to be picked from the calendar is the the

Table 5.12: A reply event created by r_3.

type	time	parent	initiator	receiver	recommendee	TL
3	2.596959	NCD_s	r_3	r_2	NCD_t	2

reply event created by r_3 to r_2. At this time r_2 is not waiting for any other reply from its recommenders, so it forwards this reply to its event parent (i.e. NCD_s). This response is simulated by creating a reply event, as illustrated by 5.13, and inserting it into the calendar. The next event to be picked from the calendar is the reply event created by r_2 to NCD_s. Since NCD_s is not waiting for any other reply from its recommenders and event parent $= -1$, NCD_s creates a complete event and inserts in into the calendar. The final event is

Table 5.13: A reply event created by r_2.

type	time	parent	initiator	receiver	recommendee	TL
3	2.637476	-1	r_2	NCD_s	NCD_t	2

picked and found to be of type COMPLETE. This denotes that NCD_s can now calculate NCD_t's reputation.

After NCD_t's reputation has been computed, NCD_s can combine it with its own direct trust in NCD_t and then decides whether to engage in a transaction with NCD_t. Assume that NCD_s decides to go ahead with the transaction with NCD_t, then NCD_s will perform the following:

- If NCD_s's TM proxy monitored the transaction, then NCD_s's TA will update the accuracy of NCD_s's recommenders. This is done by updating the accuracy field of each recommender in NCD_s's RTT and according to Equation 4.3.

- If NCD_s carried out the consistency check, then NCD_s's TA will update the consistency of NCD_s's recommenders. This is done by updating the consistency field of each recommender in NCD_s's RTT and according to Equation 4.4. Note that NCD_s will not ask recommendations from an inconsistent recommender (i.e., if the recommender's consistency equals 0). Once a recommender's consistency value is 0, NCD_s chooses uniformly at random a new recommender to replace the inconsistent one. By scanning the members of RTT, NCD_s will make sure that the new chosen recommender is not already in its RTT. Table 5.14 shows 2 active recommenders in NCD_s's RTT. The inactive recommenders (i.e. those whose consistency equals 0) will be ignored and will not be asked to give recommendations regarding NCD_t.

- If NCD_s's TM proxy monitored the transaction, then NCD_s's TA will update NCD_s's

Table 5.14: A recommender trust table maintained by NCD_s that has active and inactive recommenders.

	Consistency	Accuracy
NCD_i	1	1
NCD_j	0	1
NCD_k	1	1
NCD_m	0	1

CDTT according to Equation 4.1.

- Using Equation 3.5, NCD_s's TA will update NCD_s's PDTT according to Equation 4.1.

5.2.9 Implementation

For the simulation model, we developed our own discrete-event simulator in C language running on the UNIX environment. Our simulation program can be run through the command line. The simulation program is called *trustSim* with arguments passed from the command line to define the design parameters explained in section 5.2.3. Table 5.15 shows the correspondence between the command line arguments and the simulation design parameters. The simulation program is run using the command line as follows:

trustSim -n 30 -r 4 -e 4 -d 15 -t 5 -l 0.5 -c 0 -a 1 -m 5 -o 10 -u 5 -y 0 -p poisson 1.0 -s 6787367 -D DATA > trustSimOut

Where c is a boolean variable taking the values of 0 or 1. If c $=$ 1, then the consistency check is used. Otherwise, the consistency check is not used. Also, a is a boolean variable taking the values of 0 or 1. If a $=$ 1, then the accuracy check is used. Otherwise, the accuracy check is not used. For example, the above command line specifies running trustSim with 30 $NCDs\_num$, 4 $recs\_num$, 4 $allies\_num$, 15 $NCDs\_dishonest$,

5 $transactions\_num$, $alpha = 0.5$, consistency measure is off, accuracy measure is on, mon_freq is done every 5th transaction, con_freq is done every 10 transaction. But since con_check is off, con_freq will be ignored. The $accu\_check$ is done every 5th transaction and ϵ_{RE} is set to 0. The command line indicates that the simulation uses a poisson distribution for the arrival process with mean 1.0 using 678736 as the seed of the random number generator and printing the results of the simulation into a file called trustSimOut.

Table 5.15: Meaning of the command line arguments.

Command line argument	Simulation Design parameter
n	$NCDs\_num$
r	$recs\_num$
e	$allies\_num$
d	$NCDs\_dishonest$
t	$transactions\_num$
l	α
c	$cons\_check$
a	$accu\_check$
m	mon_freq
o	$cons\_freq$
u	$accu\_freq$
y	ϵ_{RE}
p	arrival process
s	seed
D	to indicate where the output (DATA) should be redirected to

We also automated the process of running the simulation by including a batch file that calls a Perl script. The batch file has the following format:

a1c0_005run 30 5 4 4 0 1 0 10

The Perl script is called *a1c0_005run*. This script runs the simulation with the following

parameters: $30NCDs\_num$, $5transactions\_num$, $4recs\_num$, $4allies\_num$, $cons\_check$ is off, $accu\_check$ is on, $\epsilon = 0$, and 10 $reps$. We can easily automate this script to run the simulation for all the other values of transactions per relation. But to shorten the simulation running time, we run different values of transactions per relation simultaneously on multiple processors. For example, we have Perl scripts *a1c0_010run* and *a1c0_025run* for running 10 and 25 transactions per relation, respectively. Note that the Perl script *a1c0_005run* is included in appendix C.

5.2.10 Verification and Validation

Trace File

In this section, we verify the simulation model by examining a trace file that was created as output of a sample run. We ran our simulation program using the command line as follows: **trustSim -n 6 -r 2 -e 2 -d 2 -t 4 -l 0.5 -c 0 -a 0 -m 2 -o 2 -u 2 -y 0 -p poisson 1.0 -s 6787989 -D DATA > trustSimOut**

The trace file is the output of running the simulation model for 6 NCDs, each NCD has 2 recommenders and 2 trusted allies, the consistency and the accuracy checks are turned off to simplify the trace file. Notice that, this simplification does not introduce any errors in verifying and validating the simulation model because: (a) accuracy check does not produce any events and hence does not change the status of the simulation system and (b) consistency check generate events ONLY between NCD_s and its trusted allies and hence does not affect the recommendation tree. Transactions, which take place among the different NCDs, arrive based on POISSON process with mean 1.0 and are set to 4 (i.e., we deterministically set $trans$ to 4). We set the seed for the simulation to be 6787989. The output is written to a trace file called trustSimOut. As shown below, there are 4 behavior trust relations initiated. The 4 relations or transactions are shown in bold. The first relation is from NCD_2 to NCD_1. The source (initiator) NCD is NCD_2 and the target (receiver)

NCD is NCD_1. The other 3 relations are from NCD_3 to NCD_2, from NCD_5 to NCD_0, and from NCD_4 to NCD_5, respectively.

In Section 5.2.7, we discussed the event generation scheme, event structure, and their life cycle in the simulation model. In the following exercise, we trace the events generated by the simulation program and show that trustSim is working as intended. As shown below, the first event type (ET) generated is a relation event (i.e., ET $= 1$). The event was generated at *simulation time* (time) 0.035777 time units and the parent NCD is -1 denoting that the event was generated by the source NCD and not from a recommender. The relation id (ReId) is set to 1 since this is the first relation event to be generated.

We will trace the relation event number 3 because it has the longest life cycle in this sample simulation. At line **(4)** and at simulation time 1.590335, relation number 3 is created by NCD_5 wanting to engage in a trust relationship with NCD_0. The event parent is set to -1 since the relation was initiated by a source NCD. The ReId is set to 3. To seek the reputation of NCD_0, NCD_5 requests recommendations from R_{NCD_5}. Since NCD_5 has 2 recommenders (i.e., NCD_0 and NCD_4), it should create 2 recommendation events. But as Figure 5.2.7, NCD_5 generates only one recommendation event to NCD_4. That is exactly what is expected. Since NCD_5 is seeking the reputation of NCD_0, NCD_0 can not be asked about the reputation of itself.

At line **(10)** and at simulation time 2.539233, NCD_5 sends a recommendation request to its recommender NCD_4. The event type is 2 (i.e., recommendation event), event initiator is NCD_5, event receiver is NCD_4, event parent is set to -1 since the recommendation event is generated by a source NCD and not a recommender, ReId is set to 3, event recommendee is set to NCD_0 meaning that NCD_4 is requested to give recommendation regarding NCD_0, and the event TL is set to -1 since no recommendation regarding the trustworthiness of NCD_0 has been given yet. We should mention that Tables 5.4, 5.5, and 5.6 are generated from this simulation run.

Table 5.5 shows that $DTT_{NCD_4}(NCD_0) = -1$ meaning that NCD_0 is unknown to

NCD_4. Line **(11)** shows that NCD_4 resorted to its recommenders to seek the reputation of NCD_t. The event type is 2 (i.e., recommendation event), event initiator is NCD_4, event receiver is NCD_3, event parent is set to NCD_5 because NCD_4 is still have to report (give recommendation) to its parent NCD_5. , event recommendee is set to NCD_0 meaning that NCD_3 is requested to give recommendation regarding NCD_0, and the event TL is set to -1 since no recommendation regarding the trustworthiness of NCD_0 has been given yet. Note that at line **(11)**, the NCDs visited are 1, 3, 4, and 5 before NCD_4 requests recommendation from NCD_1. This is done because NCD_4 knows for sure it will send a recommendation request to its 2nd recommender (i.e., NCD_1), which is indeed illustrated in Line **(19)**.

Table 5.5 shows that $DTT_{NCD_3}(NCD_0) = -1$ meaning that NCD_0 is unknown to NCD_3. The recommenders of NCD_3 are NCD_0 and NCD_4. Since the event already visited NCD_4 and that the event is seeking the reputation of NCD_0, NCD_3 creates a reply event to NCD_4 as shown in line **(14)**, where the event TL is assigned to $DTT_{NCD_3}(NCD_0)$.

At line **(19)** and at simulation time 3.829821, NCD_4 sends a recommendation request to its recommender NCD_1. The event type is 2 (i.e., recommendation event), event initiator is NCD_4, event receiver is NCD_1, event parent is set to 5 since the recommendation event is generated by NCD_5's recommender, ReId is set to 3, event recommendee is set to NCD_0 meaning that NCD_1 is requested to give recommendation regarding NCD_0, and the event TL is set to -1 since no recommendation regarding the trustworthiness of NCD_0 has been given yet. Table 5.5 shows that $DTT_{NCD_1}(NCD_0) = 3$ meaning that NCD_0 is known to NCD_1. Therefore, NCD_1 creates a reply event to NCD_4 as shown in line **(23)**, where $DTT_{NCD_1}(NCD_0) = 3$ is assigned to the event TL. Since NCD_4 still has to report to it parent NCD_5, the event parent is still NCD_5.

Once NCD_4 receives the reply event from NCD_1, it creates a reply event to its parent NCD_5 as shown in line **(26)** as simulation time 6.430470. The event parent is set to -1

denoting that NCD_5 is the root (i.e., the source NCD initiating the relation event). Finally and after receiving the reply from NCD_4, NCD_5 creates a complete event as shown in line **(27)** at simulation time 2.629312.

Start of simulation.....Current clock is 0.000000

... The recommenders for NCD 0 are: 4 5

... The recommenders for NCD 1 are: 0 4

... The recommenders for NCD 2 are: 1 4

... The recommenders for NCD 3 are: 0 4

... The recommenders for NCD 4 are: 1 3

... The recommenders for NCD 5 are: 4 0

	ET	time	initiator	receiver	parent	ReId	recommendee	TL	NCDs Visited
(1)	**1**	**0.035777**	**2**	**1**	**-1**	**1**	**-1**	**-1.00**	
(2)	2	0.669676	2	4	-1	1	1	-1.00	4 2
(3)	**1**	**1.489206**	**3**	**2**	**-1**	**2**	**-1**	**-1.00**	
(4)	**1**	**1.590335**	**5**	**0**	**-1**	**3**	**-1**	**-1.00**	
(5)	2	1.627027	3	0	-1	2	2	-1.00	0 4 3
(6)	2	1.640281	0	5	3	2	2	-1.00	5 0 4 3
(7)	3	2.306767	5	0	3	2	2	3.00	5 0 4 3
(8)	2	2.358854	3	4	-1	2	2	-1.00	0 4 3
(9)	3	2.482933	0	3	-1	2	2	3.00	5 0 4 3
(10)	2	2.539233	5	4	-1	3	0	-1.00	4 5
(11)	2	2.612114	4	3	5	3	0	-1.00	1 3 4 5
(12)	**1**	**2.709337**	**4**	**5**	**-1**	**4**	**-1**	**-1.00**	
(13)	2	2.821272	4	3	1	1	1	-1.00	3 4 2
(14)	3	3.115946	3	4	5	3	0	-1.00	1 3 4 5
(15)	3	3.218860	3	4	1	1	1	5.00	3 4 2
(16)	2	3.243459	4	3	-1	4	5	-1.00	1 3 4

(17)	3	3.534803	4	3	-1	2	2	2.00	0 4 3
(18)	3	3.536438	3	4	-1	4	5	3.00	1 3 4
(19)	2	3.829821	4	1	5	3	0	-1.00	1 3 4 5
(20)	4	3.893825	3	2	-1	2	2	3.00	0 4 3
(21)	3	4.041627	4	2	-1	1	1	5.00	3 4 2
(22)	4	4.730214	2	1	-1	1	1	5.00	3 4 2
(23)	3	5.255717	1	4	5	3	0	3.00	1 3 4 5
(24)	2	5.734436	4	1	-1	4	5	-1.00	1 3 4
(25)	3	5.913321	1	4	-1	4	5	3.00	1 3 4
(26)	3	6.430470	4	5	-1	3	0	3.00	1 3 4 5
(27)	4	7.629312	5	0	-1	3	0	3.00	1 3 4 5
(28)	4	8.268972	4	5	-1	4	5	3.00	1 3 4

End of simulation.....Current clock is 8.268972

Intuition

Observing Tables 5.17 and 5.18, we can notice the following. When $\alpha = 1.0$ and for the same monitoring frequency (i.e., $mon\_freq = x$, where $x = 1, 5, 10, 20$), the success rates should be very close. When $\alpha = 1.0$, reputation is ignored and thus the number of dishonest NCDs is irrelevant. This is indeed what we see in Tables 5.17 and 5.18. For example in Table 5.17: (a) for 0 dishonest NCDs, the success rates in the rows corresponding to $mon\_freq = 1$ and $\alpha = 1.0$ are very close to the success rates in the rows corresponding to $mon\_freq = 1$ and $\alpha = 1.0$ when dishonest NCDs are 15 and (b) for 0 dishonest NCDs, the success rates in the rows corresponding to $mon\_freq = 20$ and $\alpha = 1.0$ are very close to the success rates in the rows corresponding to $mon\_freq = 20$ and $\alpha = 1.0$ when dishonest NCDs are 20. Remember, each point in the table to 5.18 is the result of 10 simulation runs.

Further, the same observation holds for comparing the rows in both tables (i.e., table 5.17 and table 5.18). This observation holds because of the following reason. Table 5.17

uses the accuracy measure and Table 5.18 uses the accuracy and the consistency measure. Both measures are irrelevant when $\alpha = 1.0$ since the reputation is ignored. Observing the two tables, we indeed find that this observation holds. For example: (a) in Table 5.17 and for 0 dishonest NCDs, the success rates in the rows corresponding to $mon\_freq = 1$ and $\alpha = 1.0$ are very close to the success rates in Table 5.18's rows corresponding to $mon\_freq = 1$ and $\alpha = 1.0$ when dishonest NCDs are 15 and (b) in Table 5.17 and for 0 dishonest NCDs, the success rates in the rows corresponding to $mon\_freq = 15$ and $\alpha = 1.0$ are very close to the success rates in Table 5.18's rows corresponding to $mon\_freq = 15$ and $\alpha = 1.0$ when dishonest NCDs are 20.

5.3 Simulation Results and Discussion

5.3.1 Overview

The objective of the simulation experiments carried below is to examine the impact of different parameters on the performance of our behavior trust model. First, we investigate the convergence of PDTT to ADTT (i.e. the success rate) using coherent versus incoherent trust models. As discussed below, we conclude that using incoherent trust model yields a poor success rate and hence the trust model can not learn the trust levels that exist between the NCDs. On the other hand, using a coherent trust model yields a very high success rate and shows that the trust model can predict the trust levels that exist between the NCDs. Second, we further examine the impact of different parameters on the success rate of a coherent trust model. These parameters include accuracy, consistency, monitoring frequency, number of dishonest NCDs, direct trust, reputation, and number of transactions per relation. Sections 5.3.2 and 5.3.3 discuss the simulation experiments in detail.

Table 5.16: Using the accuracy and consistency measures: Success rate using a coherent and incoherent trust models using 150 transactions per relation.

Trust model	Monitoring frequency	α value	Number of dishonest NCDs out of 30 NCDs		
			0	15	20
Coherent	1	1.0	100.00%	100.00%	100.00%
		0.5	100.00%	100.00%	100.00%
		0.0	100.00%	100.00%	100.00%
	5	1.0	100.00%	100.00%	100.00%
		0.5	100.00%	100.00%	100.00%
		0.0	90.03%	91.78%	94.01%
	10	1.0	92.10%	92.19%	92.13%
		0.5	99.01%	100.00%	100.00%
		0.0	88.14%	88.91%	89.47%
Incoherent	1	1.0	49.08%	50.34%	50.00%
		0.5	48.85%	50.22%	49.54%
		0.0	51.26%	50.46%	50.34%
	5	1.0	49.77%	50.69%	49.89%
		0.5	49.89%	50.00%	50.11%
		0.0	50.46%	50.11%	49.89%
	10	1.0	50.11%	50.34%	50.11%
		0.5	49.77%	49.89%	50.00%
		0.0	51.29%	49.85%	49.77%

5.3.2 Coherent versus Incoherent Trust Models

In this section, we investigate the dependence of the trust model on coherent versus incoherent DTT. Please refer to Section 4.3.2 for the definition of coherent versus incoherent trust models. It should be noted that the results in Table 5.16 were obtained using the accuracy and the consistency measures as well as using 150 transaction per relation.

When the trust model is coherent, the *success rate* is very high and it is in the range of

88.14% to 100.00% as shown in Table 5.16. That is, filtering is done to isolate the dishonest recommenders. When monitoring is done every transaction, the CDTT converges quickly to the ADTT (i.e., the CDTT will have the actual trust levels). In addition, the increase of dishonest NCDs will have lesser effect because dishonest recommenders are filtered out and recommendations are adjusted at every transaction to reflect the recommender's accuracy. On the other hand, as the monitoring interval is increased, the *success rate* drops to 88.14%. Although dishonest recommenders are being isolated, the recommendations inaccuracies are having a greater impact on the overall performance of the trust model. This becomes especially clear when the monitoring is done every 10 transactions.

When the DTT is incoherent, the *success rate* is always around 50%. This shows that an NCD is not learning actual trust because it is getting conflicting reports on other NCDs and hence the low *success rate*. Further, this low *success rate* is not affected by a variation of the number of dishonest NCDs.

Now, using the coherent trust model, we examine the response of the success rate to the number of transactions per relation when using accuracy alone and accuracy plus consistency. We examine this next.

5.3.3 Agility of the Trust Model

It should be noted that when the frequency of the monitoring process (mon_freq) is set to 1, 5, 10, or 20; we mean that the TM proxy monitoring is done every, every fifth, every tenth, or every twentieth transaction, respectively. Figures 5.4 to 5.6 show the *success rate* of the trust model when using the accuracy alone. The results in the figures correspond to the numerical *success rates* in Table 5.17. Figure 5.4 illustrates the *success rate* when there are zero dishonest NCDs. It can be observed that combining direct trust and reputation (i.e., when $\alpha = 0.5$), outperforms the others (i.e., when $\alpha = 1.0$ or when $\alpha = 0.0$).

Because all recommenders are honest, reputation reinforces direct trust and therefore combining these two components yields a higher *success rate* than relying on only one of them. For example, when the monitoring frequency $= 1$ and the number of transactions $= 10$, the *success rate* reaches 85.61% when $\alpha = 1.0$ and 87.15% when $\alpha = 0.0$ but 96.90% when relying on both, as illustarted in Table 5.17 when there are zero dishonest NCDs. We can also observe that as the monitoring frequency is decreased, the trust model takes longer to reach an acceptable *success rate*. The bold *success rates* in Tables 5.17 and 5.18 represent acceptable *success rates* and they are 85.00% or more. Remember that the accuracy measure is the difference between a recommender's opinion and the true trust level obtained by the TM proxy. Hence, the more frequent is the monitoring process, the more effective is the accuracy measure and that is the reason behind the delay in reaching an acceptable *success rate* value as the monitoring process is relaxed.

However, as illustrated in Figures 5.5 and 5.6, we can observe the following. As the number of dishonest NCDs increases to 15 or 20, relying on reputation only gives poor *success rate* and the accuracy measure loses its effectiveness. Whereas relying on direct trust is not affected by the increase of the dishonest NCDs and keeps its *success rate* almost the same as in the case when there is zero dishonest NCDs. When combining both components (i.e. direct trust and reputation), reputation lowers the *success rate* of the direct trust because of the negative impact it has on direct trust. For example and as illustrated in Table 5.17, when monitoring frequency $= 1$, the number of dishonest NCDs $= 15$, and the number of transactions $= 25$, the *success rate* reaches 97.59% when $\alpha = 1.0$ and 61.84% when $\alpha = 0.0$ but only 91.61% when relying on both.

Therefore, we can conclude that relying on direct trust converges to an acceptable *success rate*. Relying on direct trust , however does not exploit cooperation which is a main goal of P2P systems. On the other hand, a reputation-based model can converge to a high *success rate* but as the number of dishonest NCDs increases, the trust model becomes sensitive to these dishonest NCDs.

To reduce the trust model's sensitivity to dishonest NCDs, we incorporate the accuracy as well as consistency measures so that dishonest NCDs can be filtered out from the recommenders set as soon as detected. The results of this approach are shown in Figures 5.7 to 5.9. The results in these figures correspond to the numerical *success rates* in Table 5.18. In Figure 5.7, when there are zero dishonest NCDs, it can be observed that the consistency measure has no effect since the number of dishonest NCDs is zero. Therefore, the *success rates* are very similar to those in Figure 5.4 when just the accuracy measure is used.

However, as illustrated in Figures 5.8 and 5.9, we can observe the following. As the number of dishonest NCDs increases to 15 or 20, relying on reputation only gives a much higher *success rate* than when using just the accuracy measure. For example, when the monitoring frequency $= 1$ and the number of dishonest NCDs $= 20$, we observe the following: (a) when the number of transactions $= 5$, the *success rate* reaches 82.18% in Table 5.18 and only 49.89% in Table 5.17 and (b) when the number of transactions $= 10$, the *success rate* reaches 92.76% in Table 5.18 and only 53.45% in Table 5.17. This clearly shows that the consistency measure is more effective in dealing with the dishonest NCDs.

Also, as the number of dishonest NCDs increases, we observe a contradictory scenario compared to Table 5.17. That is, combining both components (i.e. direct trust and reputation) gives a higher *success rate* than relying only on one of them. For example, when the monitoring frequency $= 5$, the number of dishonest NCDs $= 15$, and the number of transactions $= 25$, the *success rate* reaches 76.32% when $\alpha = 1.0$ and 80.00% when $\alpha = 0.0$ but 91.26% when relying on both. This is apparent especially when the number of transactions exceeds 5.

We can conclude that once the dishonest recommenders are filtered out form the recommendation sets , reputation reinforces direct trust and therefore combining these two components yields a higher *success rate* than relying on only one of them. Therefore, incorporating the consistency measure into the trust model can limit the effect of dishonest NCDs on the overall performance of the trust model and also speeds the convergence of

the *success rate* to a higher value.

The bold areas in Table 5.17 and Table 5.18 further show the benefit of incorporating the consistency measure into the trust model. Tables 5.17 and 5.18 show the *success rate* of the trust model when using the accuracy alone and accuracy and consistency together, respectively. In each table, there are basically three categories: when the number of dishonest NCDs equals 0, 15, and 20. The percentages in the shaded areas where the number of dishonest NCDs $= 0$ are the maximum *success rate* that can be obtained among these three categories for the same monitoring process, α value, and number of transactions. Since there are no dishonest NCDs to pollute the recommenders network, the *success rate* should be at its maximum value.

In Table 5.17 and as the number of dishonest NCDs increases, the bold area starts to shrink. This shows that the accuracy measure is not effective in limiting and preventing the dishonest NCDs from influencing the recommenders set. On the other hand, Table 5.18 show that the bold area is the same in all of the three categories. Hence, the consistency measure again shows its superiority over the accuracy measure.

5.3.4 Remarks

We conducted simulation experiments to evaluate the performance of our behavior trust model. In addition to the design and exogenous parameters used above, a generalization of the parameters used or even new parameters might have been used in the simulation. For example, we used a fixed set of recommenders. We expect that as the size of the recommendation set increases, there are two intuitions (we should mention that the case where there are no dishonest NCDs is not realistic. So, the assumption is there exists some dishonest NCDs and their objective is to pollute the recommendation network): (a) if the consistency mechanism is not being used, then the *success rate* of our trust model degrades. This is because the dishonest recommenders are not being filtered out from the

recommendation network, and (b) If the consistency mechanism is used, then the *success rate* of our trust model improves. This is because the number of honest recommenders are increasing and that has a great impact on correctly estimating the reputation of the target NCD. In Section 5.4, we carried out simulation experiments to evaluate the variation of the recommendation set of the performance of our trust model.

For the consistency determination parameter ϵ_{RE}, it is realistic to set it to 0. There is no reason for an honest recommender to give away inconsistent trust levels. A flexibility on ϵ_{RE} will just distinguish big liars from small liars. A liar, whether big or small, is considered in our trust model as dishonest.

We know that estimating trust levels is an important issue, where the objective is to observe a sequence of past values of a trust parameter and determine the future estimates. There are different different algorithms that can be used to update the trust parameters. This concept is investigated in more detail in Section 5.5.

5.4 Simulating Recommender Set Variation

5.4.1 Simulation Objective and Setup

The objective of the simulations carried out in this section is to examine the impact of increasing the size of the recommendation set on the overall performance of our trust model. We repeated the same simulation set up as in Section 5.2 with the following changes: (a) the design parameter $NCDs\_num$ is set to 30, (b) the design parameter $NCDs\_dishonest$ is set to 15 (i.e., 50% of the NCDs are dishonest, (c) since we want to investigate the impact of the recommenders set variation on the success rate of the trust model, the design parameter α is set to 0.0, (d) the design parameter mon_freq was set at 1 and 5, and (e) we varied the design parameter $recs\_num$ to take the values 2, 4, and 8. The rest of the simulation is organized as in Sections 5.2.

5.4.2 Simulation Results and Discussion

Figures 5.10 and 5.11 show how the variation in the recommender set R affects the *success rate*. The simulation results are shown in Figures 5.10 and 5.11 are obtained for 15 dishonest NCDs out of 30 NCDs. When using accuracy alone, the *success rate* decreases as the size of the recommendation set increases. This is due to the fact that dishonest recommenders are not filtered out from the recommendation set and their effect increases as the size of the recommendation set increases. On the other hand, Table 5.11 shows that as the size of the recommendation set increase, the *success rate* increases as well. This shows the superiority and the effectiveness of using the consistency measure in preventing dishonest NCDs from influencing the recommendation network.

5.5 Simulating Updating Parameters

5.5.1 Goals of the Simulation

Estimating trust levels is an important issue, where the objective is to observe a sequence of past values of a trust parameter and determine the future estimates. In Chapter 4, Section 4.3.3, we use an *exponential weighted moving average* (EWMA) process for estimating trust levels. The EWMA filter produces an estimate given by:

$$O_t = \omega O_{t-1} + (1 - \omega) O_c \tag{5.3}$$

where $0 \leq \omega \leq 1$, O_t is the newly generated estimate, O_{t-1} is the prior estimate, and O_c is the newly generated observation. If ω is large, the filter resists rapid changes in individual observations and said to provide stability. For low ω values, the filter is able to detect changes quickly and said to be agile. In [55, 91], these filters were combined to create a *flipflop* filter. A *flipflop* filter consists of two EWMA filters. One is agile with ω of

0.1 and the other is stable with ω of 0.9. A controller makes a decision to select between the two filters such that it selects the agile filter when possible, but falls back to the stable filter when new observations are unusually noisy.

In our trust model, there are trust parameters that are updated such as the trust level in DTTs, consistency and accuracy of recommender. The algorithm chosen to update the trust parameters is very important for the following reasons. First, depending on the nature of the parameter, a specific update algorithm might be preferred over another. For example, we can not use a flip flop filter, which applies an agile filter as soon as quick changes are detected. Although an agile filter is appropriate for a quick drop in the trust level, an agile filter is not appropriate for a quick increase in the trust level. The reason behind this is that As known that trust is difficult to build and easy to lose [10]. Second, using a specific update algorithm might not be suitable in detecting dishonest NCDs. For example, using EWMA would return high estimates despite the periodic occurrence of low values in the sequence (i.e., an NCD can periodically cheat and still maintain a high trust level).

At the end of Section 3.6.2, we outlined the updates that need to be done after each transaction. We detailed in Section 4.3.3 how the update algorithm, namely EWMA, is used to update direct trust, recommendation error, and recommenders' accuracy. Other update algorithms (instead of EWMA) can be used to update direct trust, recommendation error, and recommenders' accuracy. In this simulation study, we compare 4 update schemes using a setup where two NCDs are directly connected to each other such that one is being the source NCD (NCD_s) and the other is being the target NCD (NCD_t).

5.5.2 Update Algorithms

As mentioned earlier that an EWMA update scheme or a flip flop update scheme can be used for estimating trust levels. Further, we introduce 3 other update algorithms to update the trust level. We use a variation of the *flipflop*. That is, we introduce a *modified flipflop*

(MFF), where the agile filter is activated as soon as we detect a drop in value of the trust parameter beyond an acceptable threshold from the previously estimated value. The agile filter quickly downgrades the estimate. For the subsequent estimates, we switch back to the stable filter assuming that the trust parameter does not experience any further depreciations. The MFF is a realization of the trust nature that say trust is difficult to build and easy to lose. When the trust model experiences a drop in value of the trust parameter, it quickly downgrades the estimate. This simulates the statement that says trust is easy to lose. On the other hand, the trust model uses a stable filter even if there is a sudden increase in the trust parameter. This simulates the statement that says trust is difficult to build.

The previous modification activates the agile filter only for sharp declines in trust levels. It does not track the number of such declines in a specific parameter and consequently does not penalize those NCDs that periodically cheat. We further modify the *flipflop* and introduce a *weighted modified flipflop* (WMFF) to take periodic cheating into considerations by having a history queue that has the past n values of the trust level. When the kth cheating incident is detected, the history queue is modified by applying k low trust levels at the end of the trust queue. Because the history queue is limited to n entries, only $n - k$ entries from the past remains in the history queue. In computing the trust level, the history queue entries are weighted such that the weight increases linearly from the head to the tail. That is, recent trust levels weigh more than old ones.

Another way to penalize those NCDs that periodically cheat is to modify the *flipflop* filter by introducing yet another scheme called *liberally modified flipflop* (LMFF). But this time, we will take a more "liberal" approach and give the target NCD the benefit of the doubt such that the agile filter produces an estimate given by:

$$O_t = \omega^{\kappa} O_c + (1 - \omega^{\kappa}) O_{t-1} \tag{5.4}$$

where κ is the number of times the target NCD cheats. For example, the first time the

target NCD cheats, the source NCD gives the target NCD the benefit of the doubt and still consider it to be trustworthy at a high trust level. As the number of cheats increase, the trustworthiness of the target NCD decreases gracefully.

5.5.3 Simulation

We repeated the same simulation set up as in Section 5.2 with the following changes: (a) the design parameter $NCDs\_num$ is set to 2, (b) the design parameter $alpha$ is set to 1.0 meaning that NCD_s ignores the reputation of NCD_t and depends only on its own experience with NCD_t. Therefore, no recommendations are requested, no accuracy measure is taken, and no consistency measure is taken. The reason behind this is the following. In this simulation study, we focus on comparing the 4 update algorithms to be used to update direct trust between NCD_s and NCD_t as shown in Equation 4.1. The ITL is assumed to be carried by NCD_s every transaction. The update algorithm used in Equation 4.1 is EWMA. We will use the other 3 update algorithms (MFF, WMFF, and LMFF) and compare their performance. In summary, the simulation in this section is organized as in Sections 5.2 except for the following.

- $NCDs\_num = 2, mon\_freq = 1, alpha = 1.0$.
- NCD_s carries ITL every transaction such that: (a) the ITL carried by NCD_s returns a trust level of 5 if NCD_t is trustworthy and (b) the ITL carried by NCD_s returns a trust level of 1 if NCD_t is untrustworthy.
- Use and compare the performance of EWMA, MFF, WMFF, and LMFF as update algorithms used to update direct trust between NCD_s and NCD_t.

The size of the trust queue n is set to 50. The NCD_t is assumed to cheat NCD_s every 10th transaction. That is, when NCD_s carries ITL, it will find that NCD_t is untrustworthy every 10th transaction. Also, the number of transactions that NCD_s initiated with NCD_t is set to

500. The only design parameters that are relevant in this simulation study is $NCDs\_num$, α, and mon_freq.

5.5.4 Verification and Validation

Start of simulation.....

(1) Rel. # 1: ITL returns 5.0, and NCD 0's TL is 5.0, NCD 0 cheated 0 times

(2) Rel. # 2: ITL returns 5.0, and NCD 0's TL is 5.0, NCD 0 cheated 0 times

(3) Rel. # 3: ITL returns 5.0, and NCD 0's TL is 5.0, NCD 0 cheated 0 times

.

.

.

(10) Rel. # 10: ITL returns 1.0, and NCD 0's TL is 4.6, NCD 0 cheated 1 times

(11) Rel. # 11: ITL returns 5.0, and NCD 0's TL is 4.7, NCD 0 cheated 1 times

(12) Rel. # 12: ITL returns 5.0, and NCD 0's TL is 4.7, NCD 0 cheated 1 times

.

.

.

(30) Rel. # 30: ITL returns 1.0, and NCD 0's TL is 4.6, NCD 0 cheated 3 times

.

.

.

(489) Rel. # 489: ITL returns 5.0, and NCD 0's TL is 4.6, NCD 0 cheated 48 times

(490) Rel. # 490: ITL returns 1.0, and NCD 0's TL is 4.6, NCD 0 cheated 49 times

(491) Rel. # 491: ITL returns 5.0, and NCD 0's TL is 4.6, NCD 0 cheated 49 times

.

.

.

(498) Rel. # 498: ITL returns 5.0, and NCD 0's TL is 4.6, NCD 0 cheated 49 times

(499) Rel. # 499: ITL returns 5.0, and NCD 0's TL is 4.6, NCD 0 cheated 49 times

(500) Rel. # 500: ITL returns 1.0, and NCD 0's TL is 4.6, NCD 0 cheated 50 times
End of simulation.....

In addition to the verification and validation techniques discussed in Section 5.2, we examined a trace file to further verify the simulation model carried out in this section. The trace file was created as output of a sample run. We ran our simulation program using the EWMA update algorithm. We can notice that the simulation was run for 500 relations or transactions (i.e., Lines **(1)** starts at Rel. # 1 and line **(9)**ends at 500). As expected NCD_0 cheats every 10th relation. Indeed for the simulation run, we notice that: (a) in line **(10)** and for the 10 relation between NCD_5 and NCD_0, NCD_0 cheats once, (b) in line **(30)** and for the 30th relation, NCD_0 cheats 3 times, and (c) in line **(490)** and for the 490th relation, NCD_0 cheats 49 times.

Further, figure 5.12 uses a EWMA update scheme such that old values weight more that current observations. That is, it uses a stable update scheme with ω of 0.9. Observing Figure 5.12, we notice that the graph is indeed stays stable (almost a straight line) as the number of transactions increase between NCD_s and NCD_t.

In the simulation, it assumed that NCD_t cheats and given 1 by NCD_s's TM proxy every 10th transaction. That the monitoring that NCD_s carries out should return a value of 1 as the ITL of NCD_t. Also, all of the Figures 5.13, 5.14, and 5.16 use an agile filter with ω of 0.1 when experiencing a drop in the trust value. Indeed as expected, all of the Figures 5.13, 5.14, and 5.16 show an immediate drop in the trust level every 10th transaction. One should notice that the drop in the trust level does not reach 1 because old value of the trust level have a weight of 0.1.

5.5.5 Simulation Results and Discussion

In Figure 5.12, we can observe that using EWMA filter does not detect target NCD's periodic cheating. Thus, the source NCD keeps trusting the target NCD at a high trust level

of around 4.6. Figure 5.13 shows the simulation results obtained by applying MFF filter to estimate the trust level between the source and the target NCDs. In the figure, the following pattern can be observed. The target NCD is penalized as soon as it cheats, but then it behaves in a trustworthy manner for n transactions until it gains the trustworthiness of the source NCD and then it repeats the pattern.

Such repeated cheating by the target NCD can be further penalized for the number of cheats by the WMFF filter as shown in Figure 5.14. We can observe that the more times the target NCD cheats, the more transactions it needs to get back its trustworthiness at a high trust level. Figure 5.15 shows a comparison between the above mentioned three trust level estimation schemes.

Figure 5.16 shows the simulation results obtained by applying the LMFF filter. From the figure, we can observe that the number of cheats gracefully decrease the trustworthiness of the target NCD. By comparing the simulation results of applying WMFF and LMFF illustrated in Figures 5.14 and 5.16, respectively, we observe the following. In both figures, the target NCD is given a chance to be trusted at a high trust level. For low numbers of cheats, LMFF is more "liberal" in giving the chance than WMFF. For example, the target NCD still recovers to a trust level of 5 after 10 cheats by using LMFF. On the other hand, it recovers to a trust level of 3.5 after 10 cheats by using WMFF. For large number of cheats, we observe the opposite. LMFF is more "harsh" in giving the chance than WMFF. For example, the target NCD recovers to a trust level of less than 1.5 after 44 cheats by using LMFF. On the other hand, it recovers to a trust level of 2.5 after 44 cheats by using WMFF.

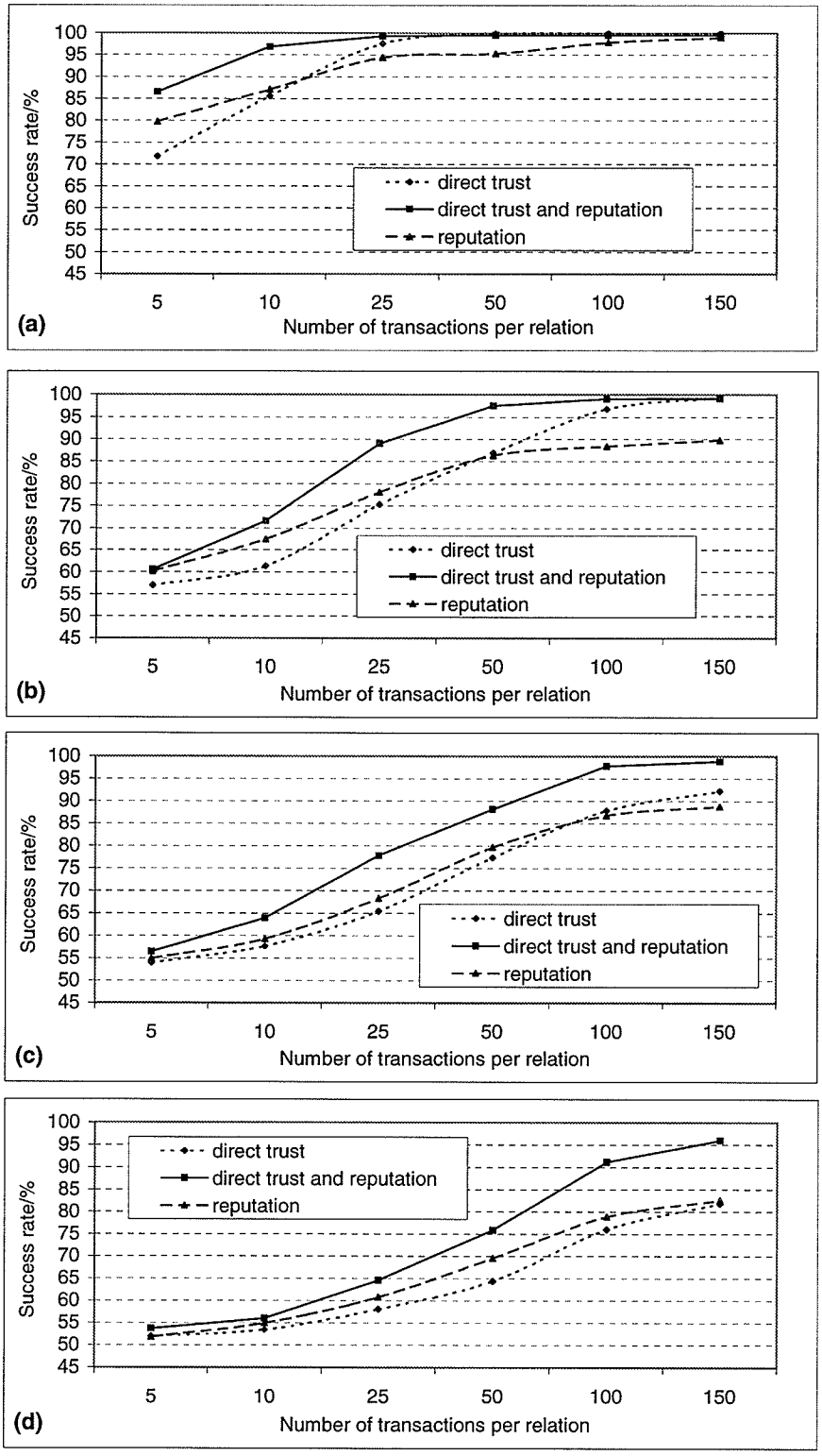


Figure 5.4: For zero dishonest NCDs out of 30 NCDs: Success rate for a coherent trust model using the accuracy measure where the monitoring frequency is: (a) 1, (b) 5, (c) 10, and (d) 20.

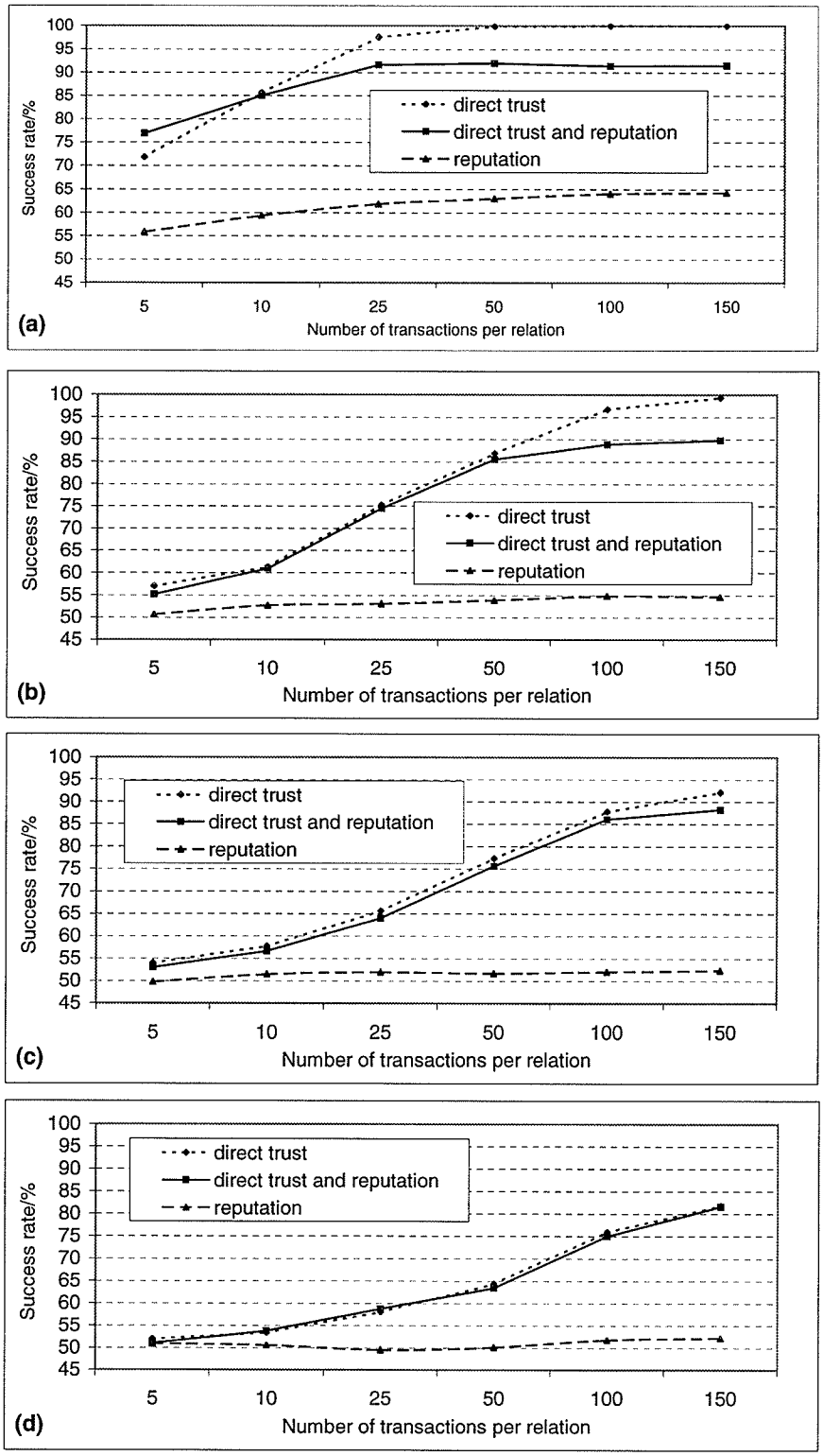


Figure 5.5: For 15 dishonest NCDs out of 30 NCDs: Success rate for a coherent trust model using the accuracy measure where the monitoring frequency is: (a) 1, (b) 5, (c) 10, and (d) 20.

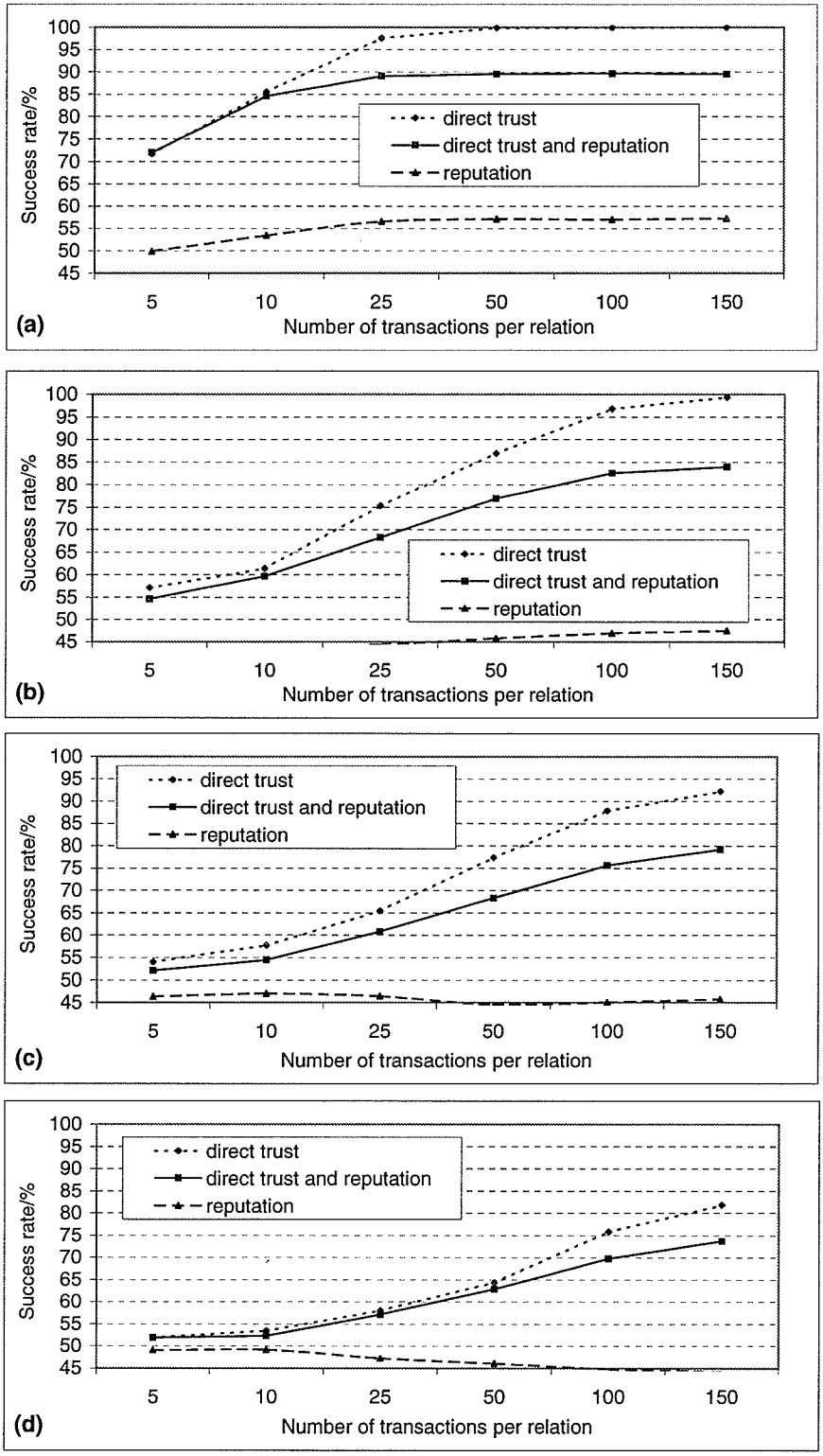


Figure 5.6: For 20 dishonest NCDs out of 30 NCDs: Success rate for a coherent trust model using the accuracy measure where the monitoring frequency is: (a) 1, (b) 5, (c) 10, and (d) 20.

Table 5.17: Success rate for a coherent trust model using the accuracy measure.

Dishonest NCDs	Mon. freq.	α value	Number of transactions per relation					
			5	10	25	50	100	150
0	1	1.0	71.83%	**85.61%**	**97.59%**	**99.86%**	**100.00%**	**100.00%**
		0.5	**86.55%**	**96.90%**	**99.31%**	**99.54%**	**99.54%**	**99.54%**
		0.0	79.77%	**87.15%**	**94.37%**	**95.29%**	**97.90%**	**99.01%**
	5	1.0	57.01%	61.38%	75.29%	**86.90%**	**96.78%**	**99.31%**
		0.5	60.57%	71.61%	**88.97%**	**97.47%**	**99.08%**	**99.11%**
		0.0	60.23%	67.47%	78.05%	**86.32%**	**88.39%**	**89.77%**
	10	1.0	54.02%	57.70%	65.40%	77.36%	**87.82%**	**92.18%**
		0.5	56.44%	63.91%	77.82%	**88.16%**	**97.82%**	**98.85%**
		0.0	54.94%	59.31%	68.28%	79.66%	**86.78%**	**88.74%**
	20	1.0	51.95%	53.45%	58.05%	64.37%	75.98%	81.84%
		0.5	53.68%	55.98%	64.60%	75.86%	**91.15%**	**95.98%**
		0.0	51.84%	54.95%	60.80%	69.54%	78.85%	82.53%
15	1	1.0	71.84%	**85.63%**	**97.59%**	**99.89%**	**100.00%**	**100.00%**
		0.5	76.90%	**85.01%**	**91.61%**	**91.95%**	**91.38%**	**91.49%**
		0.0	55.86%	59.31%	61.84%	62.99%	64.02%	64.25%
	5	1.0	57.01%	61.38%	75.29%	**86.90%**	**96.78%**	**99.31%**
		0.5	55.17%	61.01%	74.48%	**85.52%**	**88.85%**	**89.77%**
		0.0	50.69%	52.76%	53.10%	53.91%	54.83%	54.60%
	10	1.0	54.02%	57.74%	65.60%	77.36%	**87.82%**	**92.18%**
		0.5	52.99%	56.67%	64.01%	75.63%	**86.09%**	**88.28%**
		0.0	49.77%	51.49%	51.95%	51.72%	52.07%	52.41%
	20	1.0	51.95%	53.44%	58.05%	64.37%	75.98%	81.84%
		0.5	51.03%	53.80%	58.74%	63.44%	75.03%	81.61%
		0.0	50.92%	50.57%	49.54%	50.11%	51.84%	52.18%
20	1	1.0	71.67%	**85.56%**	**97.59%**	**99.89%**	**100.00%**	**100.00%**
		0.5	72.01%	**84.60%**	**89.08%**	**89.54%**	**89.62%**	**89.54%**
		0.0	49.89%	53.45%	56.55%	57.13%	57.01%	57.24%
	5	1.0	57.09%	61.38%	75.29%	**86.90%**	**96.82%**	**99.31%**
		0.5	54.60%	59.66%	68.28%	76.90%	82.53%	83.91%
		0.0	44.14%	44.03%	44.37%	45.75%	46.90%	47.47%
	10	1.0	54.02%	57.70%	65.40%	77.36%	**87.82%**	**92.18%**
		0.5	52.07%	54.45%	60.80%	68.28%	75.63%	79.20%
		0.0	46.32%	47.01%	46.44%	44.60%	45.06%	45.75%
	20	1.0	51.95%	53.45%	58.05%	64.37%	75.80%	81.84%
		0.5	51.90%	52.26%	57.13%	62.87%	69.77%	73.68%
		0.0	49.08%	49.20%	47.24%	46.09%	44.83%	44.25%

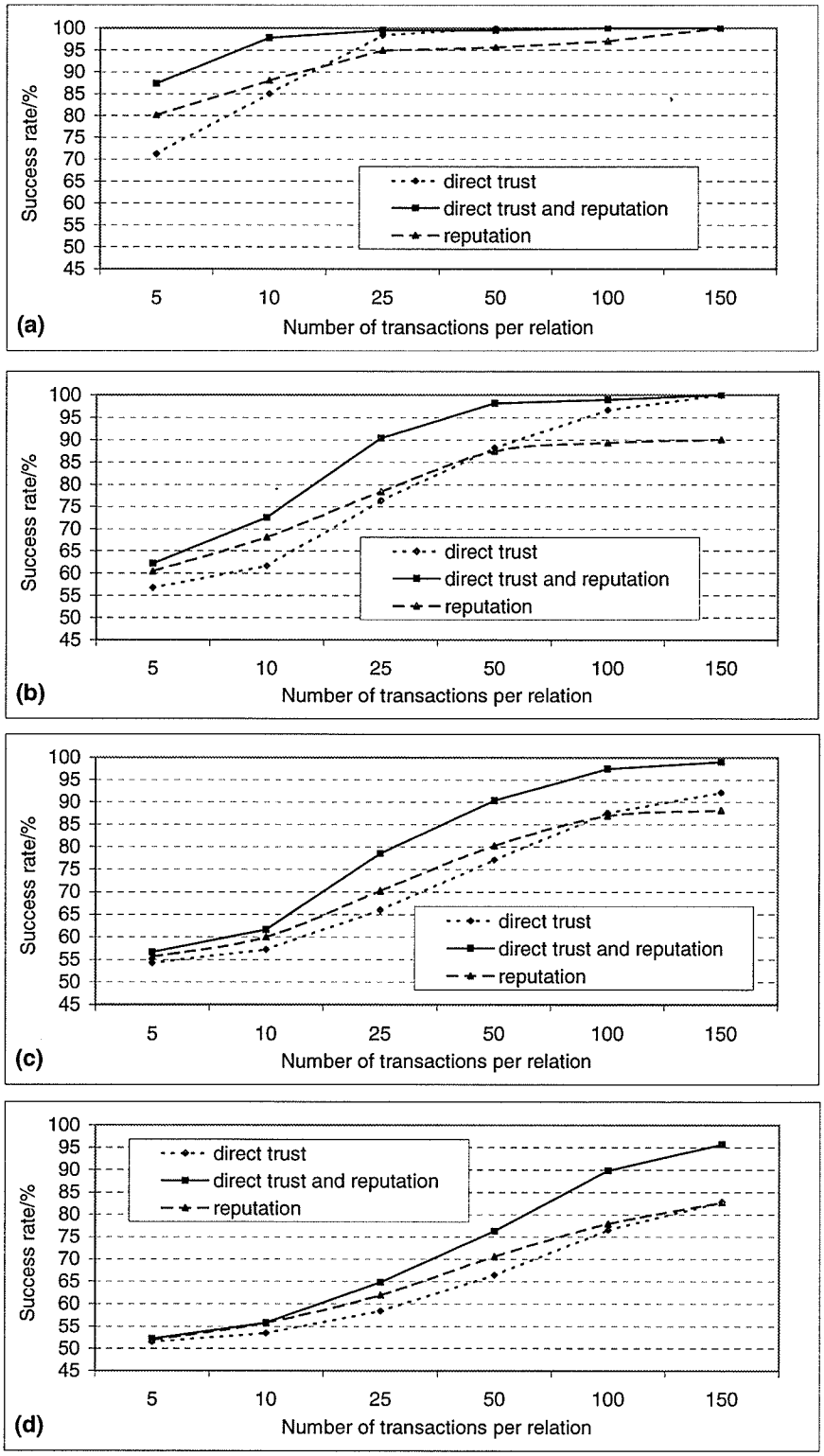


Figure 5.7: For zero dishonest NCDs out of 30 NCDs: Success rate for a coherent trust model using the accuracy and the consistency measures where the monitoring frequency is: (a) 1, (b) 5, (c) 10, and (d) 20.

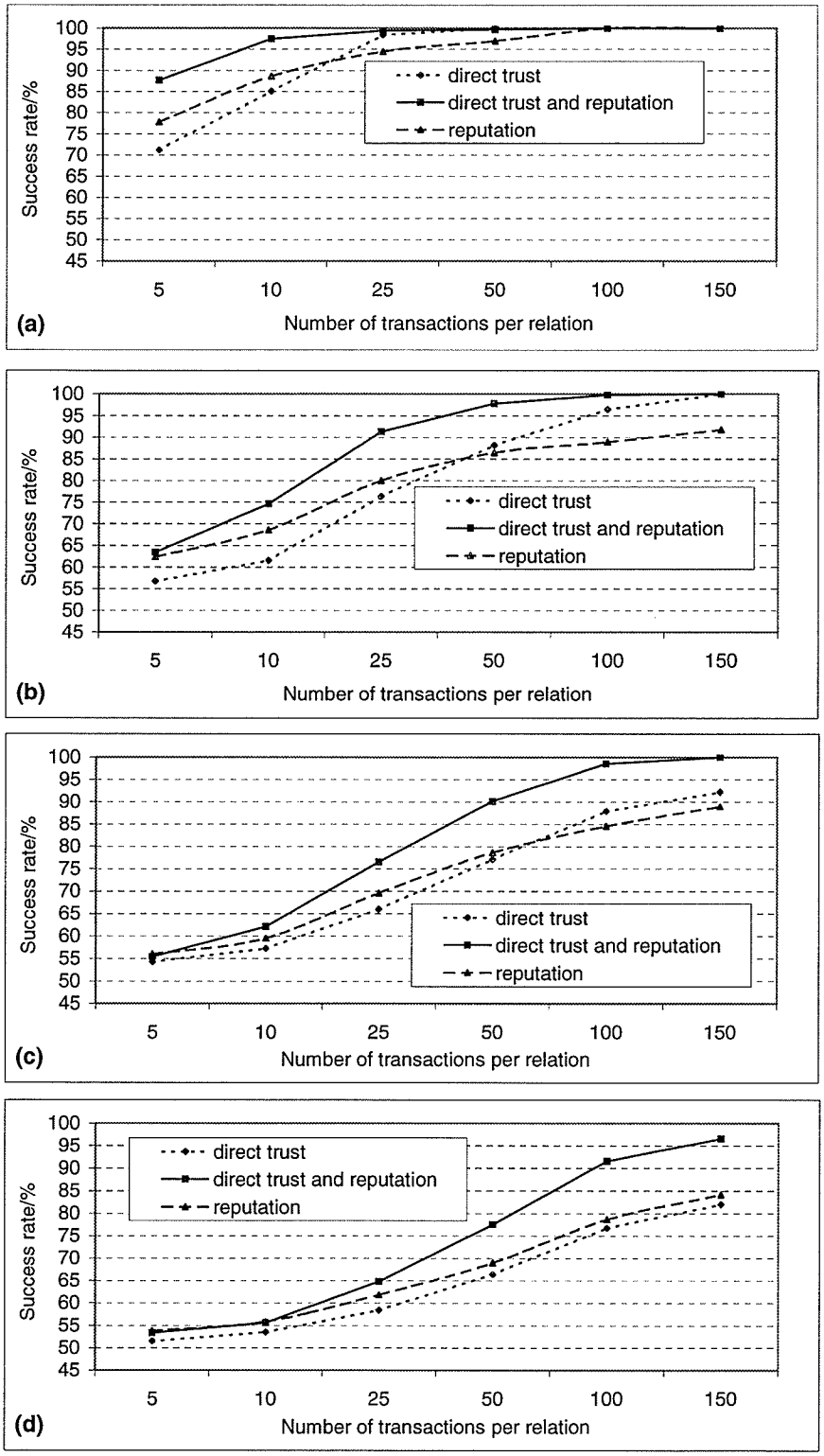


Figure 5.8: For 15 dishonest NCDs out of 30 NCDs: Success rate for a coherent trust model using the accuracy and the consistency measures where the monitoring frequency is: (a) 1, (b) 5, (c) 10, and (d) 20.

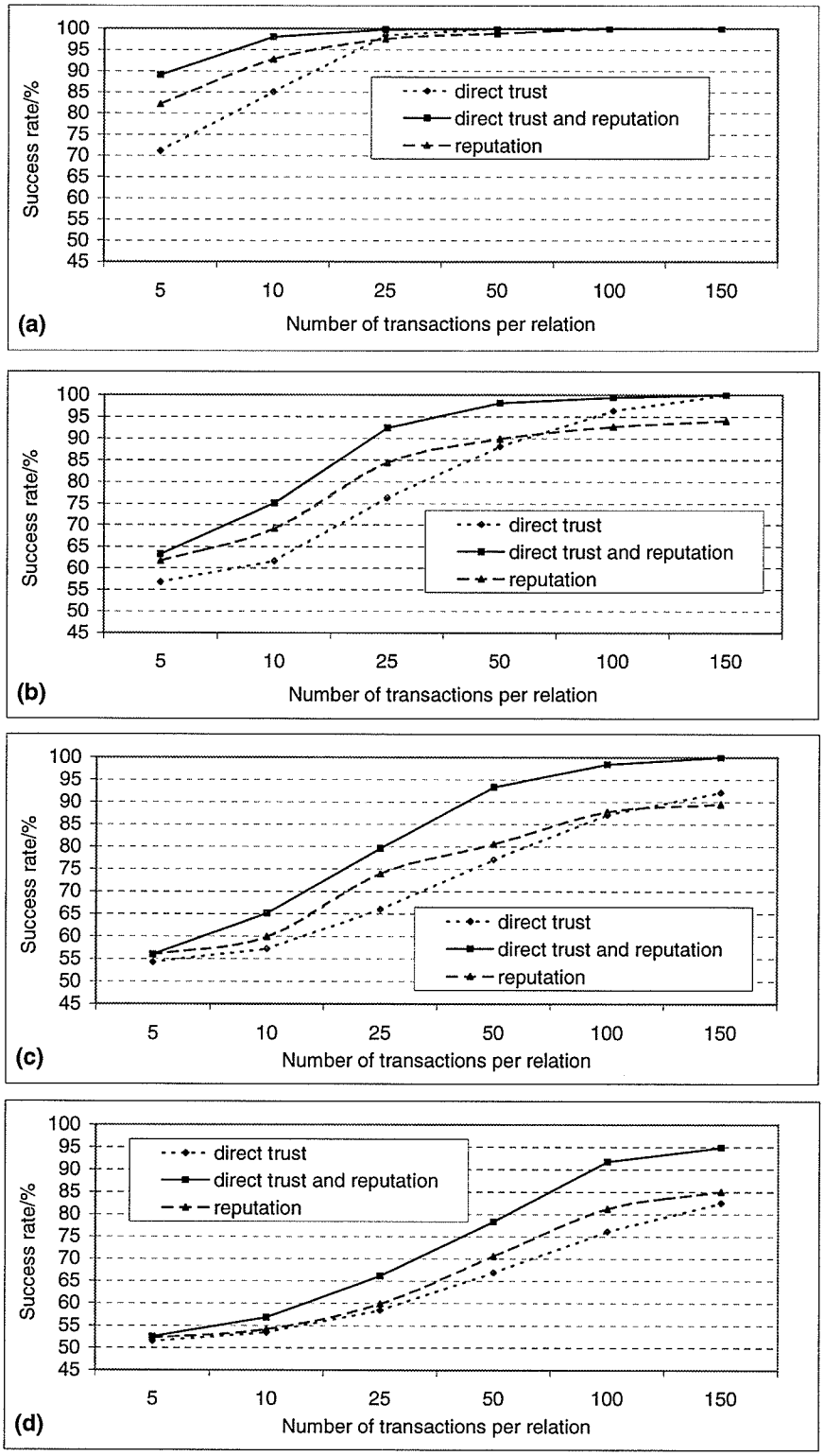
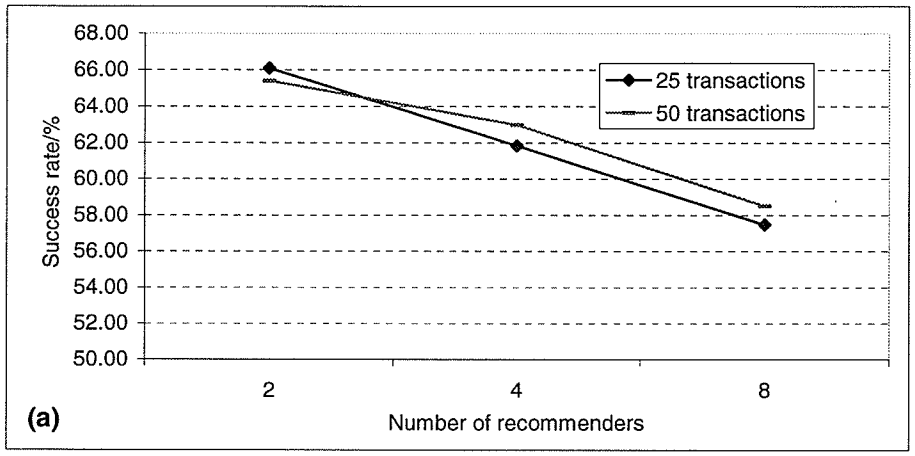


Figure 5.9: For 20 dishonest NCDs out of 30 NCDs: Success rate for a coherent trust model using the accuracy and the consistency measures where the monitoring frequency is: (a) 1, (b) 5, (c) 10, and (d) 20.

Table 5.18: Success rate for a coherent trust model using the accuracy and the consistency measures.

Dis-honest NCDs	Mon. freq.	α value	Number of transactions per relation					
			5	10	25	50	100	150
0	1	1.0	71.22%	**85.05%**	**98.39%**	**100.00%**	**100.00%**	**100.00%**
		0.5	**87.36%**	**97.82%**	**99.54%**	**99.54%**	**100.00%**	**100.00%**
		0.0	80.15%	**88.08%**	**94.94%**	**95.67%**	**97.09%**	**100.00%**
	5	1.0	56.78%	61.68%	76.32%	**88.16%**	**96.67%**	**100.00%**
		0.5	62.18%	72.53%	**90.38%**	**98.16%**	**98.97%**	**100.00%**
		0.0	60.46%	68.16%	78.34%	**87.47%**	**89.34%**	**90.03%**
	10	1.0	54.28%	57.24%	65.98%	77.13%	**87.55%**	**92.10%**
		0.5	56.67%	61.61%	78.51%	**90.34%**	**97.46%**	**99.01%**
		0.0	55.63%	60.00%	70.23%	80.23%	**86.89%**	**88.14%**
	20	1.0	51.49%	53.42%	58.39%	66.44%	76.55%	82.79%
		0.5	52.18%	55.75%	64.83%	76.32%	**89.86%**	**95.66%**
		0.0	51.84%	55.74%	61.95%	70.57%	77.93%	82.70%
15	1	1.0	71.20%	**85.06%**	**98.39%**	**100.00%**	**100.00%**	**100.00%**
		0.5	**87.70%**	**97.47%**	**99.43%**	**99.67%**	**100.00%**	**100.00%**
		0.0	77.82%	**88.62%**	**94.48%**	**97.01%**	**100.00%**	**100.00%**
	5	1.0	56.78%	61.57%	76.32%	**88.16%**	**96.44%**	**100.00%**
		0.5	63.45%	74.60%	**91.26%**	**97.82%**	**99.78%**	**100.00%**
		0.0	62.41%	68.62%	80.00%	**86.44%**	**88.93%**	**91.78%**
	10	1.0	54.25%	57.24%	65.97%	77.13%	**87.89%**	**92.19%**
		0.5	55.40%	62.07%	76.55%	**90.11%**	**98.53%**	**100.00%**
		0.0	55.98%	59.54%	69.54%	78.62%	**84.57%**	**88.91%**
	20	1.0	51.49%	53.45%	58.39%	66.44%	76.78%	82.02%
		0.5	53.33%	55.63%	64.83%	77.47%	**91.59%**	**96.59%**
		0.0	53.79%	55.86%	61.84%	68.97%	78.61%	84.14%
20	1	1.0	71.12%	**85.05%**	**98.39%**	**100.00%**	**100.00%**	**100.00%**
		0.5	**89.08%**	**98.05%**	**99.77%**	**99.89%**	**100.00%**	**100.00%**
		0.0	82.18%	**92.76%**	**97.59%**	**98.85%**	**100.00%**	**100.00%**
	5	1.0	56.78%	61.68%	76.32%	**88.16%**	**96.36%**	**100.00%**
		0.5	63.22%	75.06%	**92.41%**	**98.16%**	**99.43%**	**100.00%**
		0.0	61.72%	69.20%	84.37%	**89.89%**	**92.66%**	**94.01%**
	10	1.0	54.25%	57.24%	65.98%	77.10%	**87.13%**	**92.13%**
		0.5	55.98%	65.06%	79.54%	**93.33%**	**98.46%**	**100.00%**
		0.0	55.98%	59.89%	73.91%	80.52%	**87.76%**	**89.47%**
	20	1.0	51.49%	53.45%	58.39%	66.86%	76.13%	82.49%
		0.5	52.53%	56.78%	66.09%	78.31%	**91.71%**	**94.93%**
		0.0	52.30%	54.14%	59.89%	70.57%	81.19%	**85.01%**



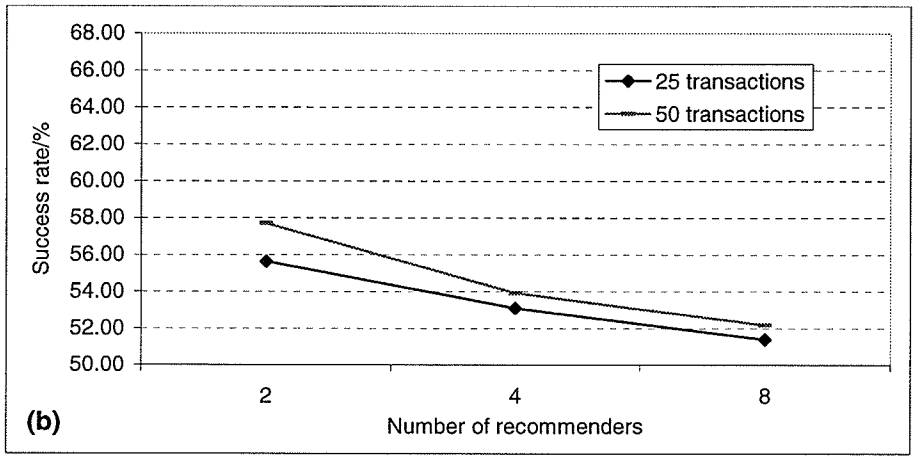


Figure 5.10: Recommenders set variation's affect on success rate for a coherent trust model using the accuracy measure where the monitoring frequency is: (a) 1, (b) 5.

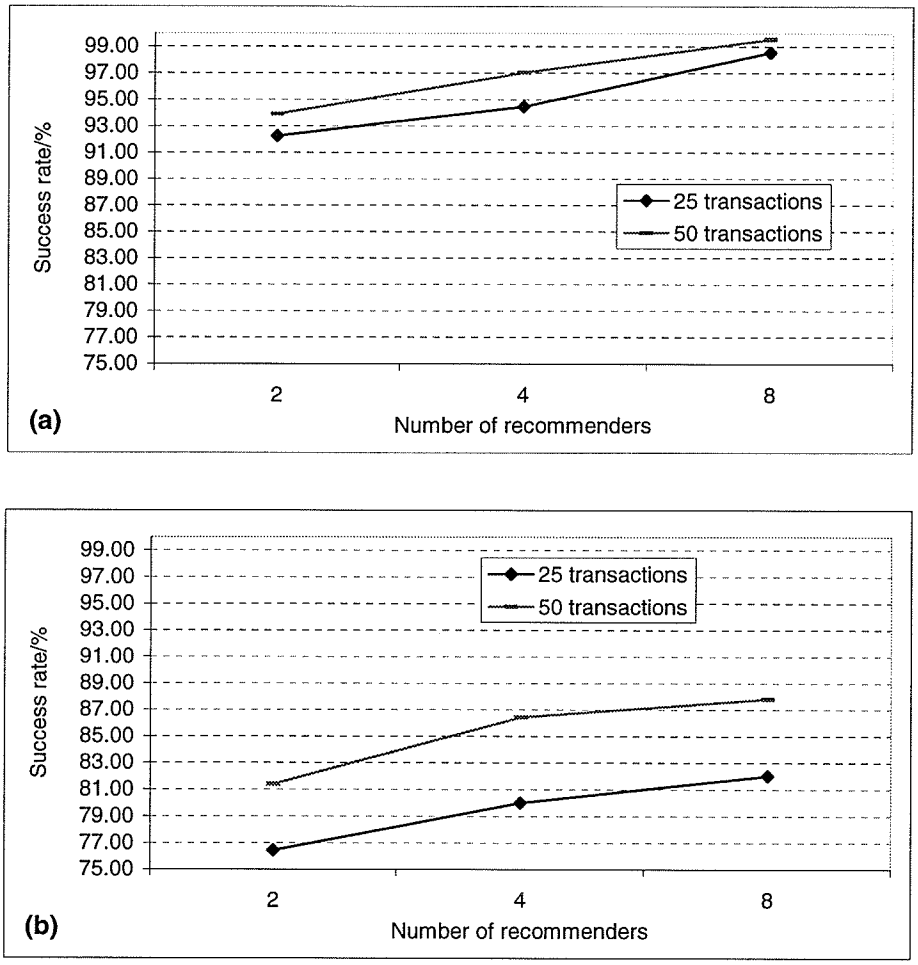


Figure 5.11: Recommenders set variation's affect on success rate for a coherent trust model using the accuracy and consistency measures where the monitoring frequency is: (a) 1, (b) 5.

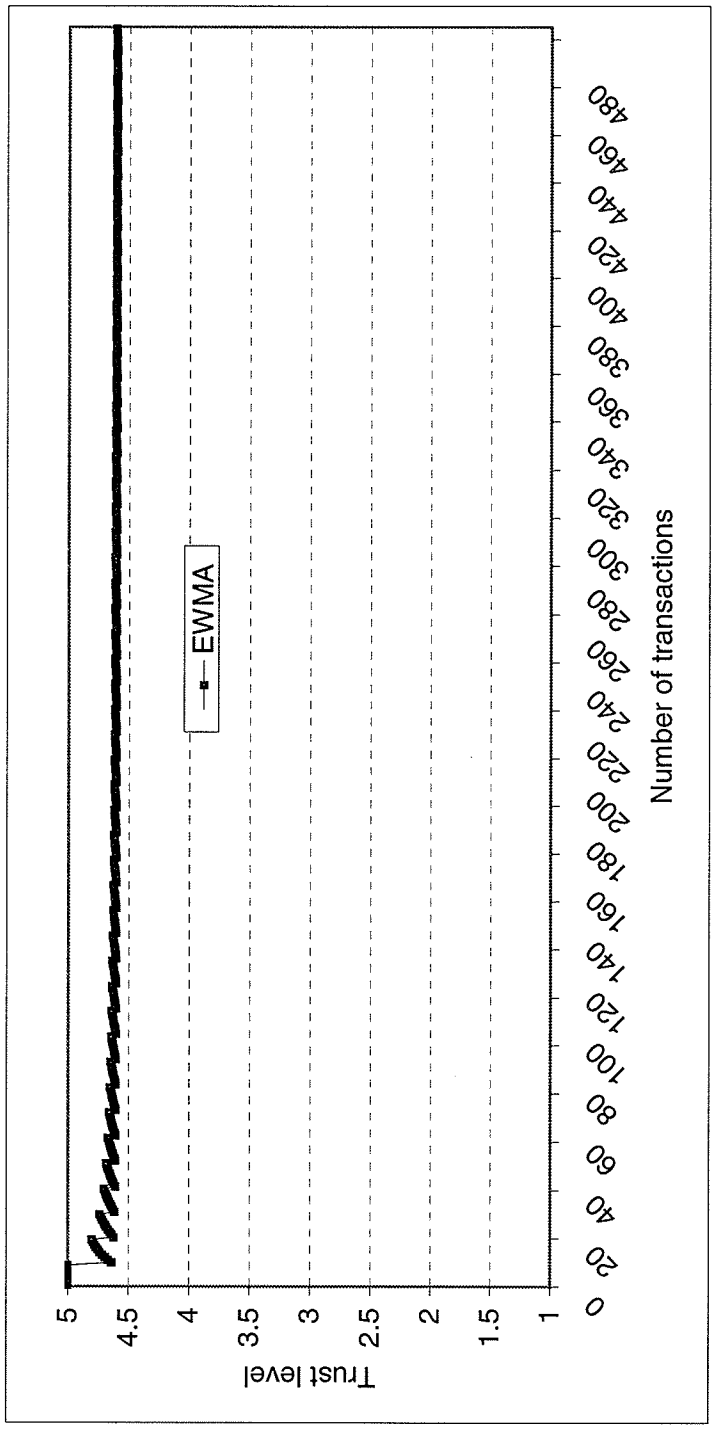


Figure 5.12: Estimating the trust level using an EWMA filter scheme.

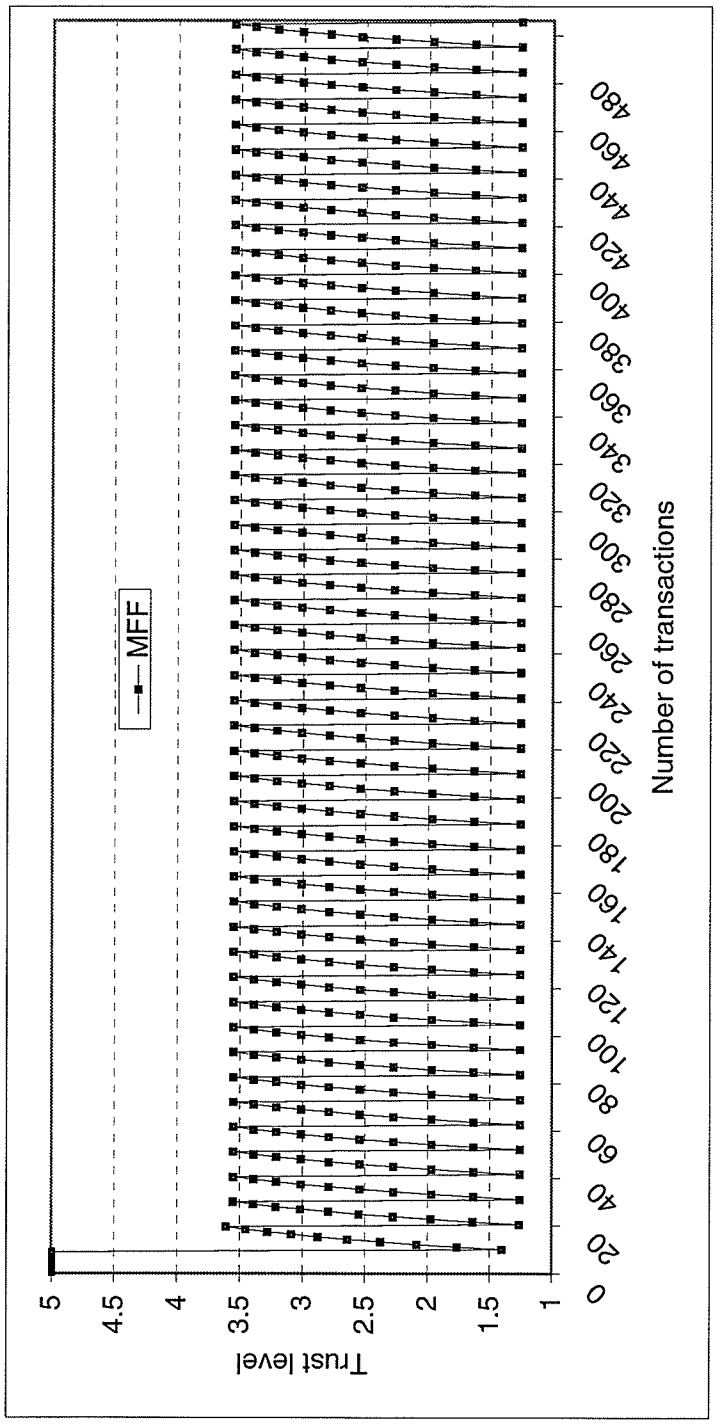


Figure 5.13: Estimating the trust level using a MFF filter scheme.

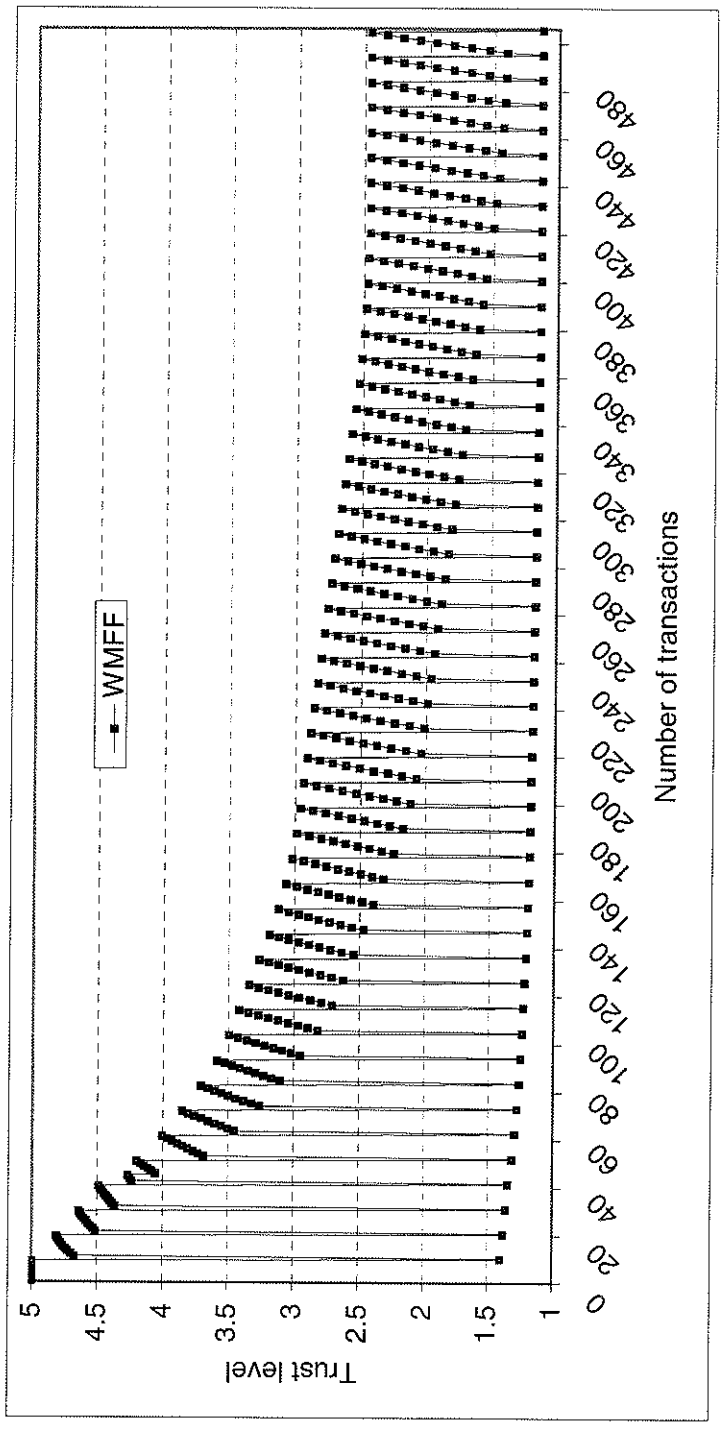


Figure 5.14: Estimating the trust level using a WMFF filter scheme.

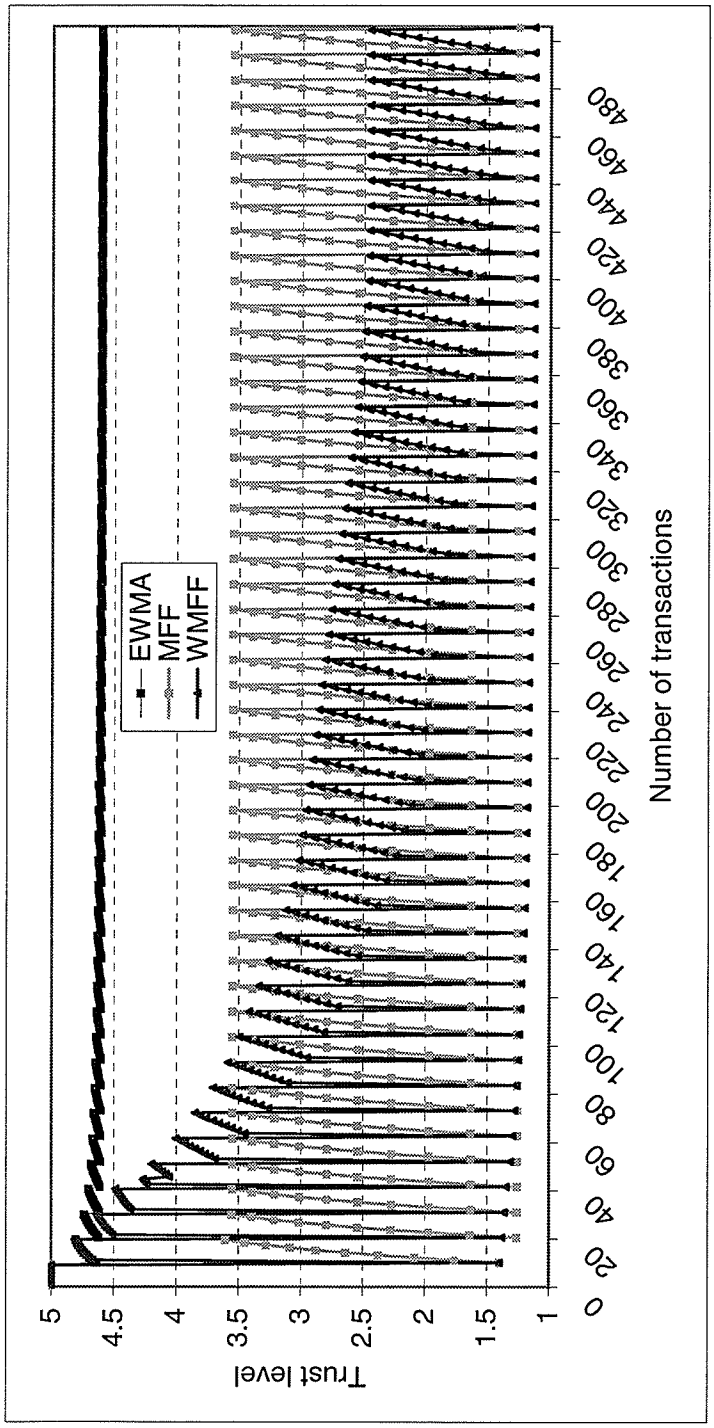


Figure 5.15: Comparison of different schemes used to estimate the trust level.

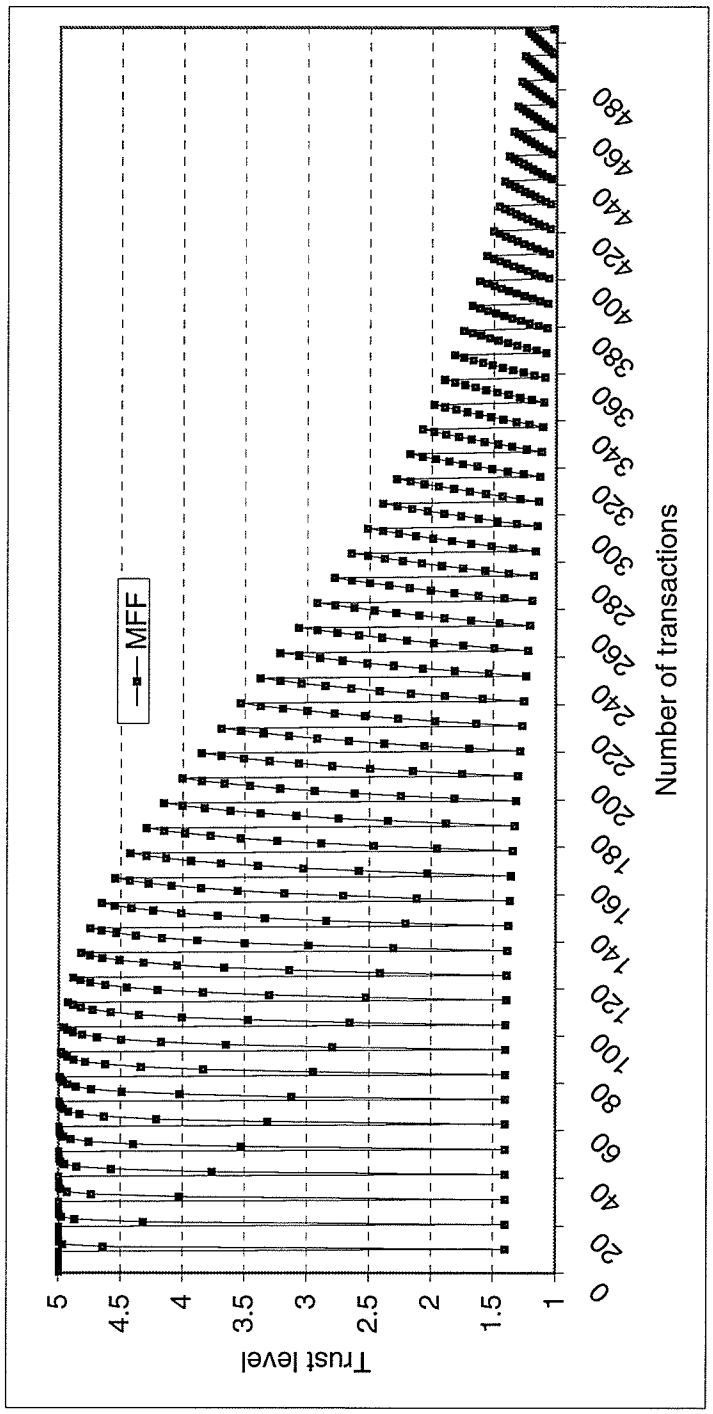


Figure 5.16: Estimating the trust level using a LMFF scheme.

5.6 Summary

This chapter presents a series of simulation results to evaluate the performance of our trust model. These results are published in various conferences and workshops including [61], [36], and [63]. Central to the model is the idea of maintaining a recommender network that can be used to obtain references to predict the trust that exists between two NCDs. Our trust model uses an *accuracy* concept to enable peer review-based mechanisms to function with imprecise trust metrics, the imprecision is introduced by peers evaluating the same situation differently. Simulation results show that a reputation-based trust model reaches an acceptable level of capability after a certain number of transactions. However, as the number of dishonest NCDs increase, the model becomes slow in reaching the acceptable level of capability.

To reduce the trust model's sensitivity to dishonest NCDs, we introduced a *consistency* concept to handle the situation where NCDs intentionally lie about other NCDs for their own benefit. Simulation results indicate that incorporating the *consistency* concept into the trust model, limits the effect of dishonest NCDs by preventing them from providing recommendations.

Another feature of our model is the flexibility to weigh direct trust and reputation differently. Simulation results show that it is better to rely on direct trust when consistency is *not* used. This can be explained by observing that due to dishonest recommenders, the reputation is tainted and using it can only lead to incorrect decisions. When the consistency is used to isolate the dishonest recommenders, we are assured of an honest set of recommenders. In this situation, simulation results indicate that significant benefits can be obtained by using reputation.

Another significant advantage of our model is that it does not depend on a majority opinion as previous models did. Therefore, our model can work even when the majority of the recommenders are dishonest. Actually, as the number of dishonest recommenders

increases, the recommenders providing recommendations to a query reduces. The number of recommenders also provides another measure of trust on the overall system because all the recommenders are considered honest.

We also discussed other parameters that should be investigated such as the variation of the recommendation set and the algorithms used to update the trust parameters. Through simulation, we investigated the affect of varying the recommendation set as well as using certain algorithms to update the trust level parameter. As outlined in the simulation results, the performance of our trust model can be improved by using some of the suggested mechanisms.

Chapter 6

On the Scalability of The Trust Model

6.1 Overview

In mapping the trust model onto P2P structured large-scale network computing system, scalability becomes a vital issue. Scalability means that a system must be deployable in a wide range of scales. Scalability means not just to operate, but to operate efficiently and with adequate *value delivered*, over the given range of scales [92]. The scalability framework is based on a *scaling strategy* for scaling up or scaling down the trust model. The scaling strategy is controlled by a *scaling factor* (k). For example, if the number of NCDs is increased from 30 to 60, we say that the system is scaled up from configuration 1 to configuration 2 by a scaling factor of 2. The system is considered to be scalable from configuration 1 to configuration 2, if the value delivered keeps pace with cost. In our trust model, the value delivered and the cost are derived by examining a behavior trust. A behavior trust of a target NCD is estimated by a source NCD interested to engage in a transaction with the target NCD. Obviously, there is a cost for the source NCD to estimate the target NCD's behavior trust. There is also a value delivered to the source NCD by estimating the target NCD's behavior trust. The value delivered is measured by

how successful is the transaction as far as the source NCD is concerned. The cost of the behavior trust will be discussed in detail in Section 6.2.

It should be mentioned that the objective of this study is to examine the scalability of the trust model at the NCD level. That is, to determine the scalability of the trust model as the number of NCDs increase. Other factors in our behavior trust model will affect its scalability and further investigation is needed to examine their impact on the overall scalability of the model. First: Besides the behavior cost, additional cost incurs at the node level. An NCD manages its nodes (clients or resources) and deploys mechanisms to allow them to join, operate, and leave the pool of clients or resources. This management needs to be done in a scalable and efficient manner because the behavior of the nodes affect the NCD's reputation and hence its interaction with other NCDs. As nodes join or leave an NCD, processing costs due to these operations have to be taken into consideration. Also, managing nodes and monitoring their behavior while they are part of an NCD incur additional processing cost. These costs increase as the number of nodes within an NCD increase. Second: In our trust model, we limited the context to primary service types such as printing, storage, and computation. This reduces the fragmentation of the trust management space, but the coarse definition of context can result in inaccurate trust level estimates. Depending on the definition of context, a huge amount of data may result. In Section 6.6, we shed more light on the scalability concerns at the node level and at the context level.

So, the goal is to generalize the scalability metric such that the overall cost includes the behavior trust, node level costs, and factors related to context. Therefore, further analysis needs to be done to include the node level cost and the impact of the context definition into the scalability metric. However, what is done in this chapter is a necessary first step towards achieving that goal.

6.2 A Behavior Trust Cost

There is a cost involved in the source NCD (NCD_s) determining the trustworthiness of the target NCD (NCD_t) and engaging in the transaction with NCD_t. We refer to this cost as *behavior trust cost*. It should be mentioned that the *behavior trust cost* will be shown for the worst case scenario. That is, the accuracy, honesty, and monitoring process are assumed to be done for every transaction. In Section 3.6 and specifically in Figure 3.2, we explained the detailed behavior trust steps that NCD_s performs as a result of engaging into a transaction with NCD_t. These detailed steps are repeated in Figure 6.1 for clarity purposes.

Now, let us go through the pseudo-code presented in Figure 6.1 and explain the behavior trust cost involved. In line **(1)**, NCD_s accesses DTT_x to determine the direct trust in NCD_t and then computes $\Theta(NCD_s, NCD_t, t, c)$ by possibly decaying the trust level obtained from DTT_x. Hence, there is an *access cost* plus a *computational cost* involved. We will skip the behavior trust cost involved in line **(2)** through line **(4)** for now and we will consider them in Section 6.2.1. Lines **(5)** through **(13)** involve *computational cost* with the exception of line **(10)**, which involves a *monitoring cost*.

From the above costs, we observe that these costs do not depend on the number of NCDs. It is obvious that the *computational cost* is independent of the number of NCDs and the *monitor cost* is done irregardless of the number of NCDs. For the *access cost*, NCD_s has to access either DTT_{NCD_s} or RTT_{NCD_s}. The size of RTT_{NCD_s} and DTT_{NCD_s} are very small. Each NCD in the trust model maintains RTT, which is initially chosen randomly and then evolves as explained in previous Chapters. The size of RTT is very small compared to the total number of NCDs and it does not depend on the number of NCDs. Further, RTT_{NCD_s} holds just two entries (consistency and accuracy) for each of NCD_s's recommenders. For the DTT_{NCD_s}, NCD_s maintains one entry for each NCD_t directly interacted with. Therefore, as the direct interaction with different NCDs increases,

Source entity x computes the behavior trust in target entity y.

(1) **compute** $\Theta(NCD_s, NCD_t, t, c)$;; as detailed in Figure 3.3
(2) **get** $RE_{NCD_s}(NCD_z, NCD_t, t, c)$, $\forall\, NCD_z \in R_{NCD_s}$;; NCD_s gets
;; recommendations regarding NCD_t
(3) **if** ((trans_num **mod** m) $= 0$)
(4) **consistency** check ;; as detailed in Figure 3.4
(5) **adjust** recommendations ;; as detailed in Figure 3.5
(6) $\Omega_{NCD_s}(NCD_t, t, c) \leftarrow \frac{\sum_{NCD_z \in R_{NCD_s}} \Upsilon(t-\tau_{NCD_z\ NCD_t}(t),c)\ S(NCD_s,NCD_t,NCD_z,t,c)}{|R_{NCD_s}|}$,
$NCD_z \neq NCD_t$;; as illustrated in Equation 3.7
(7) $\Gamma(NCD_s, NCD_t, t, c) \leftarrow \alpha\, \Theta(NCD_s, NCD_t, t, c) + \beta\, \Omega_{NCD_s}(NCD_t, t, c)$
;; as illustrated in Equation 3.5
(8) **if** NCD_s decides to proceed with the transaction
(9) **if** ((trans_num **mod** n) $==$ 0) ;; accuracy check
(10) **obtain** $ITL_{NCD_s}(NCD_t, t, c)$
(11) **else**
(12) NCD_s rejects the transaction with NCD_t
(13) **update** process ;; as detailed in Figure 3.6

Figure 6.1: Behavior trust steps.

DTT_{NCD_s} size increases in a linear fashion. Assuming that NCD_s interacted at least once with every other NCD, the maximum number of NCDs in DTT_{NCD_s} is the total number on NCDs minus 1.

We conducted a study to examine and measure the accessing costs incurred when accessing RTT_{NCD_s} and DTT_{NCD_s}. The size of RTT_{NCD_s} is set to 4. Let us consider the worst case scenario and assume that NCD_s directly interacted with all other NCDs. Therefore, the size of DTT_{NCD_s} is set to the maximum number of NCDs minus 1. We also assume a sequential access to DTT_{NCD_s} and RTT_{NCD_s}. The machine used to in this study is based on an Intel Pentium IV processor running at 1.3 GHz with memory size of 256 MB and a level 2 cache of size 256 KB. Table 6.1 shows the cost in seconds for accessing RTT_{NCD_s} and DTT_{NCD_s} with different sizes. The access cost depends largely on the size

Table 6.1: Access costs for DTT_{NCD_s} and RTT_{NCD_s}.

	Description	Cost (in seconds)
accessing RTT	size of RTT is 4	0.5×10^{-6}
accessing DTT	NCDs' number is 30	1.3×10^{-6}
	NCDs' number is 60	1.4×10^{-6}
	NCDs' number is 90	1.6×10^{-6}

of RTT_{NCD_s} or DTT_{NCD_s}. As mentioned before, the size of RTT_{NCD_s} is independent of the number of NCDs. Hence the access cost of RTT_{NCD_s} can be ignored. Also, as mentioned before, the size of DTT_{NCD_s} increases in a linear fashion as NCD_s interacts with every other NCD and the maximum number of NCDs in DTT_{NCD_s} is the total number on NCDs minus 1. From Table 6.1, as the number of NCDs increased from 30 to 90, the access cost increased only by 0.3×10^{-6}. Because the size of these tables are small and the increase of NCDs does not have much affect on their access cost, the access cost of these two tables can be ignored. The only *behavior trust cost* that we have not considered is the cost involved in lines **(2)** through **(4)** depicted in Figure 6.1. This cost is considered next.

6.2.1 Reputation Cost

In this section, we will focus on the cost illustrated in Figure 6.1, lines **(2)** through **(4)**. This cost, we call *reputation cost*, is further detailed in Figure 6.2. The *access cost*, Figure 6.2, in lines **(1)** and **(2)** can be ignored as explained in Section 6.2. In lines **(3)** to **(5)**, NCD_s carries out the consistency check by instructing its trusted allies in T_{NCD_s} to request recommendations from NCD_s's recommenders regarding NCD_t. The cost associated with the steps in lines **(3)** to **(5)** can be ignored because: (a) the trusted allies set is small, (b) the recommenders set is also small, and (c) each of the NCD_s's recommenders will receive the same recommendation request from NCD_s and from NCD_s's trusted allies. Therefore, all

these recommendation requests will form the same one and only one recommendation tree. Hence, the cost that we are concerned about is the cost associated with line **(7)**, which is expressed in more detail in Figure 6.3. We should mention two points: (a) the creation of

---

Reputation steps incurred by NCD_s acquiring recommendations to determine NCD_t's reputation.

---

(1) NCD_s accesses RTT_{NCD_s} to look up its recommenders
(2) NCD_s accesses T_{NCD_s} to look up its trusted allies
(3) **forall** $NCD_r \in T_{NCD_s}$ **do** ;; forall the trusted allies of NCD_s
(4) **forall** $NCD_z \in RTT_{NCD_s}$ **do** ;; forall recommenders of NCD_s
(5) ;; trusted ally NCD_r sends recommendation request to NCD_z
;; regarding the reputation of NCD_t
(6) $NCDs\_Visited = \{\}$;; No recommender has been sought for help yet by NCD_s
(7) SeekRep(NCD_s, -1, RTT_{NCD_s}, $NCD\_Visited$, NCD_t) ;; NCD_s seeks help from
;; its recommenders to determine the reputation of NCD_t

Figure 6.2: Reputation steps.

the recommendation tree starts from line **(7)** and (b) the cost associated with line **(7)** is due to the creation of the recommendation tree, in which NCD_s is its root node. The function SeekRep, in Figure 6.3, takes four parameters: (a) initiator, which is the node that initiates the recommendation request (i.e., the node that seeks NCD_t's reputation), (b) parent, which is the parent of the initiator, (c) the initiator's set of recommenders, (d) a list of NCDs that are already have seen the recommendation request. We use this list to prevent cycles in the recommendation tree, and (e) the target NCD that its reputation is sought. As we see in Figure 6.2 in line **(7)**, the parent is set to -1 since the initiator is NCD_s, which is the root of the recommendation tree. In Figure 6.3, the initiator starts by requesting recommendations from its recommenders. The recommendation request is sent down the recommendation

tree in line **(5)**. Line **(8)** states that if all of the initiator's recommenders have seen the recommendation request and the initiator is not the root of the recommendation tree, then the initiator sends a request reply to its parent with -1 as the trust value, which means: (a) the initiator does have previous interaction with NCD_t and (b) the initiator's recommenders have been asked for the same request by other NCDs and hence, NCD_s will get an answer from them. Note that each behavior trust relation (i.e., a relation from NCD_s to NCD_t) forms its own recommendation tree. As we discussed earlier in Section 5.2.7 that each relation has its own relation_num to keep track of events generated and match these events to their corresponding trust behavior relationship. Therefore, line **(8)**, states a definition of a leaf node as a recommendation node that does not have previous interaction with NCD_t and all of its recommenders have seen the recommendation request.

Pseudo-code for the SeekRep function, which takes five parameters as follows. SeekRep(initiator, parent, $RTT_{initiator}$, $NCDs\_Visited$, NCD_t), where initiator is the NCD seeking the reputation of NCD_t, parent is the initiator's parent, $RTT_{initiator}$ is the set of the initiator's recommenders, $NCDs\_Visited$ is the NCDs visited by the recommendation request so far, and NCD_t is the target NCD.

SeekRep(initiator, parent, $RTT_{initiator}$, $NCDs\_Visited$, NCD_t)
(1) **forall** ($NCD_z \in RTT_{initiator}$) **do** ;; forall the recommenders of the initiator
(2) RecReq_Sent = 0 ;; the initiator has not sent any recommendation request to any of its recommenders yet
(3) **if** ($NCD_z \in NCDs\_Visited$) ;; if NCD_z has seen the request, do not resend to it
(4) **else**
(5) ;; Rec_Req(initiator, NCD_z, $NCDs\_Visited$ NCD_t) ;; the initiator requests ;; a recommendation from NCD_z regarding the reputation of NCD_t
(6) $NCDs\_Visited = NCDs\_Visited + NCD_z$
(7) Req_Sent = 1
(8) **if** ((Req_Sent == 0) and (parent $\neq -1$))
(9) Rec_Reply(initiator, parent, -1)

Figure 6.3: The function of seeking reputation.

Pseudo-code for the Rec_Req function, which takes four parameters as follows.
Rec_Reply(initiator, NCD_z, $NCDs\_Visited$, NCD_t)
Where initiator is the NCD replying to NCD_z regarding the reputation of NCD_t

```
Rec_Reply(initiator, NCD_z, NCDs_Visited, NCD_t)
(1)  if ( DTT_{NCD_z}(NCD_t) ≠ −1 )
(2)      Rec_Reply( NCD_z, initiator, DTT_{NCD_z}(NCD_t) )
(3)  else
(4)      SeekRep(NCD_z, initiator, RTT_{NCD_z}, NCDs_Visited, NCD_t)
```

Figure 6.4: The function of recommendation request.

Figure 6.4 shows the pseudo-code for the recommendation request that initiated from Figure 6.3 line **(5)**. Once a recommender receives a recommendation request, there are two scenarios: (a) the recommender has previous interaction with NCD_t and hence the recommender will reply to the initiator of the recommendation request. This is outlined in lines **(1)** and **(2)** or (b) the recommender has no previous interaction with NCD_t and hence the recommender will resort to its recommenders for help and thus recursively contribute to the building of the recommendation tree. This is outlined in lines **(3)** and **(4)**. Line **(2)** states yet another definition of a leaf node, which is a recommender node in the recommendation tree that has previously interacted with NCD_t (i.e., $DTT_{NCD_z}(NCD_t) \neq -1$).

As, we can see from the above discussion that recommendation requests are pushed down the recommendation tree and replies to these recommendation requests are pushed up the recommendation tree. Obviously, there is a cost associated with: (a) sending recommendation requests and (b) sending replies to these recommendation requests. Let κ denote the cost involved in sending a recommendation request and χ denote the cost involved in sending a reply to a recommendation request. We will refer to recommendation requests and their replies as messages. Therefore, we are concerned with the number of messages

and their cost traversing down and up the recommendation tree. Let m denote the average number of messages per a behavior trust. Therefore, we can define the reputation cost (RC) as:

$$RC = \kappa m + \chi m. \tag{6.1}$$

6.3 The Scalability Metric

The scalability metric is based on *productivity*. If productivity is maintained as the scale changes, the system is considered to be scalable [92]. In our trust model, productivity is measured as the success rate divided by the cost. The cost is expressed in Equation 6.1 and the success rate is defined in Section 5.2.6. Given the following quantities:

- $C(k)$ = cost at scale k, expressed as the cost of a behavior trust in seconds.
- $f(k)$ = value delivered, expressed as the *success rate*.

Then, the productivity $F(k)$ is the value delivered per second, divided by the cost per second:

$$F(k) = f(k)/C(k) \tag{6.2}$$

The scalability metric relating systems at two different scale factors is then defined as the ratio of their productivity figures [92]:

$$\Psi(k_1, k_2) = F(k_2)/F(k_1) \tag{6.3}$$

This is the scalability metric used in the rest of this chapter. The trust model model is said to be scalable from configuration 1 to configuration 2 if productivity keeps pace with costs,

in which case Ψ will have a value greater than or not much less than unity. Based on [92], we use the threshold value of 0.8, and say the system is scalable from configuration 1 to configuration 2 if $\Psi > 0.8$.

Since the RC expressed in Equation 6.1 is the only behavior trust cost that needed to be considered, we have $C(k) = m(k)\ (\kappa + \chi)$, where $m(k)$ is the number of messages at scale k. Therefore, the scalability metric will be given by:

$$\begin{aligned} \Psi(k_1, k_2) &= F(k_2)/F(k_1) = \frac{f(k_2)}{C(k_2)} \times \frac{C(k_1)}{f(k_1)} \\ &= \frac{f(k_2)}{m(k_2)\ (\kappa + \chi)} \times \frac{m(k_1)\ (\kappa + \chi)}{f(k_1)} \\ &= \frac{f(k_2)}{m(k_2)} \times \frac{m(k_1)}{f(k_1)} \end{aligned} \tag{6.4}$$

We see from Equation 6.4 that the scalability metric depends on two things: (a) value delivered, expressed as the success rate. Simulation experiments were carried out in Section 5.2 to calculate the success rate and (b) the average number of messages (m), which are propagating down and up the recommendation tree. It turns out that m is a crucial parameter in calculating the scalability metric of our trust model. In the following section, m is investigated in more detail through simulation experiments.

6.4 Simulation

6.4.1 Goals of the simulation

We see from Equation 6.4 that the scalability metric depends on the average number of messages propagating down and up the recommendation tree (i.e., on the depth of the recommendation tree), which is in tern depends on the number of peering NCDs. In this section, the intention is to run simulation experiments to show that the way an NCD chooses

its target NCDs has an impact on m. Suppose that a source NCD (NCD_s) is interested to engage in a transaction with a target NCD (NCD_t). To determine the suitability of the candidate NCD_t, the NCD_s consults its set of recommenders to obtain NCD_t's reputation. Let NCD_r be one of the recommenders contacted by NCD_s. NCD_r will consult its DTT to find out whether it had prior transactions with NCD_t. If NCD_r had no prior transactions with NCD_t, then NCD_r will request its set of recommenders from its RTT to determine NCD_t's reputation and that is how messages are generated. If NCDs uniformly choose their target NCDs, it is more likely that NCD_s and all members of its recommenders set have interacted with NCD_t. That is, when NCD_r consults its DTT, it will find out that it had prior transactions with NCD_t. Hence, NCD_r will not contribute in forming a new message. On the other hand, if NCDs choose their target NCDs based on a normal distribution, it is more likely that NCD_s and all members of its recommenders set have different favored NCDs to interact with. That is, when NCD_r consults its DTT, it is likely to find out that it had no prior transactions with NCD_t. Hence, NCD_r will contribute in forming a new message.

In summary, the objective of the simulation experiments in this section is the following. We want to manipulate the selection phase in the simulation control flow, Figure 5.2. That is, we want to run the same simulation as in Section 5.2 (with minor changes, please see the following section) for two different cases: (a) NCD_s selects NCD_t randomly and (b) NCD_s selects NCD_t based on a normal distribution. Then compare the average number of messages for each case. Below, is the simulation experiments in detail.

6.4.2 Simulation Summary

We repeated the same simulation set up as in Section 5.2. That is, the simulation in this section is organized as in Section 5.2 except for the following (Please refer to Section 5.2 for further detail on the description of the simulation).

- $NCDs\_num$ is set deterministically at [30, 60, 90]. The NCDs are sorted in a circular array such that the neighbors of NCD_0 are NCD_1 and NCD_{29} and the neighbors of NCD_{29} are NCD_0 and NCD_{28}
- $alpha == 0.5$ meaning the NCD_s relies on its direct trust as well the reputation to determine the trustworthiness of NCD_t. We could also have set α to 0.0 as well. Since we are concerned to simulate and get the average depth of the recommendation tree, setting α to 1 is not suitable.
- $transactions\_num$ is set to 50.
- mon_freq is set deterministically at [1, 5, 10, 25, 50].
- $NCDs\_dishonest$ is set deterministically to $NCDs\_num$ / 2.
- $rels\_exist$ is set deterministically to $NCDs\_num \times 10/100$ and $NCDs\_num \times 20/100$.

6.4.3 Verification and Validation

In addition to the verification and validation techniques discussed in Section 5.2, we used the following validation technique to make sure that the target NCDs are selected according to the distribution chosen. Figures 6.5 and 6.6 are shown using 30 NCDs. As stated above and since the NCDs are sorted in a circular array such that the neighbors of NCD_0 are NCD_1 and NCD_{29} and the neighbors of NCD_{29} are NCD_0 and NCD_{28}, we plotted the average number of transactions that NCD_0 has with other NCDs. As expected, figure 6.5 shows a uniform distribution of target NCDs chosen by NCD_0. The figure shows that NCD_0 does not favor any NCD to interact with. Whereas Figure 6.6, shows a normal distribution of target NCDs chosen by NCD_0. Figure 6.6 shows that NCD_0 favors its neighbor NCDs. Both distributions in Figures 6.5 and 6.6 use a standard deviation of 5.

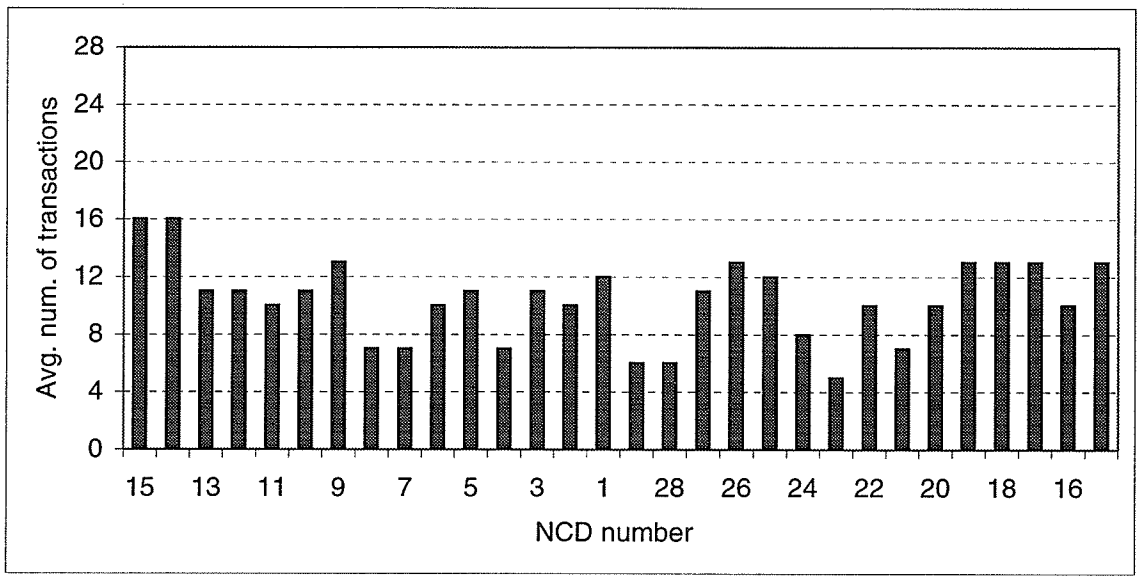


Figure 6.5: A network computing domain (NCD_0) selecting its target NCDs following a uniform distribution.

6.4.4 Simulation Results and Discussion

Simulation experiments were run and show that the average number of messages is smaller if source NCDs select their target NCDs following a uniform distribution versus a normal distribution. Table 6.2 illustrates how the average number of messages is affected by the distribution followed in selecting target NCDs. The 10% and 20% existing relations, mean that each NCD had prior direct interaction with 10% and 20% of the total number of NCDs, respectively. For more information on how these parameters such as $rels\_exist$ and $transactions\_num$ are implemented as mechanisms, please refer to Section 5.2. We also observe that selecting target NCDs based on a uniform distribution, always results in a smaller average number of messages per transaction.

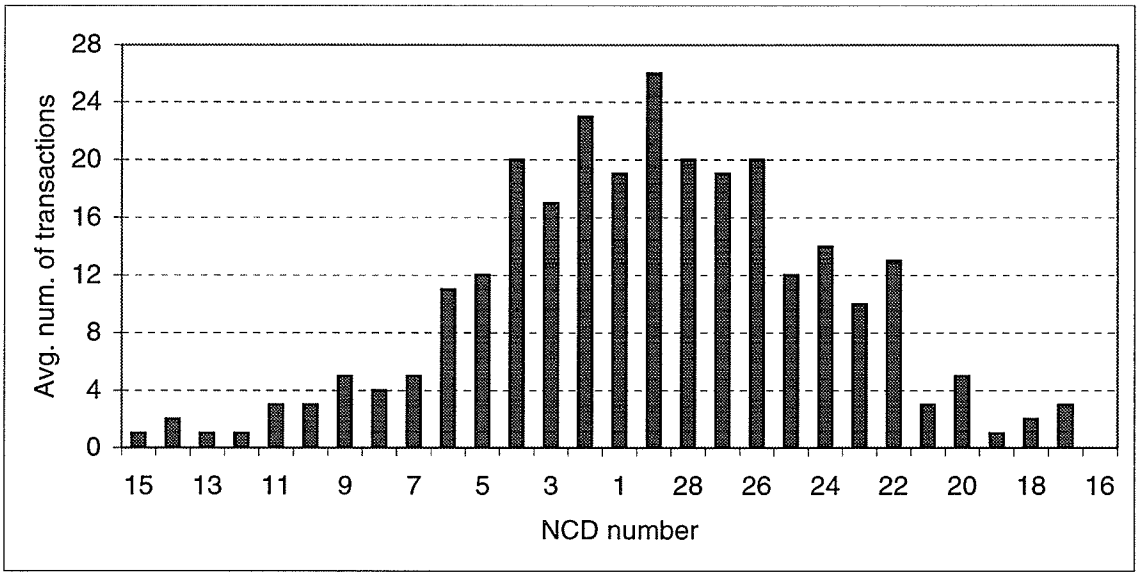


Figure 6.6: A network computing domain (NCD_0) selecting its target NCDs following a normal distribution.

6.5 Evaluation of the Scalability Metric

Simulation experiments were conducted, based on Section 5.2, to compute the success rate with different numbers of NCDs. As shown in Table 6.3, the number of NCDs are increased from 30 to 150 with the monitoring frequency done every 5 and 10 transactions. The success rates shown in Table 6.3 are obtained for 50 transactions per relation with $\alpha = 0.5$. The results show that as the number of NCDs are scaled up, our trust model still converges to an acceptable success rate.

The scalability is computed based on Equation 6.4 for 30, 60, and 90 NCDs, which corresponds to $k = 1$, $k = 2$, and $k = 3$, respectively. Table 6.4 shows the scalability results for dishonest NCDs equal half of the total number of NCDs, monitoring frequency $= 5$, $\alpha = 0.5$, and number of transactions per relations $= 50$. The results show that scaling

the trust model from scale 1 to 2 or from 1 to 3 yields a scalability value of more than 0.8. For example, using a uniform distribution and scaling the trust model from 1 to 2, we have the following: (a) at scale 1 (i.e., $NCDs\_num = 30$), the value delivered is 0.9782. That is $f(k_1) = 0.9782$. Also $m(k_1) = 4.3$ and (b) at scale 2 (i.e., $NCDs\_num = 60$), the value delivered is 0.9697. That is $f(k_2) = 0.9697$. Also $m(k_2) = 4.9$. Therefore, the scalability value, for this particular case, is computed using Equation 6.4 as:

$$\Psi(k_1, k_2) = \Psi(1, 2) = \frac{0.9697}{4.9} \times \frac{4.3}{0.9782} = 0.87 \tag{6.5}$$

The rest of Table 6.4 shows that as the trust model scales up, productivity keeps pace with cost. From the table, we can observe that when a normal distribution is followed in selecting target NCDs, the scalability decreases beyond the 0.8 threshold. The reason for this is as follows. Because of using a normal distribution to select target NCDs, a source NCD will interact most of the time with favored group of NCDs. Since the success rate between two NCDs improves as they interact, the success rate should have been weighted by the number of transactions between the source NCD and its target NCDs. Hence, the success rate will have more weight between the source NCD and each of its favored target NCDs. However, the success rate shown in Table 6.4 is computed as an equally-weighted average success rate between a source NCD and each of its target NCDs.

Table 6.2: Comparison of average number of messages for various number of NCDs.

Num. of NCDs	Transactions per relation	Selection distribution of target NCD	Avg. num. of messages per transaction	
			10% existing relations	20% existing relations
30	1	uniform	18.8	14.8
		normal	20.4	14
	5	uniform	7.6	6.4
		normal	10	8
	10	uniform	6	5.2
		normal	7.6	6.4
	25	uniform	4.8	4.3
		normal	5.2	5
	50	uniform	4.3	4.1
		normal	4.6	4.2
60	1	uniform	27.2	19.2
		normal	30	20.8
	5	uniform	9.2	7.6
		normal	11.6	9.6
	10	uniform	6.8	5.6
		normal	8.4	7.2
	25	uniform	5.2	4.8
		normal	6.4	6
	50	uniform	4.9	4.4
		normal	5.7	4.7
90	1	uniform	32.8	25.2
		normal	35.6	26.8
	5	uniform	10.4	8.8
		normal	13.2	11.2
	10	uniform	7.6	6.4
		normal	9.2	8
	25	uniform	5.6	5.2
		normal	7.2	6.4
	50	uniform	5.1	4.8
		normal	6.4	5.3

Table 6.3: Success rate for different number of NCDs where half of the NCDs are dishonest, $\alpha = 0.5$, and number of transactions per relation $= 50$.

Selection distribution of target NCD	Monitoring frequency	Number of NCDs	Success rate
uniform	5	30	97.82%
		60	96.97%
		90	97.01%
		150	95.79%
	10	30	90.11%
		60	88.40%
		90	89.77%
		150	90.01%

Table 6.4: Trust model scalability with various number of NCDs where half of the NCDs are dishonest, $\alpha = 0.5$, monitoring frequency $= 5$, and number of transactions per relation $= 50$.

Selection distribution of target NCDs	Num of NCDs	Avg. num. of messages per transaction	Success rate	Scaling from x NCDs to y NCDs	Scalability
uniform	30	4.3	97.82%	$x = 30, y = 60$	0.87
	60	4.9	96.97%		
	30	4.3	97.82%	$x = 30, y = 90$	0.84
	90	5.1	97.01%		
normal	30	4.6	94.21%	$x = 30, y = 60$	0.79
	60	5.7	91.83%		
	30	4.6	94.21%	$x = 30, y = 90$	0.68
	90	6.4	89.07%		

6.6 Intra-NCD Costs

As mentioned in Chapter 4, nodes are aggregated into NCDs and this aggregation process elects a leader who manages the member nodes to maintain a high reputation for its NCD within the global community. Suppose that a node wants to join NCD_s. Joining NCD_s, a node negotiates with NCD_s's leader who can either reject or accept the join request. A node's join request can be supported by references from prior associations. References are given by NCDs and hence can affect the referee NCD's reputation if the node's behavior is not up to the reference. As a consequence, the referee NCD might be isolated and no more references are accepted from it. This also has a consequence on the referee NCD to be isolated from NCD_s's recommendation set and consequently from other NCDs' recommendation sets. Upon joining, nodes are clustered into 5 clusters in accordance with the 5 trust levels. If the node has no references, it will be placed into cluster 1. Based on its references, a node will be placed in an appropriate cluster matching the node's trustworthiness.. The leader is responsible for promoting or demoting nodes among the 5 clusters based on the nodes' behavior. The leader maintains an *internal trust table* (ITT) that includes trust levels for the different nodes. Each time a node participates in a transaction, the leader adjusts the node's trust level in the ITT by: (a) monitoring the transaction and/or (b) getting reviews from NCDs which used the node.

From the time a node wants to join an NCD to the time it leaves. the following internal costs are involved.

1. Cost due to the joining process. This cost includes network cost as well as processing cost.

    - The network cost is measured in bytes and is represented by a join request message cost sent by the node to the leader, a reply message cost sent from the leader to the node. If the node was accepted to join, a join message cost is sent

from the node to the leader.

- The processing cost is measured in seconds and it includes accessing (reading) each of the messages at each end (the node end and the leader end), the leader accessing its DTT and/or RTT to know the referees' trustworthiness. If a referee is unknown to the NCD, the NCD can probe its recommenders for the referee reputation.

2. Cost due to the leaving process. This is a processing cost measured in seconds and it includes the leader accessing its ITT and the node's cluster to remove the node.

3. Cost due to managing the node while it is with the NCD. This cost has two components, namely processing cost and network cost.

   - The processing cost includes a monitoring cost during transactions in which the node participates, computing the node's trust level, accessing ITT to adjust the nodes trust level, and promoting or demoting the node among the clusters.
   - The network cost involves sending review request messages to these nodes which recently used the node and receiving reply review messages. These reviews are used to compute the node's trust level.

Besides the scalability concerns at the node-level, there are factors related to the context that needs to be further investigated and examined as part of the scalability study of our trust model. For example, we limited the context to primary service types such as printing, storage, and computation. This reduces the fragmentation of the trust management space, but the coarse definition of context can result in inaccurate trust level estimates. Depending on the definition of context, a huge amount of data may result. On the other hand, a coarse definition of context may result in inaccurate results. In conclusion, more investigation needs to be done on the scalability at the context level as well as at the context level.

6.7 Summary

In this chapter, we examined the scalability issue of the trust model at the NCD-level. A scalability metric based on *productivity* is used. Productivity is measured as the value delivered divided by the cost. In our trust model, the value delivered and the cost are derived by examining a behavior trust between two NCDs, namely NCD_s and NCD_t. From an NCD_s's perspective, the value delivered is measured by how successful is a transaction with NCD_t; whereas the cost is measured by NCD_s estimating the behavior trust of NCD_t prior to engaging in a transaction with NCD_t. The system is considered to be scalable from one scale to another, if the value delivered keeps pace with the cost. The scalability study shows that if scaled up from scale 1 to scale 3, then the trust model is scalable.

The scalability study examined in this chapter is done at the NCD-level. This study lays the first step towards understanding the overall scalability of the trust model. When considering cost, additional cost incurs at the node-level where an NCD manages its nodes (clients or resources) and deploys mechanisms to allow nodes to join, operate, and leave the pool of clients or resources. This management needs to be done in a scalable and efficient manner because the behavior of the nodes affect the NCD's reputation and hence its interaction with other NCDs. So, the goal is to generalize the scalability metric such that the overall cost will include the behavior trust cost, cost at the node-level, and cost related to the context.

Chapter 7

Applications of the Trust Model

7.1 Overview

The Grid [60, 81, 41, 48] is a highly scalable network computing system that is emerging as a popular mechanism for harnessing resources that are distributed on a wide-area network. Conceptually, a Grid computing system allows resources from different administrative domains to participate in it and ensures the autonomy of the different sites (referred to hereafter as domains). However, in current practice, Grid computing systems are built from resources that are contributed by institutions that agree to work together due to off-line trust relationships that exist among them [26]. To scale a Grid beyond hundreds of nodes, it is necessary to accommodate public resources, where a *priori* trust relationships do not exist among the resources [26].

A variety of different approaches can be used to construct Grid systems that fit into this class including combining elements of technology from P2P and Grid computing. Recently [93, 94, 26], there has been an interest in a new class of Grids called Peer-to-Peer Grids (P2P Grids). P2P Grid is a new trend in scientific computing and collaboration with a set of services that includes those typical of Grids and P2P networks [95]. For our purpose, we

define P2P Grids as Grids for which the different domains do not have off-line agreements that maintain static trust relationships among them. We contend that one of the most desirable features of P2P Grids is opening up the membership of the Grid much like P2P file sharing systems. This provides an opportunity for the Grid to increase its eligible number of participants.

Currently, only best-effort application specific Grids such as SETI@home bring public resources under a single virtual entity. One way to increase the applicability of such systems is to make them QoS-aware. To provide services with QoS, the resources should be managed. Because a P2P Grid is made of resources with heterogeneous trust relationships [26, 29, 30], the resource manager needs to consider these trust relationships while managing the resources. Currently, P2P Grid resource management systems (RMSs) make allocation decisions oblivious of the trust implications [61, 36, 47]. That is, the trust relationships between resources and tasks are not integrated into the allocation decision.

P2P Grid Systems have RMSs to govern the execution of tasks that arrive for service. The tasks are assumed to be independent, which is a realistic assumption. For example, independent clients submit their tasks to a collection of shared resources in a P2P Grid environment. Therefore, algorithms are necessary to assign tasks to machines and compute the execution order of the tasks assigned to one machine. The process of assigning tasks to machines and computing the execution order of the tasks assigned to one machine is called *scheduling* or *mapping* [96]. Literature on mapping independent tasks onto a heterogeneous computing environment such as a P2P Grid includes a well-known NP-complete problem with throughput as the optimization criterion [96, 97]. Researchers such as [96] have investigated mapping heuristics based the following assumptions: (a) tasks are mapped non-preemptively, (b) mappers are organized centrally, (c) tasks are assumed to have no deadlines or priorities associated with them, and (d) tasks are indivisible (i.e., a task cannot be distributed over multiple machines).

The mapping heuristics can be grouped into two categories, namely immediate mode

and batch mode heuristics [96]. Immediate mode means that a tasks is mapped or scheduled as soon as it arrives for service at the mapper or scheduler, whereas batch mode means that tasks are collected into a set that is examined for mapping at pre-scheduled times called *mapping events*. The trade-offs among and between immediate mode and batch mode heuristics are studied in detail in [96]. The independent set of tasks that are considered for mapping at the mapping events is called a *meta-task*. A meta task can include newly arrived tasks (i.e., the ones arriving after the last mapping event) and the ones that were mapped earlier but did not begin execution. While immediate mode heuristics consider a task for mapping only once, batch mode heuristics consider a task for mapping at each mapping event until the task begins execution.

The primary objective of these resource management algorithms (i.e., mapping heuristics) is *makespan* minimization, where makespan is defined as the time required to complete all tasks. The mapping heuristics use makespan minimization as their mapping criterion. It should be noted that makespan is related to the throughput of a heterogeneous systems such as a P2P Grid. Makespan does not consider quality of service (QoS) nor trust when assigning tasks to resources. In this study, we examine the integration of the notion of "trust" into resource management such that the allocation process is aware of the trust implications. To the best of our knowledge, no existing literature directly addresses the issues of trust-aware resource management [36].

The rest of this chapter is organized as follows. A trust model for P2P Grid systems is outlined in Section 7.2. Notation and terminology used in the rest of this chapter is discussed in Section 7.3. Trust-aware resource management algorithms are discussed in Sections 7.4 and 7.7. Evaluation of security overheads and the analysis of trust-aware schemes are presented in Sections 7.1.1 and 7.5, respectively. Finally, the performance of the proposed trust-aware resource management algorithms are examined in Section 7.8.

7.1.1 Evaluation of Security Overheads

We claimed that there is security overhead due to applying security mechanisms to address the security concerns from both clients and resources. We conducted a study to examine the overhead of securing data transmissions for 100 Mbps and 1000 Mbps networks. The machines used were base on an Intel Pentium III processor running at 866 MHz with memory size of 256 MB and a level 2 cache of size 256 KB. Tables 7.1 and 7.2 show the security overhead for secure transmissions using *secure copy* (scp) versus the regular transmission using *remote copy* (rcp) for different network speeds and with different file sizes. As illustrated in Tables 7.1 through 7.2, using `scp` introduces an overhead caused by the addition of security to the file transfer.

Table 7.1: Secure versus regular transmission for a 100 Mbps network.

File size/MB	Regular transmission using rcp/(sec)	Secure transmission using scp/(sec)	Overhead
1	0.19	0.63	69.84%
10	1.37	2.45	44.08%
100	9.77	15.34	36.31%
500	48.88	77.56	36.70%
1000	97.00	155.07	37.45%

From Table 7.2, we observe that the security overhead negates the benefits of using the high speed network. Also, the security overhead as shown in Table 7.2 is significant for the secure transmission when compared to the regular transmission using `rcp`.

Furthermore, a performance study was done in [98] where three target benchmark applications are processed by *Minimal i386 Software Fault Isolation Tool* (MiSFIT) [99] and *Security Automata SFI Implementation* (SASI x86SFI) [98] sandboxing systems. *Software*

Table 7.2: Secure versus regular transmission for a 1000 Mbps network.

File size/MB	Regular transmission using rcp/(sec)	Secure transmission using scp/(sec)	Overhead
1	0.34	0.65	47.69%
10	0.50	2.18	77.06%
100	4.98	14.23	65.00%
500	22.44	69.86	67.88%
1000	46.05	138.30	66.70%

fault isolation (SFI) is a sandboxing technique for transforming code written in unsafe language into safe compiled code. `MiSFIT` specializes the `SFI` technique to transform C++ code into safe binary code whereas `SASI x86SFI` specializes `SFI` to transform x86 assembly language output of the GNU gcc C compiler to safe binary code. The three target applications used are: (a) a memory intensive application benchmark called *page-eviction hotlist*, (b) *logical log-structured disk*, and (c) a command line message digest utility called `MD5`.

Page-eviction hotlist has the highest runtime overhead of 137% on `MiSFIT` and 264% on `SASI x86SFI` compared to the execution of the target applications on the target systems with no sandboxing. The other two benchmark applications performed as follows (compared to their execution on the target systems with no sandboxing): the *logical log-structured disk* has runtime overhead of 58% on `MiSFIT` and 65% on `SASI x86SFI`, whereas `MD5` has runtime overhead of 33% on `MiSFIT` and 36% on `SASI x86SFI`.

The additional overhead caused by techniques such as sandboxing may negate the performance advantages gained by the Grid computing and hence we contend that it is essential for the scheduler to consider the security implications while performing resource allocations.

7.2 A Trust Model for Peer-to-Peer Grids

A P2P Grid is composed of several domains (also referred to as NCDs) that are made up of resources and clients. These NCDs are considered interconnected domains that interact in a P2P fashion to share resources and services amongst themselves. The trust model is deployed on each NCD and the trust model elements such as DTT_{NCD}, RTT_{NCD}, TM_{NCD} proxy, TA_{NCD}, and T_{NCD} are designed to operate and evolve trust in a purely distributed manner. There is no NCD that is omniscient. Rather each NCD: (a) has its own view of the trustworthiness of other NCDs and stores this information in DTT_{NCD}, (b) controls the monitoring process of its own transactions using a TM_{NCD} proxy, (c) has its own set of recommenders and set of trusted allies, and (d) maintains DTT_{NCD} and RTT_{NCD} using its own TA_{NCD}. Each NCD cooperates with its peer NCDs by sharing information in the form of recommendations.

From a trust-aware resource management system's perspective, a resource of an NCD has the following attributes: (a) *type of contexts* (ToCs) it supports and (b) a trust level for each ToC. Because an NCD is responsible for representing the resources in the trust model, the NCD and the resource negotiate on the ToCs and the corresponding trust levels that can be advertised for a resource. The ToCs supported by an NCD are determined by the functionalities of the resources that are part of an NCD (for example, an NCD could support printing, storing data, and display services as ToCs). Associating a trust level with each ToC provides the flexibility of selectively opening services to clients. Similarly, a client of an NCD has its own trust attributes including: (a) ToCs sought and (b) trust levels associated with the ToCs.

Table 7.3 shows an example DTT for a set of NCDs where TL_{ij}^{k} is the *offered trust level* (OTL) that NCD_i offers to NCD_j to engage in activity within context k. Suppose we have client X from NCD_j wanting to engage in activities within contexts c_1, c_2, and c_3 on resource Y in NCD_i. Because resources and clients inherit the trust levels from

the NCD they are associated with, we can compute the OTL for the composite activity between X and Y, i.e., OTL $= \min(TL_{ij}^{c_1}, TL_{ij}^{c_2}, TL_{ij}^{c_3})$. There is a *required trust level* (RTL) from the client side specifying the trust level required by NCD_j's client (i.e., X) to use NCD_i's resource (i.e., Y). Also, there is a RTL from the resource side specifying the trust level required by NCD_i to use its resources. If the OTL is greater than on equal to the maximum of client and resource RTLs, then the transaction can proceed with no additional security overhead. Otherwise, there will be a security overhead. Table 7.4 shows the *trust supplement* (TS) table for different RTL and OTL values. The $TS(j, m)$ values are given by RTL $-$ OTL. When RTL $\leq$ OTL, $TS(j, m)$ is zero and when RTL $= 6$, $TS(j, m)$ is 6. RTL can have a value 6 that is not provided by any OTL. This is supported in our trust model so that clients or resources can enforce enhanced security by increasing their RTL value to 6.

Table 7.3: An example of a direct trust table between NCDs.

Network Computing Domains	Type of contexts	Network Computing Domains				
		NCD_1		...	NCD_j	
		Trust Level	Timestamp	...	Trust Level	Timestamp
NCD_1	c_1	$TL_{11}^{c_1}$	$T_{11}^{c_1}$	...	$TL_{1j}^{c_1}$	$T_{1j}^{c_1}$
	$\vdots$	$\vdots$	$\vdots$	...	$\vdots$	$\vdots$
	c_k	$TL_{11}^{c_k}$	$T_{11}^{c_k}$	...	$TL_{1j}^{c_k}$	$T_{1j}^{c_k}$
$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
NCD_i	c_1	$TL_{i1}^{c_1}$	$T_{i1}^{c_1}$	...	$TL_{ij}^{c_1}$	$T_{ij}^{c_1}$
	$\vdots$	$\vdots$	$\vdots$	...	$\vdots$	$\vdots$
	c_k	$TL_{i1}^{c_k}$	$T_{i1}^{c_k}$	...	$TL_{ij}^{c_k}$	$T_{ij}^{c_k}$

Table 7.4: the trust supplement table.

Required trust level (RTL)	Offered trust level (OTL)				
	1	2	3	4	5
1	0	0	0	0	0
2	1	0	0	0	0
3	2	1	0	0	0
4	3	2	1	0	0
5	4	3	2	1	0
6	6	6	6	6	6

7.3 Notation and Terminology

Let $ET(j, m)$ be the *execution time* for task j on machine m. In an ET matrix, the numbers along a row indicate the execution time of the corresponding task on different machines. Variation along the rows is referred as *machine heterogeneity* of a task. Similarly, the numbers along a column of the ET matrix indicate the execution time of the machine for different tasks on one machine. Variation along columns is referred to as *task heterogeneity*. An ET can be classified into two classes, namely consistent and inconsistent. An ET is said to be consistent if whenever machine m has a lower execution time than machine q for task j, the same is true for any task k. If an ET matrix is not consistent, then it is referred to as inconsistent.

Definition: A matrix of execution is said to be consistent if:

$$ET(j, m) > ET(j, q) \Rightarrow ET(k, m) > ET(k, q) \ \forall \ j, k \in T, m, q \in M \qquad (7.1)$$

where M is set of machines and T is set of tasks. Also, Let $C(j, m)$ denote the *Completion time* of task j on machine m and let α_m denote the completion time of the last task assigned

to machine m so far.

7.4 Resource Management Based on Security Overhead Minimization

7.4.1 Overview

In previous generation network computing systems, RMSs were primarily responsible for allocating resources for tasks. They also performed functions such as resource discovery and monitoring that supported their primary role. In P2P Grid systems, with distributed ownership for the resources and tasks, it is important to consider *quality of service* (QoS) and security while allocating resources. Integration of QoS into RMS has been examined by several researchers [100, 101]. However, security is implemented as a separate subsystem of the P2P Grid [47] and the RMS makes the allocation decisions oblivious of the security implications.

Integrating trust into resource management algorithms is motivated by the following scenarios. P2P Grid computing systems provide a facility that enables large-scale controlled sharing and inter-operation among resources that are distributively owned and managed. Trust is a major concern for the consumers and producers of services that participate in a P2P Grid. Some resource consumers may not want their applications mapped onto resources that are owned and/or managed by entities they do not trust. Similar concerns apply from the resource producer side as well. The current generation of distributed systems addresses these concerns by providing security at different levels. Suppose resource M is allocated to task T. Resource M can employ sandboxing techniques to prevent task T from eavesdropping or interfering with other computation or activities ongoing on M. Similarly, task T may employ encryption, data hiding, intelligent data encoding, or other mechanisms to prevent M from snooping into the sensitive information carried by task T.

Security concerns, such as the ones mentioned above, have hindered the acceptance and wide-spread use of P2P Grid applications [102, 103].

Based on the above scenarios, we hypothesize that if the RMS is aware of the security requirements of the resources and tasks, it can perform the allocations such that the security overhead is minimized [63, 36]. This is the goal of the *trust-aware resource management system* (TRMS) studied and examined here. The TRMS achieves this goal by allocating resources considering a trust relationship between the resources and the clients. If an RMS maps a resource request strictly according to the trust, then there can be a severe load imbalance in a large-scale wide area system such as the P2P Grid. On the other hand, considering just the load balance or resource-task affinities, as in existing RMSs, causes inefficient overall operation due to the introduction of the overhead caused by enforcing the required level of security. Mapping according to load balance or trust considerations results in diverging schedules. The former spreads the requests for the sake of load balance while the latter segregates them for security considerations. In the TRMS algorithms examined here, the minimization criterion is derived from load balancing and security considerations. The load balancing is accomplished by the makespan minimization criterion. Let $ST(j, m)$ be the processing time associated with the TS (Table 7.4) if task j is assigned to machine m. To consider security while allocating tasks to resources, $ST(j, m)$ should be included when computing the total *processing time* $PT(j, m)$ of task j on machine m. This is shown in Equation 7.2.

$$PT(j, m) = ET(j, m) + ST(j, m) \tag{7.2}$$

In our simulations, the $ST(j, m)$ are assigned values as a percentage of the $ET(j, m)$. When trust is not considered, $ST(j, m)$ values are set to 50% of $ET(j, m)$ as shown in Equation 7.3. On the other hand, when trust is considered, $ST(j, m)$ values are computed by multiplying $ET(j, m)$ values by a weighted TS value (Table 7.4). We arbitrarily choose

the weight for TS as 15/100 as shown in Equation 7.4. Therefore, when trust is considered, on average the $ST(j,m)$ values are calculated as 45% of the $ET(j,m)$ [36].

$$ST(j,m) = ET(j,m) \times 50/100 \quad (7.3)$$

$$ST(j,m) = ET(j,m) \times TS(j,m) \times 15/100 \quad (7.4)$$

The sole purpose of the *trust-aware resource management* (TRM) algorithms is to demonstrate the utility of the trust model. We present three TRM algorithms as example applications of integrating trust into the RMS where clients belonging to different NCDs present the tasks for task executions and the TRM algorithms allocate the resources. Different tasks belonging to the same NCD may be mapped onto different NCDs.

The three TRM algorithms implemented are based on the following three trust-unaware heuristics based on [96]: *minimum completion time* (MCT) heuristic, *min-min* heuristic, and *sufferage* heuristic. For further details on these three mapping heuristics, please refer to [96]. We will refer to these three heuristics as: trust-unaware *minimum completion time* (MCT) heuristic, trust-unaware *min-min* heuristic, and trust-unaware *sufferage* heuristic.

The three TRM algorithms implemented are: (a) trust-aware *minimum completion time* (MCT) heuristic, (b) trust-aware *min-min* heuristic, and (c) trust-aware *sufferage* heuristic. The MCT is an on-line or immediate mode mapping heuristic, whereas the min-min and sufferage are batch mode mapping heuristics.

For the trust-unaware algorithms, the idea is to map a task j to a machine m that gives the earliest completion time without considering the security overhead. Although the completion time was calculated in terms of $PT(j,m)$ (Equation 7.2) and α_m, $ST(j,m)$ is not considered when mapping j to m. For the trust-aware algorithms, $ST(j,m)$ is considered when mapping as well as calculating $PT(j,m)$.

For the on-line mode mapping heuristic, the TRM schedules client tasks as they arrive. For the batch mode mapping heuristics, the TRM collects client tasks for a predefined time interval to form batch of tasks (i.e., a meta-task).The meta-task is then scheduled by the `TRM-scheduler` function, which is called when the current time is equal to the `current scheduling event time` that is equal to τ. The TRM-scheduler function, schedules the *meta-task* based on the two batch mode mapping heuristics, namely min-min and sufferage.

7.4.2 Trust-aware Minimum Completion Time Algorithm

The MCT heuristic [96] is an immediate mode mapping heuristic. The MCT heuristic, shown in Figure 7.1, assigns each task to a machine as soon as the task arrives for service. The task is assigned to a machine that results in that task's earliest completion time. This causes some tasks to be assigned to machines that do not have the minimum processing time for them. As a task arrives, all the machines are examined to determine the machine that gives the earliest completion time for the task. The completion time is computed as shown in line **(2)**. Then α_m, representing the completion time of the last task assigned to machine m so far, is updated to reflect the available time of machine m. Note that trust is integrated into the allocation decision as shown in line **(2)**. When calculating the completion time of task j on machine m (i.e., $CT(j,m)$), the $ST(j,m)$ is taken into consideration.

7.4.3 Trust-Aware Min-min Algorithm

The `TRM-scheduler` algorithm schedules a batch of tasks called *meta-task*. To map the *meta-tasks*, a min-min heuristic [96] is used as shown in Figure 7.2. First, the $CT(j,m)$ entries are computed using the $ET(j,m)$, $ST(j,m)$, and α_m. For each task j, the machine m^* that gives the earliest completion time is determined by scanning th i^{th} row of the CT matrix. The task j^* that has the smallest processing time is determined and then assigned

(1) **for** all machines m **do**
(2) $CT(j,m) = ET(j,m) + ST(j,m) + \alpha_m = PT(j,m) + \alpha_m$
(3) Assign task j to the machine m^* that gives the earliest completion time.
(4) $\alpha_{m^*} = \alpha_{m^*} + ET(j,m^*) + ST(j,m^*) = \alpha_{m^*} + PT(j,m^*)$

Figure 7.1: RMS scheduling algorithm using the minimum completion time heuristic.

to m^*. The matrix CT and α_m are updated and the above process is repeated with tasks that have not yet been assigned a machine.

function RMS-scheduler(meta-task R_v, τ_n)
(1) **for** all tasks j in meta-task R_v **do**
(2) **for** all machines m **do**
(3) $CT(j,m) = ET(j,m) + ST(j,m) + \alpha_m = PT(j,m) + \alpha_m$
(4) **do until (** all tasks in R_v are scheduled
OR the minimum machine completion time $> \tau_n$ **)**
(5) **for** each task j in R_v find the earliest completion time
and the machine (m^*) that obtains it.
(6) Find the task j^* with the minimum processing time.
(7) Assign j^* to the machine m^* that gives the minimum earliest completion time.
(8) Delete task j^* from R_v
(9) $\alpha_{m^*} = \alpha_{m^*} + ET(j^*,m^*) + ST(j^*,m^*)$
(10) Update $CT(j,m)$ for all j
(11) **enddo**

Figure 7.2: RMS scheduling algorithm using the min-min heuristic.

7.4.4 Trust-Aware Sufferage Algorithm

The `TRM-scheduler` algorithm schedules a batch of tasks called *meta-task* based on [96] called the Sufferage heuristic. The Sufferage heuristic is based on the idea that better

mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it. Let the *sufferage value* of task j be the difference between its second earliest completion time (on some machine m) and its earliest completion time (on some machine q). That is, using m will result in the best completion time for j, and using q will result in the second best completion time for j.

As shown in Figure 7.3, the initialization of the trust-aware Sufferage algorithm is similar to the min-min heuristic. However, for each iteration of the **for** loop in Lines **(6)** to **(14)**, the algorithm picks an arbitrary task j from the *meta-task* R_v and assigns it to a machine m that gives the earliest completion time for task j. Then, tentatively assigns m to j if m is unassigned, marks m as assigned, and removes j from R_v. If however there was another task k that was assigned to machine m previously, the algorithm chooses the task (among j and k) that suffers the most (i.e., has a higher sufferage value) if not assigned to machine m. It should be noted that the unchosen task (among j and k) will not be considered again for execution until the next iteration of the **do** loop on line **(4)**.

7.5 Analysis of the Trust-Aware Schemes

The goal of the three mapping heuristics (MCT, Min-min, and Sufferage) is to minimize the makespan, where makespan is defined as the maximum among the available times of all machines after they complete the tasks assigned to them. Initially $\alpha_m = 0, \forall\, m$.

Theorem: The makespan obtained by an optimal trust-aware scheduler, which minimizes makespan, is always less than or equal to the makespan obtained by any trust-unaware scheduler.

Proof: Let X^T_{km} be the mapping function computed by the optimal trust-aware scheduler, where $X^T_{km} = 1$, if task j is assigned to machine m and 0, otherwise. Let X^{UT}_{km} be the mapping function computed by the trust-unaware scheduler, where $X^{UT}_{km} = 1$, if task j

function RMS-scheduler(meta-task R_v, τ_n)
(1) **for** all tasks j in meta-task R_v **do**
(2) **for** all machines m **do**
(3) $CT(j,m) = ET(j,m) + ST(j,m) + \alpha_m = PT(j,m) + \alpha_m$
(4) **do until (** all tasks in R_v are scheduled
OR the minimum machine completion time $> \tau_n$ **)**
(5) Mark all machines unassigned
(6) **for** each task j in R_v find the earliest completion time
and the machine m that obtains it.
(7) sufferage value = second earliest completion time - earliest completion time
(8) **if** m is unassigned
(9) assign j to m, mark m assigned, and delete j from R_v
(10) **else**
(11) **if** sufferage value of k already assigned
to m is less than the sufferage value of j
(12) unassign j, add k back to R_v
(13) assign j to m
(14) delete j from R_v
(15) Update $\alpha_{m^*} = \alpha_{m^*} + ET(j^*,m^*) + ST(j^*,m^*)$
(16) Update $CT(j,m)$ for all j

Figure 7.3: RMS scheduling algorithm using the sufferage heuristic.

is assigned to machine m and 0, otherwise. Let the makespan obtained by the optimal trust-aware scheduler be:

$$\Lambda_T = \max_m \{\sum_{j=0}^{n-1} [ET(j,m) + ST(j,m)] \times X_{jm}^T\}$$

Let the makespan obtained by the trust-unaware scheduler be:

$$\Lambda_{UT} = \max_m \{\sum_{j=0}^{n-1} [ET(j,m) + ST(j,m)] \times X_{jm}^{UT}\}$$

By virtue of the optimality of X^T,

$$\Lambda_T \leq \max_m \{\sum_{j=0}^{n-1} [ET(j,m) + ST(j,m)] \times X_{jm}\}$$

where X_{jm} is any mapping function. Therefore,

$$\Lambda_T \leq \max_m \{\sum_{j=0}^{n-1} [ET(j,m) + ST(j,m)] \times X_{jm}^{UT}\} = \Lambda_{UT}$$

since X_{jm}^{UT} is a mapping function.

7.6 Practical Issues

So far, we have proposed integrating the "trust" notion into RMSs to reduce the security overhead while allocating resources to tasks. We accomplished this by assuming that the security overhead can be quantified and tied to the trust supplement values in Table 7.4. From a practical point of view, this assumption is not realistic. For example, if the trust supplement (TS) is 3, then the processing time due to the security supplement is calculated as indicated in Equation 7.4:

$$ST(j,m) = ET(j,m) \times 45/100 \tag{7.5}$$

Therefore, we assume that using a security mechanism will patch and fill the security gap of 45% of $ET(j,m)$. That is, we assume that there exists a security solution (e.g., SSH, sandboxing, etc.) to exactly fill a gap of size x. In other words, we quantify the chosen security solution to supplement the gap and bring up the level of security to what the client or resource requires. In practice, quantifying the level of security SSH or sandboxing gives to a particular transaction is very difficult to come up with.

In Section 7.7, we enhance the idea of integrating the "trust" notion into RMSs such that the above problem is alleviated. We integrate trust into RMSs such that it tracks the trust relationships among the different NCDs and brings together only those NCDs that have high levels of trust among them in any given allocation. Allocating tasks on untrustworthy resources is risky and a *risk overhead* and *risk penalty* are associated . We do not try to quantify or supplement the risk involved. Rather, we minimize the risk penalty.

7.7 Resource Management Based on Risk Minimization

7.7.1 Overview

In this section, we show how an algorithm based on a simple resource management heuristic can be modified to perform trust aware resource management. An untrustworthy NCD can promise resources or services and fail to deliver them and it can go down during peak hours. One way of holding a resource accountable is to maintain a trust parameter dedicated to its NCD and update it accordingly. The *risk overhead* that the resource manager might incur represents re-allocating the task to another NCD and re-scheduling it. One of the advantages of avoiding resources from untrustworthy NCDs in a single *virtual collection* is the potential of reducing the *risk overhead* associated with sustaining the transactions. As a result, the trust-aware allocation enables the resource manager to provide a higher level of assurance on the delivered performance.

The objective of this exercise is to show the utility of the trust model and not to solve the resource management problem optimally. In this study, we choose the *min-min* heuristic [96] as the base algorithm. To recap, the *min-min* heuristic has two phases. In the first phase; for each task j, the machine that gives the earliest completion time is determined by scanning the i^{th} row of the CT table. In the second phase; the task k that has the minimum completion time is determined and then assigned to the machine chosen in the

first phase. The goal of the *min-min* heuristic is to assign a set of tasks $\{0 \ldots n-1\}$ such that $\{\max_m\{\alpha_m\}\}$ is minimized for all m, where n is the number of tasks and m is the number of machines.

In this section, we refer to $TS(j,m)$ as the *risk factor* (known earlier as "trust supplement" in Section 7.2) when assigning j to m and is computed as in Table 7.4.

To perform resource management, we can apply the *min-min* algorithm such that it makes allocation decisions based only on execution times. That is, allocation decisions are made without considering any risk. This plain *min-min* algorithm, referred to as *trust unaware min-min* algorithm, is concerned with one parameter (i.e., execution time) when allocating tasks onto machines. On the other hand, we introduce two variations of the *min-min* algorithm such that completion cost as well as risk are considered when allocating tasks onto machines. First, is the *trust-aware trade-off* algorithm, which makes the allocation decisions based on a combination of completion time and risk and is flexible in weighing the two components differently. The objective of *trust aware trade-off* algorithm is to make a trade-off between the two components and choose a solution that gives significant preferences gain. This objective is called a "bi" objective formulation. The *min-min* algorithm has a "uni" objective formulation since it only considers completion times. Second, is the *trust aware maximum risk* algorithm , where we set an upper bound (B_{max}) on $TS(j,m)$ for any individual allocation that the resource management is willing to take. The difference between $TS(j,m)$ and B_{max} is computed and a large penalty factor is added to machine m's completion time if the difference is ≥ 0. These two trust-aware algorithms are discussed in detail next.

7.7.2 Trust Aware Trade-off Algorithm

Since there are n tasks, each task can be allocated to k different machines. For each of these allocations, we associate $CT(j,m)$ and $TS(j,m)$. For each j, the trade-off algorithm finds

the maximum $CT(j,m)$ and the maximum $TS(j,m)$. Using these two maximum numbers, the trade-off algorithm normalizes the i^{th} row of the CT and the *risk factor* tables, respectively. The *risk factor* table is the same as the *supplement time* table and is shown in 7.4. Hence, we end up with normalized CT and *risk factor* tables. Because the completion time changes with each assignment, the completion times need to be recomputed and renormalized with every assignment.

Let w_c and w_r represent weights for the two components, namely completion time and *risk factor*. These two weights imply the trade-off between the two components. Because we are dealing with normalized quantities, the weights do not imply that some risk is equivalent to some completion time. By using these two weights, we can transform the "bi" objective formulation to a "uni" objective formulation where we are dealing with one parameter. This is done by combining the corresponding entries for each i^{th} row in the normalized CT and *risk factor* tables to end up with a *normalized trade-off* table. Since *min-min* algorithm can deal with a single parameter minimization, we can apply it as follows. In the first phase: For each task j, the machine that gives the minimum value is determined by scanning the i^{th} row of the *normalized trade-off* table. In the second phase: The task k that has the minimum execution cost is determined and then assigned to the corresponding machine. The algorithm proceeds as in Figure 7.2 except that steps **(3)** and **(9)** are replaced by Equation 7.6 and Equation 7.7, respectively:

$$CT(j,m) = \frac{w_c\ (ET(j,m)+\alpha_m)}{\max_m\{ET(j,m),\alpha_m\}} + \frac{w_r\ TS(j,m)}{\max_m\{TS(j,m)\}} \quad (7.6)$$

$$\alpha_{m^*} = \alpha_{m^*} + ET(j^*,m^*) \quad (7.7)$$

7.7.3 Trust Aware Maximum Risk Algorithm

The objective of this algorithm is to avoid high risk allocations by setting an upper bound on the *expected risk factor*. Suppose that B_{max} is the maximum risk that the resource manager

is willing to take for any individual allocation. A factor Q is added to the completion time of m as follows:

$$CT(j,m) = \begin{cases} CT(j,m) + Q & \text{if } TS(j,m) \geq B_{max} \\ CT(j,m) & \text{otherwise} \end{cases}$$

where Q is a large penalty factor. Then, the *min-min* algorithm proceeds as usual with the modified CT cost. By adding the Q factor to selected entries, we avoid untrustworthy machines. That is, we eliminate those task-machine pairs that have risk more than or equal to B_{max}.

7.8 Performance Evaluation of the Trust-Aware Algorithms

7.8.1 Goals of the Simulation

To highlight the benefits of trust-aware resource management, we investigate the following. First, simulation experiments were conducted to examine the performance of resource management based on security overhead minimization. We accomplished this by examining the improvement in completion time of the trust-aware resource management over the trust-unaware resource management. Second, simulation experiments were constructed to investigate two factors that impact the performance of the risk minimization-based resource management: (a) makespan for the complete schedule and (b) makespan variability. The makespan variability is defined as the variation in makespan as the risk penalty value changes. We accomplished that by measuring makespan that considers only completion time with no risk penalty. Then, we measured the actual makespan considering completion time that includes risk penalty. By varying the risk penalty value (i.e., Q), we can compute the variability of the actual makespan and compare it to the makespan.

7.8.2 Overview

Figure 7.4 shows the simulation model and the simulation entities used in the simulation experiments. In the simulation model, the physical system that consists of a collection of NCDs peering with each other is simulated by a collection of peering simulated NCDs. Each NCD represents a collection of nodes (i.e., clients and resources) as shown in Figure 7.4. The interaction between NCDs is simulated by clients from NCD_j submitting tasks to be executed on resources in NCD_i. Although, a global resource manager that coordinates the allocation of resources, the trust estimates is done is a purely P2P manner as explained in Section 7.2. The trust estimates are evolved and maintained by the behavior trust model proposed in this thesis. The global resource manager is represented by the P2P Grid scheduler as depicted in Figure 7.4. The P2P Grid scheduler is assumed to have knowledge of the execution time of task i on machine j (i.e., ET) and consequently CT, PT, as well as α_i. In addition, the P2P Grid scheduler is assumed to have knowledge of the trust levels between the different NCDs (i.e., ST). These different tables are shown in Figure 7.4.

The predicted direct trust table (PDTT) has structure as shown in Table 7.3, where TL_{ij}^{k} is the offered trust level (OTL) that NCD_i offers to NCD_j to engage in activity within context k. The values in the PDTT are estimate of the trust that exists between the peering NCDs. These values are estimated, evolved, and maintained by the behavior trust model proposed in this thesis. The PDTT used in the simulation here is basically the same PDTT that is used to track the evolution of the trust relationships among the peering NCDs in Section 5.2.

As shown in Table 7.5, an example of ET matrix is generated for $t_{max} = 3$ and $m_{max} = 3$. The ST matrix shown in Figure 7.4 has structure as the ET matrix and its entries are calculated based on Equation 7.3 or 7.4. The PT matrix has structure as the ET matrix and its entries are the sum of the corresponding entries of ET and ST as expressed in Equation 7.2. The α vector shown in Figure 7.4 , consisting of m_{max} entries, represents

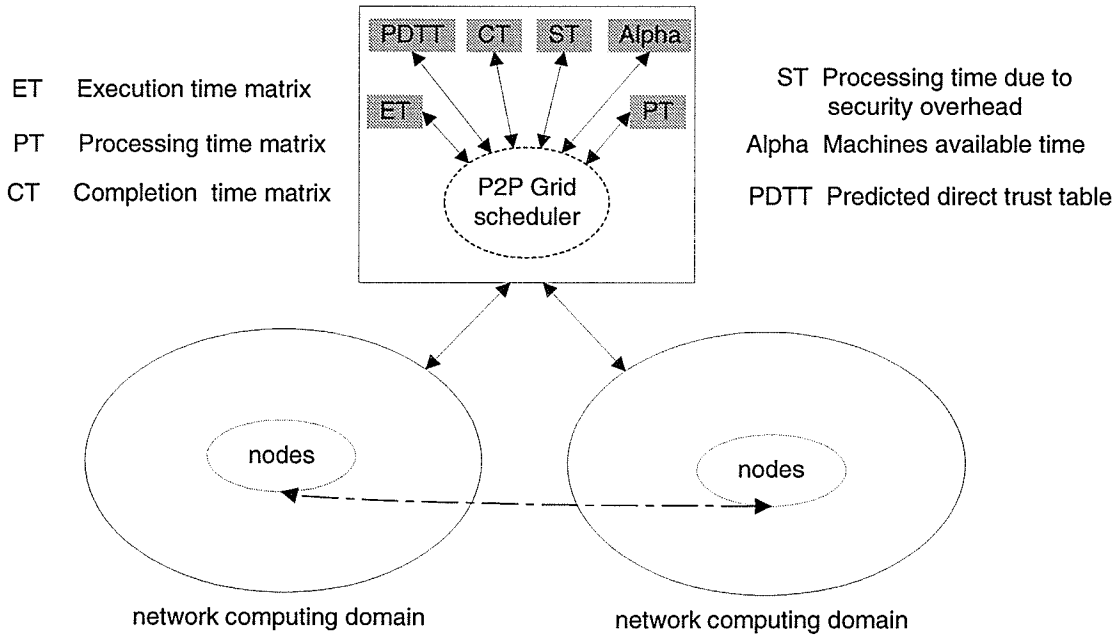


Figure 7.4: Simulation model.

the completion time of the last task assigned to each of the m_{max} machines so far. Initially, the entries in the α vector in set to 0 indicating that all machines are ready to accept tasks. The CT matrix has the same structure as the ET matrix and its entries are the sum of the corresponding entries of PT and α.

7.8.3 Design and Exogenous Parameters

Table 7.6 shows the design and exogenous parameters used in the simulation. The NCDs' transactions process was simulated using a discrete event simulator. The term randomly generated over a range [a, b] means that the number is generated using a discrete (integer-valued) uniform distribution over a, a+1, ..., b inclusive. That is written as U[a, b]. The

Table 7.5: An example execution time matrix. Element in row i column j, $ET(i,j)$ = execution time of task i if assigned to machine j.

i/j	0	1	2
0	50	20	15
1	20	30	15
2	20	60	15

(1) Let Γ_t be an arbitrary constant quantifying task heterogeneity, being smaller for low task heterogeneity.
Let N_t be a number picked from the uniform random distribution with range [1, Γ_t]
(2) Let Γ_m be an arbitrary constant quantifying machine heterogeneity, being smaller for low machine heterogeneity.
Let N_m be a number picked from the uniform random distribution with range [1, Γ_m]
(3) Sample N_t for t_{max} times to get a vector $q[0..(t_{max}-1)]$.
(4) Generate the EEC matrix, $e[0..(t_{max}-1), 0..(m_{max}-1)]$ as follows:
for t_i from 0 to $(t_{max}-1)$
for m_j from 0 to $(t_{max}-1)$
pick a new value for N_m
$e[i,j] = q[i] \times N_m$

Figure 7.5: Generating the execution time matrix.

transactions that take place among the NCDs arrive at the NCDs based on a Poisson process. The design parameter $reps$ denotes how many times the simulation run is repeated. In simulating our trust model, $reps$ is set deterministically at 10. That is, each point in Table 7.11 to Table 7.18 is the result of 10 simulation runs.

The trust model topology used in the simulation consists of 20 NCDs (i.e., $NCDs\_num = 20$). The number of tasks generated from NCDs' clients are set deterministically to 10,000 (i.e., $t_{max} = 10,000$). The type of contexts (i.e., $ToCs$) is randomly generated over a range

Table 7.6: Design and Exogenous parameters used in the simulation.

Symbol	Definition	Design parameter values
$reps$	How many times the simulation is repeated for each point in the graphs and tables	$reps = 10$
$NCDs\_num$	Number of NCDs	$NCDs\_num = 20$
t_{max}	Maximum number of tasks	$t_{max} = 10,000$
m_{max}	Maximum number of machines	$m_{max} = 20$
ToC	Type of context required for each task	$ToC = U[1,4]$
RTL	Required trust level	$RTL = U[1,6]$
OTL	Offered trust level	$OTL = U[1,5]$
Γ_m	Constant quantifying low machine heterogeneity	$\Gamma_m = 10$
Γ_t	Constant quantifying low task heterogeneity	$\Gamma_t = 1000$

$[1,4]$ meaning that each task involves in at least one ToC but no more than four $ToCs$. The two $RTLs$ are randomly generated over a range $[1,6]$ representing the five values of trust levels (Please refer to Table 3.1). Clients or resources can specify $RTL = 6$ meaning that they want to enforce enhanced security or they are risk averse. Whereas, the OTL values are randomly generated over a range $[1,5]$ representing the 5 trust levels (Please refer to Table 3.1). The constants 10 and 1000 are used as the values of Γ_m for low machine heterogeneity and Γ_t for low task heterogeneity [96], respectively. These are used according to Figure 7.5 to generate the execution time matrix (i.e., ET). As explained in Section 7.3, an ET can be consistent or inconsistent. From the ET generated above (i.e., Figure 7.5, a consistent ET can be generated by sorting the execution times across the machines for each task. Whereas, an inconsistent ET can be generated by simply leaving the ET generated

above as such.

7.8.4 Conceptual Model

Figure 7.6 shows the simulation control flow. First, we initialize $rep\_num = 0$ for running the simulation $reps$ times. The setting phase is explained in Section 7.8.3. In the initialization phase, simulation entities such as ET, ST, PT, PDTT, α, and CT are set to their initial conditions. First, the m_{max} entries in the vector α is initialized to 0 stating that all machine are ready at time 0. Second, the ET matrix in initialized as shown in Figure 7.5. The PDTT is assumed to be known and it contains the estimated trust levels that exist among the different NCDs. For simplicity, we assume that these trust relationships are constants for the duration of the simulation time. The PDTT entries are initialized to be randomly generated over a range $[1, 5]$. The entries of the ST matrix are initialized according to Equation 7.3 or 7.4 based on the RTLs specified by the the client side and the resource side and the OTL specified by the resource side. The entries of PT and CT are initialized based on ET, ST, and α.

In the selection phase, an NCD_j is randomly chosen to represent a client X's task wanting to engage in activities on resource Y in NCD_i. The variable *tasks_num* is incremented by 1. If the scheduling scheme used is MCT (i.e., an online or immediate mode), then the task is allocated to a machine as soon as it arrives and begins execution. On the other hand, if the scheduling scheme used is min-min or sufferage (i.e., batch mode), then the following is done. First, the task is inserted into a meta-task. If it is time to map the meta-task, then the scheduler starts mapping the tasks within the meta-task into machines. Otherwise, a another task is generated and inserted into the meta-task.

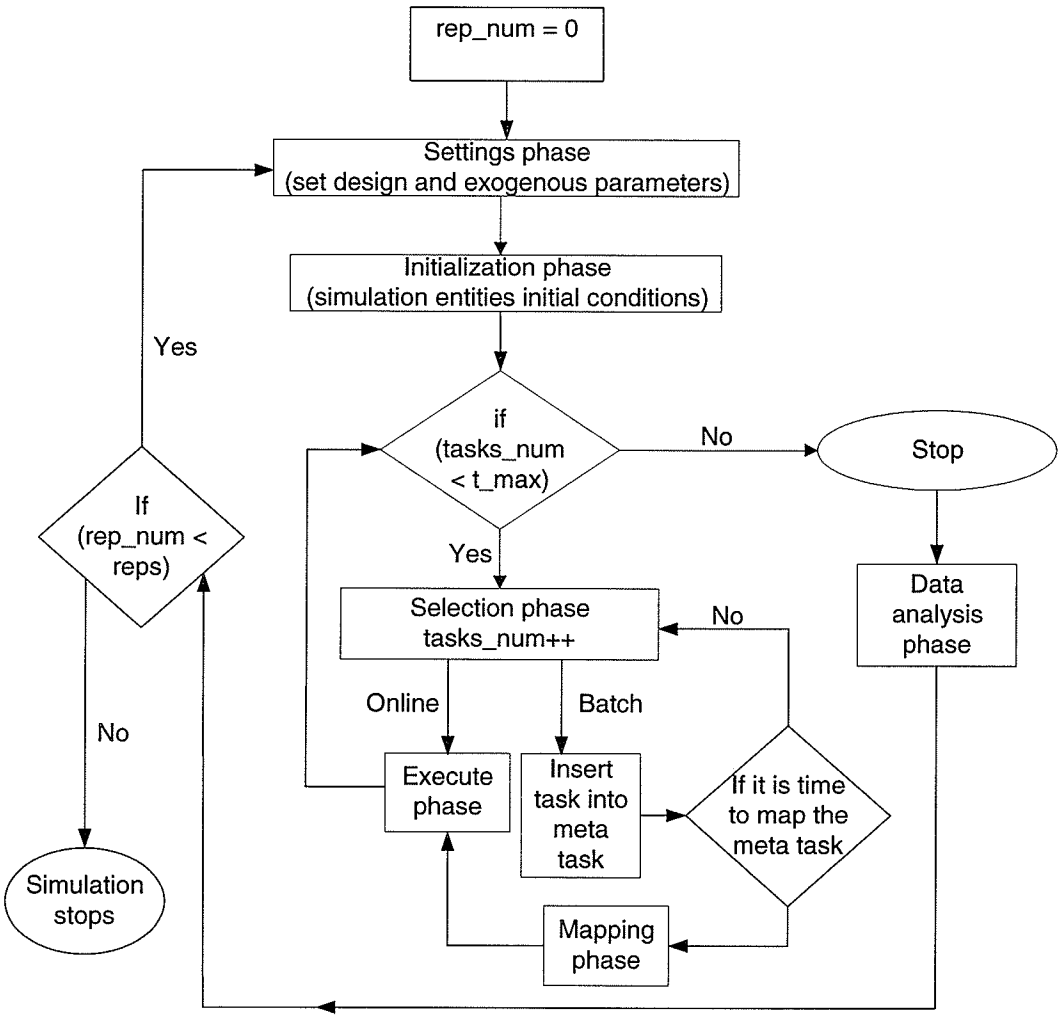


Figure 7.6: Simulation control flow.

7.8.5 Performance Metrics

In the first set of the simulation experiments, we examine the process of resource management based on security overhead minimization. The goal of resource management based on security minimization is to map tasks to machines such that the overall completion time,

over m machines, is minimized. We used the average completion time metric to compare the performance of the trust-aware and trust-unaware algorithms based on security overhead minimization.

In the second set of simulation experiments, we investigate two factors that impact the performance of trust-aware resource management: (a) makespan [96] for the complete schedule and (b) makespan variability. Makespan is defined as $max_{i \in m}(\alpha_i)$, where m is the number of machines. Makespan is a measure of the throughput of the whole resource allocation process. Robustness is measured as the largest variation in risk penalty that can be tolerated without running out of bounds on the completion time. The risk penalty is computed as a proportion (ξ) of the ET. Therefore, ξ indicates the variation in risk penalty.

Let $\mathrm{RO(j, m)}$ denote the risk overhead incurred by assigning task j onto machine m. Then, the risk overhead is calculated as follows:

$$\mathrm{RO(j, m)} = \frac{\mathrm{TS(j, m)}}{6} \, \xi \, \mathrm{EC(j, m)} \tag{7.8}$$

We normalize the *risk factor* by dividing it by 6 (i.e., $\frac{TS(j,m)}{6}$) to indicate how likely it is to incur a risk penalty. For example, *risk factor*s of 1 and 4 indicate $1/6 = 17\%$ and $4/6 = 67\%$ chance of incurring the risk penalty, respectively. By varying ξ, we can increase the variation of risk penalty and examine if the actual makespan runs out of bounds on the expected makespan.

Although the completion time of a task j mapped onto machine m is computed without considering the risk overhead, the actual completion time of the task j is going to be determined by the completion time plus the risk overhead of executing j on m. The risk overhead determines the cost (in terms of time) that the resource management incurs because of misbehaved resources.

7.8.6 Event Generation

In this section, we discuss the type of events that change the state of the system and present a flow chart to show how each event is being generated in our system. There are 4 event types, namely *arrival*, *mapping*, *begin*, and *complete*. An arrival event is created to simulate task j arrival at the scheduler. A mapping event is created to simulate the process of assigning a task to a machine and computing the execution order of the tasks assigned to that particular machine. A begin event is created to simulate the execution of a task on a particular machine. Finally, a *complete* event is created to simulate the readiness of a machine after finishing executing a certain task.

The event generation process starts by creating an arrival event and inserting it into the calendar (i.e. event list). This step is taken to initialize the calendar. Then, the event-generation process enters the main body that loops as long as the calendar is not empty. An event is picked up from the calendar and based on the event type, the system state changes as explained below. After event processing control returns to the main body to select the next event, the standard event-oriented simulation algorithm [90] is followed. We now detail the state changes occurring for each event type.

If the event type is ARRIVAL, two things can happen. First, if the scheduling scheme used is MCT (i.e., an online or immediate mode), then a begin event is generated to simulate the allocation and the start of the execution of the arrived task on the allocated machine. The begin event in inserted into the calendar and a new event is picked from the calendar. Second, if the scheduling scheme used is min-min or sufferage (i.e.,a batch mode), then the arrived task is inserted into the meta-task. If the current time equals the current scheduling event time, then a mapping event is created to map the meta-task. the mapping event in inserted into the calendar. Otherwise, a new event is picked from the calendar.

If the event type is MAPPING, the appropriate scheduler (min-min or sufferage) is used to map the tasks within the meta-task to machines. One a task is mapped to a machine, a

begin event is generated in inserted into the calendar. Then, a new event is picked from the calendar.

If the event type is BEGIN, the mapped task starts execution on its machine. Then, a new event is picked from the calendar. If the event type is COMPLETE, the task that started execution on its machine completes its execution and the update process takes place. That is, the appropriate data structure is updated (i.e., ET, CT, PT, α, and ST). The update process is explained below. Then, a new event is picked from the calendar.

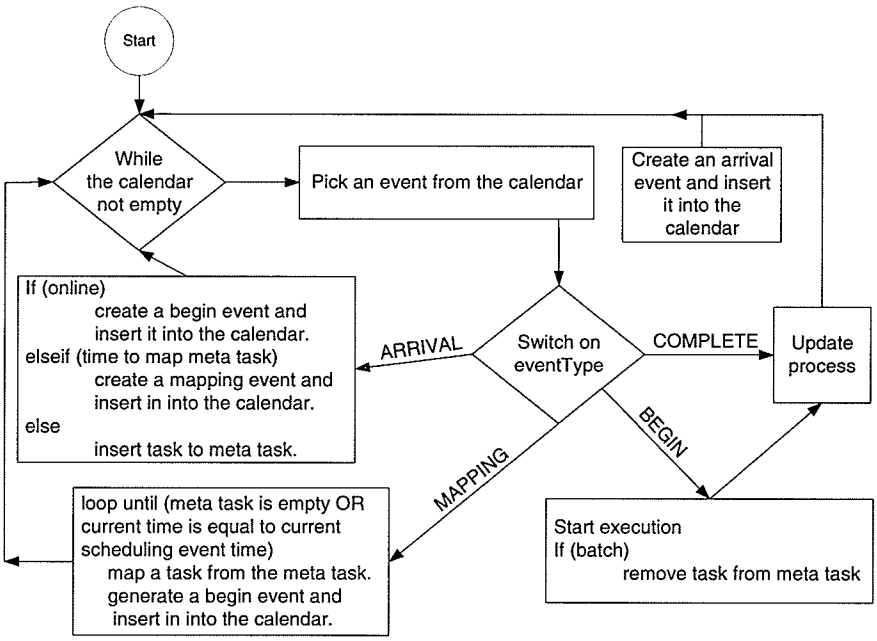


Figure 7.7: Event generation control flow.

After task j has completed execution on machine m, then the scheduler (i.e., MCT, min-min, or sufferage) will perform the following:

- If the scheduler mode is batch, then task j is removed from the meta-task.

- The ready time of machine m is updated. That is, the entry α_m is updated to reflect the ready time of machine m after executing task j.

- The completion time of machine m is updated. That is $CT(j, m)$ is updated forall j. For example, if task 0 is to begin execution on machine 1 and the processing time $PT(0, 1) = 15$ units time. Then Table 7.7 will updated as in Table 7.8.

Table 7.7: An example completion time matrix. Element in row i column j, $CT(i, j)$ = completion time of task i if assigned to machine j.

i/j	0	1	2
0	50	20	15
1	20	30	15
2	20	60	15

Table 7.8: An example completion time matrix. Element in row i column j, $CT(i, j)$ = completion time of task i if assigned to machine j.

i/j	0	1	2
0	50	35	15
1	20	45	15
2	20	75	15

7.8.7 Implementation

For the simulation model, we developed our own discrete-event simulator in C language running on the UNIX environment. Our simulation program can be run through the command line. The simulation program is called *trustAppl* with arguments passed from the command line to define the design and exogenous parameters shown in Table 7.6. The simulation program is run using the command line as follows:

trustAppl -m 5 -t 100 -u 1 -p poisson 1.0 -h c_lolo -a min-min -r 8343324 -D DATA > trustApplOut

where m is the number of machines taking the value of 5, t is the number of tasks taking the value of 100, u is a boolean variable taking the values of 0 or 1. If $u = 1$, then the scheduling algorithm is trust-aware. Otherwise, the scheduling algorithm is trust-unaware. p indicates the arrival process follows a poisson distribution for the arrival process with mean 1.0, h indicates the category of ET matrix is consistent low-low heterogeneity, a indicates the scheduling algorithm is min-min, r indicates that 8343324 is the seed of the random number generator, and finally the results of the simulation into a file called trustApplOut.

We also automated the process of running the simulation by including a batch file that calls a Perl script. The batch file has the following format:

trustApplMin-min 20 10000 10

The Perl script is called *trustApplMin-min*. This script runs the simulation with the following parameters: 20 m_{max}, 10000 t_{max}, and 10 $reps$. This script runs the simulation using the min-min scheduler using both c_lolo (i.e., consistent low-low heterogeneity ET matrix) and ic_lolo (i.e., inconsistent low-low heterogeneity ET matrix. Other Perl scripts are created to run the other types of the simulation experiments (i.e., using MCT and sufferage).

7.8.8 Verification and Validation

Trace File

In this section, we verify the simulation model by examining a trace file that was created as output of a sample run. We ran our simulation program using the command line as follows: **trustAppl -m 5 -t 8 -p poisson 1.0 -h i_lolo -a mct -r 8343324 -D DATA > trustApplOut** The trace file is the output of running the simulation model for 3 machines (i.e., $m_{max} = 3$), 8 tasks (i.e., $t_{max} = 8$), the arrival process follows a poisson distribution for the arrival process with mean 1.0, the category of ET matrix is inconsistent low-low heterogeneity, the scheduling algorithm is mct, the seed of the random number generator is 8343324, and finally the results of the simulation into a file called trustApplOut.

In Section 7.8.6, we discussed the event generation scheme, In the following exercise, we trace the events generated by the simulation program and show that trustAppl is working as intended. Table 7.9 shows the total processing time (i.e., execution time plus security overhead) of the 5 tasks if executed on each of the 3 machines.

Table 7.9: A processing time matrix. Element in row i column j, $PT(i, j)$ = total processing time if task i assigned to machine j.

i/j	0	1	2
0	661.96	470.00	1089.11
1	1280.08	814.96	1368.25
2	582.69	1074.50	1179.24
3	389.44	273.24	278.95
4	1568.78	1441.07	1593.25

We will trace the events generated and shown in Table 7.10. Event type 0 represents an arrival event, 1 represents a complete event, and 2 represents a BEGIN event. Since the

scheduling algorithm used in this example is MCT, the mapping event is not represented in Table 7.10 (Please refer to 7.8.6). The arrival events of the 5 tasks are in lines **(1)**,**(3)**, **(5)**, **(7)**, and **(8)**, respectively. When the first task (i.e., task 0 arrives at simulation time 0.728677 to the scheduler, the ready times of all the machines are 0. Since the scheduling algorithm is MCT, task 0 is assigned a machine as soon as it arrives and it begins execution at time 0.728677 on machines 1. From Table 7.9, machine 1 gives the earliest processing time for task 0. Line **(2)** shows the ready time for machine 1 changed to $PT(0,1) = 470.00$ plus the simulation time 0.728677. As expected all arrival events generated completed on lines **(10)**, **(11)**, **(13)**, **(14)**, and **(15)**, respectively.

Table 7.10: Events generated from running a simulation example.

Line number	Event type	Simulation time	Task number	Machine number	Machines ready times		
					0	1	2
1	0	0.728677	0		0.00	0.00	0.00
2	2	0.728677	0	1	0.00	470.73	0.00
3	0	0.781710	1		0.00	470.73	0.00
4	2	0.7817104	1	0	1390.95	470.73	0.00
5	0	2.836522	2		1390.95	470.73	0.00
6	2	2.836522	2	2	1390.95	470.73	1182.07
7	0	4.049642	3		1390.95	470.73	1182.07
8	0	5.094594	4		1390.95	743.98	1182.07
9	2	470.731841	3	1	1390.95	2185.05	1182.07
10	1	470.731841	0	1	1390.95	2185.05	1182.07
11	1	743.976480	3	1	1390.95	2185.05	1182.07
12	2	743.976480	4	1	1390.95	2185.05	1182.07
13	1	1182.074436	2	2	1390.95	2185.05	1182.07
14	1	1390.945666	1	0	1390.95	2185.05	1182.07
15	1	2185.047824	4	1	1390.95	2185.05	1182.07

7.9 Simulation Results and Discussion

7.9.1 Investigating Resource Allocation Based on Security Overhead Minimization

When not using trust, the idea is to map a task j to machine m that gives us the earliest completion time without considering the security overhead. Although the completion time was calculated in terms of the execution time of j on m plus the security overhead of executing j on m, the security overhead is not considered when mapping i to m. For the trust-aware heuristics, the security overhead is considered when mapping as well as when calculating the completion time of executing j on m.

Tables 7.11 and 7.12, show the benefit of integrating the trust notion into an MCT-based RMS. Table 7.11 was run for the inconsistent LoLo heterogeneity with 5 machines. In Table 7.11, the completion time was reduced by about 38%. Table 7.12 was run for the consistent LoLo heterogeneity with 5 machines. In Table 7.12, the completion time was reduced by about 35%.

Table 7.11: Comparison of average completion time for inconsistent LoLo heterogeneity using the MCT heuristic.

Number of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement in completion time when using trust
50	No	92.86%	5,817.38	36.99%
	Yes	93.56%	3,665.23	
100	No	96.29%	11,244.77	37.59%
	Yes	96.12%	7,018.38	

Tables 7.13 and 7.14 show the benefit of integrating the trust notion into a Minmin-based RMS. Table 7.13 was run for the inconsistent LoLo heterogeneity with 5 machines.

Table 7.12: Comparison of average completion time for consistent LoLo heterogeneity using the MCT heuristic.

Number of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement in completion time when using trust
50	No	93.90%	4,786.27	34.44%
	Yes	93.96%	3,137.78	
100	No	96.51%	9,117.53	34.26%
	Yes	96.81%	5,994.25	

Table 7.13: Comparison of average completion time for inconsistent LoLo heterogeneity using the Minmin heuristic.

Number of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement in completion time when using trust
50	No	90.56%	3,983.04	23.51%
	Yes	90.87%	3,046.79	
100	No	93.71%	7,227.78	23.34%
	Yes	94.35%	5,540.47	

In Table 7.13, the completion time was reduced by almost 24%. Table 7.14 was run for the consistent LoLo heterogeneity with 5 machines. In Table 7.14, the completion time was

Table 7.14: Comparison of average completion time for consistent LoLo heterogeneity using the Minmin heuristic.

Number of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement in completion time when using trust
50	No	93.17%	3,750.59	25.28%
	Yes	92.53%	2,802.27	
100	No	96.15%	6,712.27	25.32%
	Yes	95.91%	5,012.39	

reduced by almost 26%.

Table 7.15: Comparison of average completion time for inconsistent LoLo heterogeneity using the Sufferage heuristic.

Number of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement in completion time when using trust
50	No	92.59%	5,257.31	39.66%
	Yes	93.96%	3,172.09	
100	No	96.60%	9,609.78	38.40%
	Yes	97.08%	5,919.49	

Table 7.16: Comparison of average completion time for consistent LoLo heterogeneity using the Sufferage heuristic.

Number of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement in completion time when using trust
50	No	94.14%	4,473.05	32.67%
	Yes	95.32%	3,011.81	
100	No	97.11%	8,356.33	33.19%
	Yes	97.33%	5,582.56	

Tables 7.15 and 7.16 show the benefit of integrating the trust notion into a Sufferage-based RMS. Table 7.15 was run for the inconsistent LoLo heterogeneity with 5 machines. In Table 7.15, the completion time was reduced by almost 40%. Table 7.16 was run for the consistent LoLo heterogeneity with 5 machines. In Table 7.16, the completion time was reduced by almost 33%.

In summary, the experiments performed to evaluate the overhead of securing remote computation indicate that the overhead is significant and techniques for minimizing such overhead by eliminating redundant application of secure operations can greatly enhance the

overall performance. Simulations performed to evaluate the effectiveness of incorporating trust into resource management heuristics indicate that the overall quality of the schedules obtained by the resource allocation process can be improved by about 40%.

7.9.2 Investigating Resource Allocation Based on Risk Minimization

As shown in Tables 7.17 and 7.18, the two weights wc and wr are not applicable (NA) to the *min-min* and *maximum risk* algorithms. From Table 7.17, we can observe that the *min-min* algorithm's actual makespan does not tolerate large risk penalty variation. For example, as the risk penalty increases from 0.01 to 10.0, the actual makespan of the *min-min* algorithm increases from $16,094.54$ to $101,871.41$, respectively.

On the other hand, the other two algorithms, which minimizes the risk, perform much better and are more resistant to large variation of risk penalty. In Table 7.17 and for the *trade-off* algorithm with $wr = 0.8$, we can observe that as the risk penalty increases from 0.01 to 10.0, the actual makespan increases from $40,325.78$ to just $42,899.47$, respectively. Notice that when $wr = 0.0$, the *trade-off* algorithm performs very close to the *min-min* algorithm because the allocation decisions are made without considering risk (i.e., the weight assigned to risk is zero). The same observations can be made on Table 7.18.

We can conclude that the *min-min* algorithm, which does not consider any risk when making allocation decisions (i.e., the allocation decisions are made considering only the completion times), performs very poor in terms of actual makespan as the risk penalty increases. Hence, the *min-min* algorithm is not robust. On the other hand, the *trade-off* and the *maximum risk* algorithms, which make the allocation decisions considering both completion times and risk, perform well in terms of completion times and robustness. They both tolerate large variation in risk penalty without going out of bounds on the completion times.

7.10 Summary

As an application, the trust model is used to incorporate trust into a resource management system such that the allocation decisions are trust cognizant. In this chapter, we enhance the global resource manager of a P2P Grid such that it tracks the trust relationships among the different resource and client domains and brings together only those domains that have high levels of trust among them in any given allocation. The performance evaluation involved performing simulations to evaluate the benefit on incorporating trust into resource management systems. The simulation results indicate that if the allocation decisions are made without considering trust, the resource management system performs very poor in terms of actual makespan as the risk penalty increases and hence, it is not robust. On the other hand, if the resource management is trust-aware such that it makes allocation decisions considering both completion times and risk, it will perform well in terms of completion times and robustness. In Summary, the simulation results indicate that due to trust awareness, the performance of resource management systems can improve the overall quality of the schedules obtained by the allocation process.

Table 7.17: Expected and actual makespans of various resource management algorithms using an inconsistent LoLo heterogeneity

ξ value	RMS algorithm	wc value	wr value	Expected makespan	Actual makespan
0.01	*min-min*	NA	NA	16,011.28	16,094.54
	trade-off	0.0	1.0	178,672.41	178,686.17
		0.2	0.8	40,323.55	40,325.78
		0.4	0.6	39,474.82	39,477.04
		0.6	0.4	40,244.76	40,244.76
		0.8	0.2	35,273.99	35,274.98
		1.0	0.0	15,985.72	16,055.42
	maximum risk	NA	NA	30,195.42	30,211.99
1.0	*min-min*	NA	NA	16,011.28	24,336.83
	trade-off	0.0	1.0	178,672.41	180,048.14
		0.2	0.8	40,323.55	40,546.34
		0.4	0.6	39,474.82	39,697.60
		0.6	0.4	40,244.76	40,244.76
		0.8	0.2	35,273.99	35,372.55
		1.0	0.0	15,985.72	25,297.04
	maximum risk	NA	NA	30,195.42	31,853.08
2.0	*min-min*	NA	NA	16,011.28	32,662.38
	trade-off	0.0	1.0	178,672.41	181,423.87
		0.2	0.8	40,323.55	40,769.12
		0.4	0.6	39,474.82	39,920.38
		0.6	0.4	40,244.76	40,244.76
		0.8	0.2	35,273.99	35,471.10
		1.0	0.0	15,985.72	34,784.72
	maximum risk	NA	NA	30,195.42	33,683.85
5.0	*min-min*	NA	NA	16,011.28	58,584.02
	trade-off	0.0	1.0	178,672.41	185,551.06
		0.2	0.8	40,323.55	41,437.47
		0.4	0.6	39,474.82	40,588.74
		0.6	0.4	40,244.76	40,754.60
		0.8	0.2	35,273.99	37,185.09
		1.0	0.0	15,985.72	63,247.76
	maximum risk	NA	NA	30,195.42	39,442.76
10.0	*min-min*	NA	NA	16,011.28	101,871.41
	trade-off	0.0	1.0	178,672.41	192,429.71
		0.2	0.8	40,323.55	42,899.47
		0.4	0.6	39,474.82	42,451.45
		0.6	0.4	40,244.76	42,798.63
		0.8	0.2	35,273.99	41,317.49
		1.0	0.0	15,985.72	110,686.16
	maximum risk	NA	NA	30,195.42	49,040.93

Table 7.18: Expected and actual makespans of various resource management algorithms using a consistent LoLo heterogeneity

ξ value	RMS algorithm	wc value	wr value	Expected makespan	Actual makespan
0.01	*min-min*	NA	NA	29,329.68	29,465.75
	trade-off	0.0	1.0	159,694.38	159,709.95
		0.2	0.8	47,283.77	47,283.77
		0.4	0.6	46,455.13	46,457.37
		0.6	0.4	44,506.76	44,506.76
		0.8	0.2	38,164.80	38,170.80
		1.0	0.0	30,880.70	31,042.49
	maximum risk	NA	NA	37,379.00	37,395.98
1.0	*min-min*	NA	NA	29,329.68	45,194.34
	trade-off	0.0	1.0	159,694.38	161.251.30
		0.2	0.8	47,283.77	47,283.77
		0.4	0.6	46,455.13	46,679.13
		0.6	0.4	44,506.76	44,506.76
		0.8	0.2	38,164.80	38,788.64
		1.0	0.0	30,880.70	47,680.00
	maximum risk	NA	NA	37,379.00	39,352.39
2.0	*min-min*	NA	NA	29,329.68	62,247.77
	trade-off	0.0	1.0	159,694.38	162,808.22
		0.2	0.8	47,283.77	47,283.77
		0.4	0.6	46,455.13	46,903.12
		0.6	0.4	44,506.76	44,506.76
		0.8	0.2	38,164.80	39,446.96
		1.0	0.0	30,880.70	64,958.86
	maximum risk	NA	NA	37,379.00	41,328.56
5.0	*min-min*	NA	NA	29,329.68	113,408.08
	trade-off	0.0	1.0	159,694.38	167,478.98
		0.2	0.8	47,283.77	47,507.24
		0.4	0.6	46,455.13	47,575.10
		0.6	0.4	44,506.76	46,272.47
		0.8	0.2	38,164.80	41,693.09
		1.0	0.0	30,880.70	116,795.44
	maximum risk	NA	NA	37,379.00	47,257.06
10.0	*min-min*	NA	NA	29,329.68	198,675.25
	trade-off	0.0	1.0	159,694.38	175,263.58
		0.2	0.8	47,283.77	52,822.19
		0.4	0.6	46,455.13	51,324.57
		0.6	0.4	44,506.76	50,433.86
		0.8	0.2	38,164.80	45,914.84
		1.0	0.0	30,880.70	203,189.74
	maximum risk	NA	NA	37,379.00	57,137.90

Chapter 8

Conclusions and Future Work

8.1 Overview

In this thesis, a trust model was presented for P2P systems, where different network computing domains peer with each other to create large systems by sharing their resources or services while not binding themselves to each other through agreements. We defined the trust model and described schemes used in the model. We also outlined mechanisms for computing the notions of trust and reputation.

Our trust model uses an *accuracy* concept to enable peer review-based mechanisms to function with imprecise trust metrics, the imprecision is introduced by peers evaluating the same situation differently. Simulation results show that a reputation-based trust model reaches an acceptable level of capability after a certain number of transactions. However, as the number of dishonest NCDs increase, the model becomes slow in reaching the acceptable level of capability.

To reduce the trust model's sensitivity to dishonest NCDs, we introduced an *honesty* concept to handle the situation where NCDs intentionally lie about other NCDs for their own benefit. Simulation results indicate that incorporating the *honesty* concept into the

trust model, limits the effect of dishonest NCDs by preventing them from providing recommendations.

Another feature of our model is the flexibility to weigh direct trust and reputation differently. Combining both of direct trust and reputation equally (i.e. $\alpha = 0.5$) enables the trust model to perform at its highest capability when using the accuracy and honesty measures. Because dishonest NCDs are filtered out of the recommender set, reputation reinforces direct trust and combining them results in a higher capability than relying only on one of them. On the other hand, when using just the accuracy measure, combining direct trust and reputation lowers the trust model capability because of the negative impact it has on direct trust. Having the flexibility of combining direct trust and reputation gives the trust model the leeway to choose the strategy that best fits it.

Estimating trust levels is an important issue to determine future estimates of the trust value. We demonstrated that an exponentially weighted moving average algorithm is inadequate for estimating the future value of a trust parameter. A new filter algorithm was introduced and it was shown to detect periodic cheating.

A study was performed to examine the scalability of the trust model. The scalability study demonstrated that the trust model is scalable at the NCD-level from scaling factor of 1 to a scaling factor of 3 if $\Psi > 0.8$. Although, scalability at the node-level was not examined, this study lays the first step towards understanding the overall scalability of the trust model.

As an application of our trust model, we incorporate trust-awareness into a peer-to-peer Grid's resource management system such that the allocation decisions are trust cognizant. The simulations performed to evaluate the effectiveness of the modifications indicate that due to trust awareness, the performance of resource management systems can improve the overall quality of the schedulers.

8.2 Thesis Contributions

This thesis proposes a behavior trust model for P2P structured large-scale network computing domains. The contributions of this thesis are two folds: contributions to the model itself and contributions to the utility of the model. These contributions are as follows:

- A trust modeling framework that separates accuracy and honesty. This separation enables the trust model to work effectively even when the number of honest NCDs are less that 50% of the NCDs' population. Further, knowing the number of honest NCDs gives a confidence level about the effectiveness of the trust model.

- A novel framework for determining and maintaining honest set of recommenders. Having honest set of recommenders contributes to the efficiency of the trust model by sending queries to just honest recommenders and hence all the recommendations received are valuable. This does not just increase the efficiency of the trust model, but also utilizes the network bandwidth. Other trust models ask for recommendations from NCDs whether they are honest or not and then by processing these recommendations, the recommendations that are believed to be dishonest are discarded.

- The applicability of modeling trust by integrating it into resource management systems. To the best of our knowledge, no existing literature directly addresses the issues of trust aware resource management systems. Simulation results indicate that trust aware resource management systems tolerate the largest variation in risk penalty without running out of bounds on the completion time.

- A scalability analysis of the trust model is done and a scalability metric is derived. The scalability study demonstrates that the trust model is scalable at the NCD-level under certain conditions. As trust model is scaled up from a scaling factor of 1 to a scaling factor of 3 where $\Psi > 0.8$, the simulation studies show that the value delivered keeps up with the cost.

- A trust level estimation schemes were proposed to be used in the trust model. Trust level estimation using traditional running average schemes suffer from drawbacks of returning high estimates despite the periodic occurrence of low values in the sequence (i.e., an NCD can periodically cheat and still maintain a high trust level). The simulation results show that the proposed trust level estimation schemes outperform the traditional running average schemes in detecting periodic cheating.

8.3 Directions for Future Work

The work as part of this thesis paves the way for the following initiatives. Future work includes several issues that are an extension of the trust model proposed in this thesis. These issues are: (a) dynamics of trust, (b) using trust decay to shape the recommender set, and (c) formal representation and estimation of the trust notion and using fuzzy sets for modeling the uncertainty in trust levels.

8.3.1 Dynamics of Trust

When computing direct trust and reputation, trust may decay with time. For example, if NCD_s trust domain NCD_t at a given trust level based on experience five years ago, NCD_s's trust in NCD_t today is likely to be lower unless they have continued to interact since then. Therefore, a decay function needs to be applied when obtaining direct trust levels or when giving recommendations as illustrated in Equations (3.6) and (3.7). There are some issues that need to be sorted out before the decay function can be simulated and examined. First, how does the decay function apply to the trust levels. We need to explore the issue of quantity versus time. That is, by how much a trust level should be decayed and what is a reasonable time interval to decide applying the decay. For example, we might decide that if NCD_s has not interacted with NCD_t for x number of time units, then

NCD_s's trust in NCD_t should be decreased by y. The question is, what is x and what is y. Second, should there be a generic decay function mechanism and leave the implementation details to each individual NCD. Also, what is the trade offs of implementing a generic decay function that is used by all the NCDs versus individual NCD's decay implementations. Finally, how does that the different implementation approaches of the decay function effect the overall performance of the trust model.

8.3.2 Using Trust Decay to Shape the Recommender Set

Since the peer reviews play a vital role in estimating the trust level, the recommender set is a very important component in the trust model proposed in this thesis. The trust model uses the trusted allies set T to shape the recommenders set R and the objective is to have an honest set of recommenders. The trust model deploys the trusted allies checking mechanisms to ensure that all of its recommenders are honest. It turns out that having honest recommenders can give misleading reviews. Suppose that NCD_r is a recommender that NCD_s uses to collect reviews about other NCDs. At this point, let us assume that NCD_r is honest. If NCD_r is inactive and has not interacted with other NCDs for a long time, its trust levels in its DTT become stale. When NCD_s receives recommendations from NCD_r, these recommendations maybe as misleading as recommendations received from a dishonest NCD. Therefore, NCD_s should have active and honest recommenders in its R. This scenario emphasizes and illustrates the importance of a decay function. Hence, we can further shape R by using the decay function.

8.3.3 Coherent and Incoherent Trust Models

Since NCDs give recommendations using their DTTs, the structure of DTT used by the trust model is essential for recommendations to be useful (i.e., the structure of DTT is essential for the trust model to learn from recommendations). Section 4.3.2 of the thesis

classifies a trust model to be coherent if it uses a coherent DTT. Otherwise, the trust model is considered to be incoherent. Now, the question is "What makes a DTT coherent or incoherent?".

Section 4.3.2 of the thesis defines a coherent trust model as follows: if the variation along any column of the trust model's DTT is below a certain threshold, then the trust model is considered to be coherent. Otherwise, the trust model is considered to be incoherent. That is, the trust model will learn from recommendations as long as the variation along any column of the trust model's DTT is below a certain threshold.

Putting more thought into this, the effectiveness of the accuracy measure is a good indicator of whether the recommendations are useful to the learning process of the trust model. This is because before an NCD can use the recommendation received from a recommender, the NCD must adjust the recommendation to reflect the recommender's accuracy (i.e. this process determines if the recommendation is useful or not). Since the accuracy measure works by comparing the ITL to the recommendation and then shifting up or down (i.e. adjusting) the recommendation accordingly to narrow the gap between ITL and the recommendation, the accuracy measure will be effective as long as the recommendation is consistently low or consistently high in relation to the ITL. Hence in this case, the trust model will learn from recommendations. This notion is not captured by the coherent versus incoherent definition given in Section 4.3.2. Therefore, further investigation needs to be carried out to correct the definition of trust model coherency and its effect on the overall performance of the trust model.

8.3.4 Scalability at the Node-level

The trust model maintains trust levels between NCDs. That is, each NCD has a global trustworthiness in the eyes of other NCDs. This global reputation of an NCD is affected by how trustworthy its nodes (resource and clients) behave when engaged in a transaction.

Determining the scalability of the trust model, one should examine the scalability at the NCD-level as well as the scalability at the node-level. We carried out the scalability and the NCD-level and demonstrated that the trust model is scalable. As nodes join, leave, and while they are part of an NCD, additional costs incur and these costs increase as the number of nodes per an NCD increase. A future work is to generalize the scalability study to include and investigate the scalability at the node-level.

8.3.5 Formal Trust Representation and Estimation

Trust involves specifying and reasoning about beliefs. In the trust model proposed in this thesis, there are various parameters that contribute to the trust evaluation process. including our own belief (direct trust), number of recommenders, recommenders' accuracy and honesty, and the weight given to direct trust and recommenders' opinion (reputation). These parameters can be represented as fuzzy values. The trust notion is a subjective and vague point of view about how the future behavior of other entities would fit in the expectations of others. Therefore, fuzzy sets can be used to combine trust levels and formally define the notion of trust. We designed the trust model mathematically and it will be interesting to incorporate a simple, rule-based IF X AND Y THEN Z approach to solving the trust level estimation. In addition, since the trust model is a learning process, the learning operation can be formalized using fuzzy logic. Fuzzy logic is relatively new scientific field and attempts to capture the fuzziness or imprecision of the real world.

8.4 Concluding Remarks

Organizing large-scale network computing systems in a *peer-to-peer* (P2P) fashion is a manifestation of one of the fundamental design principles on the Internet. Current research is focusing on improving P2P systems and one of the future directions is to combine P2P

and Grid technologies. One of the key issues identified in the evolution of P2P technologies is the trust issue.

This thesis presents a trust model for P2P structured large-scale network computing systems. The most widely used trust modeling approach is to use a network of recommenders to obtain references and use these to predict the trust between two entities. This approach is known to suffer from drawbacks such as trustworthiness of the recommenders and scalability.

To address this problem, a solution is proposed where a recommender is independently evaluated using accuracy and honesty measures. This thesis explains using simulation results how the separation of accuracy and honesty helps in addressing the above issues. To demonstrate the utility of the trust model, a trust aware resource allocation model is developed such that it can be used to make trust cognizant resource allocations. To the best of our knowledge, this is the first study to integrate trust into resource management systems. The simulation results indicate that significant preferences gain can be obtained through this integration.

As a final note, our trust model implicitly provides two levels of incentives for NCDs: (a) by modeling honesty, the trust model provides a mechanism for giving an incentive to recommenders to truthfully give recommendations and hence be cooperative in the P2P environment. If a recommender is dishonest, it will be isolated and the rest of the P2P environment will not ask recommendations from it and (b) by modeling trust, the model provides another level of incentive to NCDs to be trustworthy and behave as expected. By enabling trust-aware resource management systems, best behavior is motivated and that in turn improves the overall system performance.

Appendix A

Abbreviations used in this thesis

ADTT	Actual Direct Trust Table
CDTT	Computed Direct Trust Table
CT	Completion Time
DTT	Direct Trust Table
ET	Execution Time
EWMA	Exponential Weighted Moving Average
ITL	Instantaneous Trust Level
LMFF	Liberally Modified Flipflop
LoLo	Low Task Low Machine Heterogeneity
MFF	Modified Flipflop
NC	Network Computing
NCD	Network Computing Domain
OTL	Offered Trust Level
PDTT	Predicted Direct Trust Table
PT	Processing Time
R	Set of Recommenders

RC	Reputation Cost
RMS	Resource Management System
RO	Risk Overhead
RTL	Required Trust Level
RTT	Recommender Trust Table
SR	Success Rate
ST	Processing Time Due To Trust Supplement
T	Set of Trusted Allies
TA	Trust Agent
TL	Trust Level
TM	Transaction Monitoring
ToC	Type of Context
TS	Trust Supplement
WMFF	Weighted Modified Flipflop

Appendix B

Calculating the success rate

A pseudo-code is presented in Figure B.1 to illustrate how SR(t) is calculated in our simulation experiments. It can be observed that each entry (except when $i = j$) in ADTT is compared to its corresponding entry in PDTT and the result is a success or a failure. A success if $PDTT_i(j)$ correctly predicts $ADTT_i(j)$ and a failure otherwise. Each of ADTT and PDTT has i rows and j columns. For example, $ADTT_i(j)$ indicates the trust level that NCD_i has in NCD_j. Please notice that we ignored the context and time stamps in the ADTT and PDTT for simplification purposes.

```
(1)  success = failure = 0
(2)  for ( i=0 to NCDs_num
(3)     for (j=0 to NCDs_num
(4)          if ( i ≠ j )
(5)               if ( (ADTT_i(j) ∈ [3,5]) and (PDTT_i(j) ∈ [3,5]) )
(6)                    success++
(7)               elseif ( (ADTT_i(j) ∈ [1,2]) and (PDTT_i(j) ∈ [1,2]) )
(8)                    success++
(9)  success rate = success / ( NCDs_num × (NCDs_num − 1) ) ×100
```

Figure B.1: Calculating the success rate.

Appendix C

Perl script controlling the trust model simulation

In Figure C.1, lines **(1)** to **(8)** parses the command line and assigns the appropriate line command argument to its corresponding simulation design parameter. Line **(9)** prints to the screen the command line arguments assuring that the command line is parsed correctly. Line **(10)** sets a different random number generator's seed for each of the $reps$ runs. Lines **(11)** to **(13)** sets the design parameters α, $NCDs\_dishonest$, and mon_freq, respectively. Line **(14)** specifies how many times the simulation needs to be repeated. Line **(15)** specifies a loop to be repeated 3 times, where j an index for $NCDs\_dishonest$ (i.e., $NCDs\_dishonest[j] = 0$ when $j = 0$, $NCDs\_dishonest[j] = 15$ when $j = 1$, and $NCDs\_dishonest[j] = 20$ when $j = 2$). Line **(16)** specifies a loop to carry out the mon_freq values and line **(17)** specifies a loop to carry out the α values. Finally, Lines **(18)** to **(20)** run the executable simulation program *trustSim* and direct the output to an output file called *trustSim.out*.

A Perl script interfacing with our simulation program

```
(1)  $NCDs_num = $ARGV[0];
(2)  $transactions_num = $ARGV[1];
(3)  $recs_num = $ARGV[2];
(4)  $allies_num = $ARGV[3];
(5)  $cons_check = $ARGV[4];
(6)  $accu_check = $ARGV[5];
(7)  $epsilon = $ARGV[6];
(8)  $reps = $ARGV[7];
(9)  printf("%s %s %s %s %s %s %s %s", $ARGV[0], $ARGV[1], $ARGV[2], $ARGV[3],
     $ARGV[4], $ARGV[5], $ARGV[6], $ARGV[7])
(10) @random_seed = ("6787367", "8423979", "83444890", "834389", "64679867", "8343324",
     "4487461", "1435439", "652210", "4871035");
(11) @alpha = ("1.0", "0.5", "0.0");
(12) @NCDs_dishonest = ("0", "15", "20");
(13) @mon_freq = ("1", "5", "10", "20");
(14) for ($i=0; $i < $reps; $i++)
(15)     for ($j=0; j < 3; $j++)
(16)         for ($k=0; $k < 4; $k++)
(17)             for ($n=0; $n < 3; $n++)
(18)                 $cmdLine = sprintf("trustSim -n %s -r %s -e %s -d %s -t %s -l %s -c %s
                     -a %s -m %s -o %s -u %s -y %s -p poisson 1.0 -r %s -D DATA >>
                     trustSim.out", $NCDs_num, $recs_num, $allies_num, $NCDs_dishonest[$j],
                     $transactions_num, $alpha[$n], $cons_check, $accu_check, $mon_freq[$k],
                     $mon_freq[$k], $mon_freq[$k], $epsilon, $random_seed[$i]);
(19)             print $cmdLine;
(20)             system $cmdLine;
```

Figure C.1: Perl script controlling our trust model simulation process.

Bibliography

[1] H. Nissenbaum, "Will Security Enhance Trust Online, or Supplant it?," in *Trust and Distrust Within Organizations: Emerging Perspectives, Enduring Questions*, R. Kramer and K. Cook, Eds., pp. 155–188. Russell Sage Publications, New York, NY, 2004.

[2] C. Corritore, B. Kracher, and S. Wiedenbeck, "On-line trust: concepts, evolving themes, a model," *International Journal of Human Computer Studies*, vol. 58, no. 6, pp. 737–758, June 2003.

[3] IBM Global Services, "Trust: Opening up the opportunities of e-business," Tech. Rep., *IBM Global Services, Executive Tek Report*, Jan. 2002.

[4] D. D. Clark, J. Wroclawski, and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet," in *Proceedings of the 2002 conference on applications, technologies and protocols for computer communications.*, Aug. 2002, pp. 347–356.

[5] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," in *the 33rd Hawaii International Conference on System Sciences - Volume 6*, Jan. 2000, pp. 04–07.

[6] L. Rasmusson, A. Rasmusson, and S. Jansson, "Reactive Security and Social Control," in *19th National Information Systems Security Conference*, Oct. 1996, pp. 701–703.

[7] L. Rasmusson and S. Jansson, "Simulated Social Control for Secure Internet Commerce," in *New Security Paradigms Workshop*, Sep. 1996, pp. 18–26.

[8] F. B. Schneider, Ed., *Trust in Cyberspace*, National Academy Press, Washington, D.C., 1999.

[9] H. Nissenbaum, "Securing Trust Online: Wisdom or Oxymoron," in *Boston University Law Review Office of Technology Assessment. Electronic Surveillance and Civil Libraries*, June 2001, pp. 635–664.

[10] M. A. Patton and A. Josang, "Technologies for Trust in Electronic Commerce," *Electronic Commerce Research*, vol. 4, no. 1, Jan. 2004.

[11] S. Grabner-Krauter and E. A. Kaluscha, "Emperical research in on-line trust: a review and critical assessment," *International Journal of Human Computer Studies*, vol. 58, no. 6, pp. 783–812, June 2003.

[12] B. Friedman, P. H. Khan, and D. C. Howe, "Trust Online," *Communications of The ACM*, vol. 43, no. 12, pp. 34–40, 2000.

[13] C. Brenton, *Mastering Network Security*, SYBEX Network Press, Alameda, California,, 1999.

[14] E. Rescorla, "Security holes... Who cares?," in *12th USENIX Security Symposium*, Aug. 2003, pp. 75–90.

[15] D. Moore, C. Shannon, and J. Brown, "Code-Red, a case study on the spread and victims on an Internet warm," in *Internet Measurement Workshop*, Nov. 2002.

[16] Microsoft Corporation, "Microsoft Security," http://www.microsoft.com/security.

[17] Cisco Systems Inc., "Security at Cisco," http://www.cisco.com/security.

[18] M. Nelson, "Security bug hits Microsoft Java Virtual Machine," *Java World*, Sep. 1999.

[19] Sun Microsystems Inc., "Sun Microsystems Security," http://www.sun.com/security.

[20] T. Grandison and M. Sloman, "Trust Management Tools for Internet Applications," in *1st International Conference on Trust Management (iTrust 2003)*, May 2003, pp. 91–107.

[21] A. Oram, Ed., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly and Associates, Sebastopol, CA, 2001.

[22] B. Yang and H. Garcia-Molina, "Designing a Super-peer Network," in *19th International Conference on Data Engineering (ICDE)*, Banbalore, India, Mar. 2003.

[23] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An Experiment in Public-Resource Computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[24] R. Brayanrd, D. Kosic, A. Rodriguez, J. Chase, and A. Vahdat, "Opus: An overlay Peer Utility Service," in *15th International Conference on Open Architectures and Network Programming (OPENARCH)*, June 2002.

[25] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms," in *5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Dec. 2002, pp. 239–254.

[26] I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing," in *2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Feb. 2003.

[27] B. Dragovic, S. Hand, T. Harris, E. Kotsovinos, and A. Twigg, "Managing Trust and Reputation in the XenoServer Open Platform," in *1st International Conference on Trust Management (iTrust 2003)*, May 2003, pp. 59–74.

[28] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer Computing," Tech. Rep. HPL-2002-57, HP Laboratories, Palo Alto, Mar. 2002.

[29] R. Dingledine, M. J. Freedman, and D. Molnar, "Accountability," in *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, A. Oram, Ed., pp. 271–340. O'Reilly and Associates, Sebastopol, CA, 2001.

[30] M. Waldman, L. F. Cranor, and A. Rubin, "Trust," in *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, A. Oram, Ed., pp. 271–340. O'Reilly and Associates, Sebastopol, CA, 2001.

[31] B. Yu and M. P. Singh, "An Evidential Model for Distributed Reputation Management," in *1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, July 2002, pp. 294–301.

[32] S. Breban and J. Vassileva, "A Coalition Formation Mechanism Based on Inter-Agent Trust Relationships," in *1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, July 2002, pp. 306–308.

[33] A. J. Menezes, P. C. Oorshot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, Fifth Edition, 2001.

[34] C. Adams and S. Farral, "Internet X.509 public key infrastructure certificate management protocols," Mar. 1999.

[35] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder policy specification language," in *Workshop on Policies for Distributed Systems and Networks*, Jan. 2001, pp. 18–38.

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

[37] G. Pierre and M. van Steen, "A Trust Model for Peer-to-Peer Content Distribution Networks," Draft paper. Work in progress. `http://www.cs.vu.nl/~gpierre/publi/TMPTPCDN_draft.php2`, Nov. 2001.

[38] B. K. Alunkal, "Grid Eigen Trust: A Framework for Computing Reputation in Grids," M.S. thesis, Dept. of Computer Science, Graduate College of the Ilinois Institute of Technology, Chicago, 2003.

[39] F. Azzedin and M. Maheswaran, "Evolving and Managing Trust in Grid Computing Systems," in *EEE Canadian Conference on Electrical & Computer Engineering (CCECE '02)*, May 2002, pp. 1424–1429.

[40] S. Brainov and T. Sandholm, "Incentive Compatible Mechanism for Trust Revelation," in *1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, July 2002, pp. 310–311.

[41] I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Fransisco, CA, 1999.

[42] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," in *IEEE Conference on Security and Privacy*, 1996.

[43] M. Blaze, "Using the KeyNote Trust Management System," AT&T Research Labs, 1999.

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

[45] K. Hickman, "The SSL Protocol (version 2)," Netscape Communications Corporation, Feb. 1995.

[46] M. Blaze, J. Ioannidis, and A. Kermytis, "Trust Management for IPSec," in *Network and Distributed System Security Symposium (NDSS '01)*, pp. 139–151. San Diego, CA, Feb. 2001.

[47] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *ACM Conference on Computers and Security*, Nov. 1998, pp. 83–92.

[48] I. Foster and C. Kesselman, "The Globus project: A status report," in *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 4–18.

[49] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *5th Annual International Conference on Mobile Computing and Networks (Mobicom '99)*, Aug. 1999, pp. 24–35.

[50] L. Xiong and L. Liu, "A Reputation-Based Trust Model for Peer-to-Peer eCommerce Communities," in *IEEE International Conference on E-Commerce(CEC'03)*, Jun. 2003, pp. 270–280.

[51] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A Reputation-based Approach for Choosing Reliable Resources in Peer-to-Peer Networks," in *9th ACM Conference Computer and Communications Security*, Nov. 2002, pp. 207–216.

[52] Document Revision 1.2, "The Gnutella Protocol Specification v0.4," `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`, June 2001.

[53] R. Chen and W. Yeager, "Poblano: A Distributed Trust Model for Peer-to-Peer Networks," `http://security.jxta.org`, 2001.

[54] L. Gong, "JXTA: A network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, May/June 2001.

[55] M. Kim and B. Noble, "Mobile Networks Estimation," in *7th Annual Conference Mobile Computing and Networking*, July 2001, pp. 298–309.

[56] S. Sen and N. Sajja, "Robustness of Reputation-based Trust: Boolean Case," in *1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, July 2002, pp. 288–293.

[57] K. Aberer and Z. Despotovic, "Managing Trust in a Peer-2-Peer Information System," in *10th International Conference on Information and Knowledge Management (CIKM'01)*, Nov. 2001, pp. 310–317.

[58] S. Hand, T. Harris, E. Kotsovinos, and I. Pratt, "Controlling the XenoServer Open Platform," in *6th International Conference on Open Architectures and Network Programming (OPENARCH'03)*, Apr. 2003.

[59] R. Jurca and B. Faltings, "Towards Incentive-Compatible Reputation Management," in *AAMAS 2002 Workshop on Deception, Fraud & Trust in Agent Societies*, 2002, pp. 138–147.

[60] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems," *Software Practice & Experiance*, vol. 32, no. 2, pp. 135–164, Feb. 2002.

[61] F. Azzedin and M. Maheswaran, "A Trust Brokering System and Its Application to Resource Management in Public-Resource Grids," in *2004 International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Apr. 2004, pp. 26–30.

[62] F. Azzedin and M. Maheswaran, "Trust Modeling for Peer-to-Peer based Computing Systems," in *12th IEEE Heterogeneous Computing Workshop (HCW 2003)(in conjunction with IPDPS 2003)*, Apr. 2003.

[63] F. Azzedin and M. Maheswaran, "Towards Trust-Aware Resource Management in Grid Computing Systems," in *First IEEE International Workshop on Security and Grid Computing*, May 2002, pp. 452–457.

[64] T. Grandison, *Trust Management for Internet Applications*, Ph.D. thesis, Dept. of Computing, University of London, July 2003.

[65] C. Corritore, B. Kracher, and S. Wiedenbeck, "Trust and Tchnology," *International Journal of Human Computer Studies*, vol. 58, no. 6, pp. 633–635, June 2003.

[66] B. Dragovic, E. Kotsovinos, S. Hand, and P. R. Piezuch, "XenoTrust: Event-based distributed trust management," in *2nd International Workshop on Trust and Privacy in Digital Business*, Sep. 2003.

[67] A. Baier, "Trust and Antitrust. Ethics," in *Ethics*, 1986, pp. 231–260.

[68] N. Luhmann, "Familiarity, confidence, trust: Problems and alternatives," in *Trust: Making and Breaking Cooperative Relations*, D. Gambetta, Ed., pp. 94–107. Basil Blackwell, New York, 1988.

[69] B. Misztal, "Trust in Modern Societies," 1996, Polity Press, Cambridge MA.

[70] J. Lewis and A. Weigert, "Trust as a social reality," *Social Forces*, vol. 63, no. 4, pp. 967–985, Jun. 1985.

[71] iTrust, "First International Conference on Trust Management," `http://www.trstmanagement.ccirc.ac.uk`, Mar. 2003.

[72] iTrust: Working group on trust management in dynamic open systems, "iTrust," `http://www.itrust.uoc.gr`, June 2002.

[73] M. Langheinrich, "When Trust Does Not Compute - The Role of Trust in Ubiquitous Computing," in *5th International Conference on Ubiquitous Computing, Workshop on Privacy*, Oct. 2003.

[74] J. B. Rotter, "Generalized expectancies for interpersonal trust," *American Psychologist*, vol. 26, pp. 443–452, 1971.

[75] L. S. Wrightsman, "Interpersonal trust and attidutes toward human nature," in *Measures of personality and social psychological attitudes*, J. P. Robinson, P. R. Shaver, and L. S. Wrightsman, Eds., pp. 372–412. Academic Press, San Diego, CA, 1991.

[76] G. L. De Furia, *Interpersonal Trust Survey: Facilitator's Guide*, Wiley & Sons, Incorporated, John, New York, NY, 1991.

[77] N. Luhmann, "Trust and Power," in *Wiley, Chichester*, 1979.

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

[79] T. F. Lunt, "Detecting intruders in computer systems," in *Conference Auditing and Computer Technology*, 1993.

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

[81] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations," *International Journal on Supercomputer Applications*, vol. 15, no. 3, 2001.

[82] M. A. Baker, R. Buyya, and D. Laforenza, "The Grid: International efforts in global computing," ACM Computing Surveys, Oct. 2000.

[83] K. Krauter and M. Maheswaran, "Towards a High Performance Extensible Grid Architecture," in *14th International Symposium on High Performance Computing Systems and Applications (HPCS 2000)*, June 2000.

[84] W. W. Johnston, D. Gannon, and B. Nitzberg, "Information Power Grid Implementation Plan: Research, Development, and Testbeds for High Performance, Widely Distributed Collaborative, Computing and Information Systems Supporting Science and Engineering," Tech. Rep., NASA Ames Research Center, http://www.nas.nasa.gov/IPG, 1999.

[85] M. Maheswaran and K. Krauter, "A Parameter-based Approach to Resource Discovery in Grid Computing Systems," in *1st IEEE/ACM International Workshop on Grid Computing (Grid '00)*, Dec. 2000, pp. 181–190.

[86] M. S. Blumenthal and D. D. Clark, "Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world," *ACM Trans. Internet Technology*, vol. 1, no. 1, pp. 70–109, Aug. 2002.

[87] D. Clark, "Talk at DARPA NGI PI meeting," in *Tucson, AZ.*, Mar. 1998.

[88] S. Lee, R. Sherwood, and B. Bhattacharjee, "Cooperative Peer Groups in NICE," in *IEEE Infocom*, Apr. 2003.

[89] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, no. 10, Oct. 2000.

[90] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, New York, NY, 1991.

[91] J. S. Chase, D. C. Anderson, P. N. Thakar, and A. M. Vahdat, "Managing Energy and Server Resources in Hosting Centers," in *18th Symposium Operating Systems Principles (SOSP)*, Oct. 2001.

[92] P. Jogalekar and M. Woodside, "Evaluating the Scalability of Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 6, pp. 589–603, June 2000.

[93] D. Talia and P. Trunfio, "Towards a Synergy Between P2P and Grids," *IEEE Internet Computing*, vol. 7, no. 4, pp. 94–96, July 2003.

[94] G. Fox, D. Gannon, S. Ko, S. Lee, S. Pallickara, M. Pierce, X. Qiu, X. Rao, A. Uyar, M. Wang, and W. Wu, "Peer-to-Peer Grids," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A. Hey, Eds., pp. 471–490. John Wiley & Sons Ltd, West Sussex, England, 2003.

[95] M. Wang, G. Fox, and S. Pallickara, "A Demonstration of Collaborative Web Services and Peer-to-Peer Grids," in *International Conference on Information Technology: Coding and Computing (ITCC'04)*. Apr. 2004.

[96] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, Nov. 1999.

[97] O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.

[98] U. Erlingsson and F. B. Schneider, "SASI Enforcement of Security Policies: A Retrospective," in *New Security Paradigms Workshop*, Sep. 1999, pp. 87–95.

[99] C. Small and M. Seltzer, "MiSFIT: A Tool for Constructing Safe Extensible C++ Systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 33–41, July/Sep. 1998.

[100] I. Foster, A. Roy, and V. Sander, "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation," in *8th International Workshop on Quality of Service (IWQoS '00)*, June 2000, pp. 181–188.

[101] M. Maheswaran, "Quality of service driven resource management algorithms for network computing," in *1999 International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA '99)*, June 1999, pp. 1090–1096.

[102] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. M. Figueira, J. Hayes, G. Obertelli, J. M. Schopf, G. Shao, S. Smallen, N. T. Spring, A. Su, and D. Zagorodnov, "Adaptive Computing on the Grid Using AppLeS," *IEEE Transactions Parallel Distributed Systems*, vol. 14, no. 4, pp. 369–382, Apr. 2003.

[103] K. Czajkowski, I. Foster, C. Kesselman, "Resource Co-Allocation in Computational Grids," in *8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, 1999, pp. 219–228.