AUTONOMIC TRUST MANAGEMENT IN DYNAMIC SYSTEMS

by

BRENT JASON LAGESSE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2009

To Ed and Pat Scholling

ACKNOWLEDGEMENTS

I would like to thank my parents for their encouragement in all that I have done. I would also like to thank all of my friends and family for all their help and especially the distractions they have provided to keep me sane over the past five years.

I would like to thank Mohan Kumar and Matthew Wright for all of their help in reviewing and working with me during my graduate studies. I would also like to thank Manfred Huber, Gergely Zaruba, and Yonghe Liu for serving on my thesis committee. I would like to thank all the members of PICO who have helped me with my work by asking me questions and giving me feedback.

Finally, I would like to thank the National Science Foundation (NSF ECCS-0824120), National Physical Science Consortium, and Lawrence Livermore National Laboratory for their financial support during my graduate study.

July 20, 2009

ABSTRACT

AUTONOMIC TRUST MANAGEMENT IN DYNAMIC SYSTEMS

BRENT JASON LAGESSE, Ph.D.

The University of Texas at Arlington, 2009

Supervising Professors: Mohan Kumar and Matthew Wright

Research in pervasive computing is aimed at creating environments where users can seamlessly benefit from ubiquitous computing resources despite the complexity of the environment. Providing security in such systems is a difficult task since traditional security mechanisms often require significant user attention and do not scale well to large, mobile, and open environments. To combat this problem, distributed trust has been proposed to provide security in pervasive systems. While much research has been performed in the area, many vulnerabilities and insufficiencies still exist, especially in mobile ad-hoc systems that cannot support distributed trust mechanisms requiring pre-existing infrastructure and cooperation. Dynamic pervasive systems operate in highly dynamic environments that introduce additional challenges such as intermittent connectivity and lack of infrastructure.

This dissertation addresses several problems pertinent to the design and deployment of distributed trust mechanisms in dynamic pervasive systems. In particular, this dissertation presents the design and evaluation of the following framework and mechanisms to enhance security in dynamic systems. The Distributed Trust Toolkit (DTT) is a modular framework for the design and deployment of distributed trust

mechanisms over a wide variety of systems, networks, and devices. Adaptive Resource Exploration (AREX) and Reliable Service Composition (ReSCo) are built for two specific classes of applications that occur frequently in dynamic systems. AREX uses a game theoretic approach to motivate strategic, malicious entities to attack less often. ReSCo is designed for dynamic service composition systems and works by adapting to make selections of compositions paths and nodes. Social Trust (SoTru) is a system for augmenting trust mechanisms such as AREX and ReSCo with information from users' social networks to reduce risk and enhance their performance. A unique feature of the above contributions is that each can be used independently or in combination to address challenges in secure dynamic systems. DTT facilitates the integration of AREX, ReSCo and SoTru into existing dynamic systems. AREX and ReSCo provide scalable, low cost security mechanisms that provide protection despite hostile, open, and mobile environments. When used together, with the addition of SoTru, the ideas presented in this dissertation can be used to enhance the effectiveness and seamlessness of security in dynamic systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Trust is a term used in many fields, including psychology, sociology, economics, and computer science, and has a variety of definitions [1, 2, 3, 4]. Trust is used to create some expectation of success in a cooperation between two separate entities. This research is about establishing trust effectively and efficiently within dynamic systems. This chapter introduces motivations, challenges, and current solutions.

1.1 Motivation

Advances in technologies such as mobile phones, PDAs, and sensors along with wireless communication technologies are increasing system dynamicity. In dynamic systems, it is critical to be able to quickly and securely access resources and services provided by other entities in the system. Providing efficient and secure access to resources and services increases in difficulty as systems are made more open and dynamic. The credibility of individual users becomes too complex to monitor and enforce, so systems must break from traditional security mechanisms and instead rely on methods for establishing trust. Existing methods for establishing trust rely on third parties to establish trust; however, relying on a third party introduces vulnerabilities that can lead to negative results.

In addition to vulnerabilites and complexities of the trust mechanisms themselves, the framework on which trust mechanisms are built increases the difficulty of deployment. Trust mechanisms should be easily portable to a variety of devices, op-

erating systems, and networks that are represented in the large, diverse set of options available to users.

1.2 Distributed Trust

While trust is difficult to define, it is used in security to determine which entities should cooperate. The reason for using trust is that in many distributed systems, especially those of an open and dynamic nature, it is not feasible to use traditional techniques that involve a central source guaranteeing the reliability of users and resources. Furthermore, traditional security mechanisms run counter to the pervasive computing vision – technology that fades into the background, working but unnoticed. A number of mechanisms exist for establishing trust in dynamic systems [5, 6, 7, 8, 9, 10, 11, 12]; however, each has vulnerabilities and insufficiencies that need to be addressed. The most challenging of these vulnerabilities to protect against involve the collusion between malicious nodes to disrupt the accuracy and distribution of trust information to the nodes that require it.

Most trust mechanisms in distributed systems consist of either using certificates as a guarantee or reputation derived from the experiences of other entities in the system. Both of these techniques require the trust of a third party, even if, as in the case of some reputation systems, the trust may not be complete. The difficulty with trusting a third party or even multiple third parties is that the trusting entity is susceptible to attacks by a malicious third party as a result of deception. For example, a third party could behave completely honestly and then attack or third parties could collude to lie about each other and make it appear that they are benign nodes.

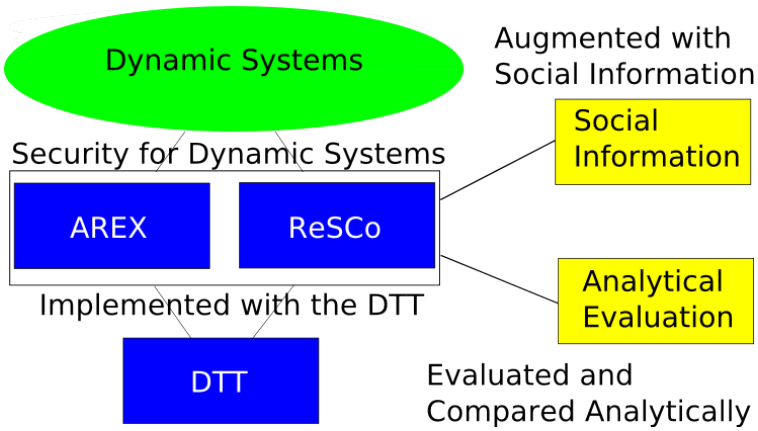

Figure 1.1. Overview of Dissertation Components.

1.3 Major Research Contributions

This dissertation addresses the challenges associated with automated establishment of trust between entities in an invisible and resource-effective manner within the constraints of a dynamic system. This is accomplished through the establishment of a trust framework and trust mechanisms designed to operate without the use of a third party. Furthermore, this dissertation explores the augmentation of the initial trust mechanisms when the use of third parties is a valid design choice. Figure 1.1 shows how the contributions fit together. The major contributions are as follows.

1.3.1 A Generic Framework for Implementing Trust Mechanisms

Effective security mechanisms are essential to the widespread deployment of dynamic pervasive systems. Pervasive environments are expected to have many different types of hardware and software. Pervasive computing research endeavors to provide users the flexibility to access a variety of services in a transparent manner, regardless of what devices, technologies or interfaces they use [13]. Providing security and reliability of services is particularly challenging in these environments. Much of the re-

search focus on security in pervasive computing has revolved around distributed trust management. While such mechanisms are effective in specific environments, there is no generic framework for deploying and extending these mechanisms over a variety of pervasive systems. This dissertation presents the design and implementation of a novel framework called the Distributed Trust Toolkit (DTT) for implementing and evaluating trust mechanisms in pervasive systems. The DTT facilitates the extension and adaptation of trust mechanisms by abstracting trust mechanisms into interchangeable components. Furthermore, the DTT provides a set of tools and interfaces to ease implementation of trust mechanisms and facilitate their execution on a variety of platforms and networks. Specifically, trust mechanisms are abstracted into *trust blocks* composed of Presentation, Computation, and Communications Protocol components. These components can be extended or replaced to adapt trust mechanisms for specific requirements and environments. The DTT Daemon executes trust blocks in addition to storing trust information and providing communication interfaces and an API for the trust blocks.

The DTT is general enough to apply to a variety of types of trust, such as certificate-based, role-based, and reputation-based, in addition to the mechanisms introduced in this dissertation. In addition to representing trust mechanisms, the framework also eases evaluation of the individual components of trust mechanisms in a variety of environments and against different attacker models. Furthermore, the modular nature of the framework allows the design and implementation of interchangeable components to ease implementation and adaptation of trust mechanisms.

1.3.2 Trust Information Sharing Mechanism

In addition to the adaptability and extensibility provided by this design, experiments have been conducted to demonstrate that use of the DTT improves utilization

of resources and enhances performance of existing trust mechanisms in pervasive systems. The DTT introduces a mechanism to utilize the heterogeneity of pervasive computing environments to improve the sharing of trust information – specifically within Personal Area Networks (PANs), where pre-existing trust of applications is implicit. Information is shared within a trust group to reduce the amount of energy and time spent acquiring trust information. Trust groups are formed based on a *host node* that all member nodes attach to and make requests through. When a member node needs access to new trust information, it queries the host node that either satisfies the request with recently cached information or queries other nodes in the system to acquire the requested information.

1.3.3 Algorithm for selecting entities to trust

Service composition schemes create high-level application services by combining several basic services. A challenge in composing services dynamically is selecting which node to access the service from. Selecting a service hosted by an unreliable or malicious node could result in degraded quality of service and potentially significant security breaches. This dissertation presents a stochastic selection mechanism designed to accomodate context-aware adaptation of trust establishment. The algorithm balances the exploration of unknown entities with the utilization of well-known and well-trusted entities. In doing so, the selection algorithm mitigates the effects of mobility, dynamicity, and traitor attacks to which open systems are vulnerable. Service composition schemes for dynamic, open systems such as mobile peer-to-peer must be cognizant of the possibility of failures and attacks. In open systems, it is seldom feasible to guarantee the trustworthiness of each node prior to access; however, there may be several possible ways to compose the same high-level service, each having a different (though possibly overlapping) set of nodes that can satisfy the

composition. This dissertation addresses the service selection problem with Reliable Service Composition (ReSCo), a mechanism designed to identify trustworthy nodes and determine reliable compositions in dynamic, open systems.

ReSCo is a modular, adaptive middleware component that selects from possible composition paths and nodes to enhance the reliability of service compositions. This dissertation presents extensive simulation results of ReSCo to illustrate its suitability as a middleware component for open, dynamic service composition systems. The effectiveness of ReSCo and selection modules to successfully create service compositions in both static and mobile distributed systems is validated through analytical and simulation results.

1.3.4 Game Theoretic Algorithm for Determining Trust

In open, dynamic systems, participants may need to access resources from unknown users. A critical security concern in such systems is the access of faulty resources, thereby wasting the requester's time and energy and possibly causing system damage. A common approach to mitigating this problem involves reputation mechanisms; however, since reputation relies on cooperation, a reputation mechanism's effectiveness can be significantly diminished in hostile environments. Reputation systems also require substantial communication among peers leading to: i) vulnerability to errors caused by intermittent connectivity; ii) message delivery disruptions caused by malicious peers; and iii) energy-sapping message overheads. To overcome this problem, a low-cost, adaptive mechanism designed to provide security for peers in hostile and uncertain environments was developed. This mechanism is implemented in a system for Adaptive Resource Exploration (AREX) that increases the system's utility for benign peers and decreases the system's utility for malicious peers. AREX reduces energy costs, protects benign peers, and diminishes malicious peers' motiva-

tion to attack in a variety of applications even in hostile environments. Furthermore, AREX can utilize system context to further enhance its adaptation process and improve performance.

AREX uses a utility model for the interaction of providing and consuming entities defined in this dissertation. Based on the utility model, the dissertation defines and discusses a game-theoretic algorithm for determining trustworthy entities. The algorithm adapts to approximate a Nash equilibrium when the game is played against entities that also play a Nash equilibrium. When entities play strategies other than a Nash equilibrium, the algorithm adapts to improve utility based on the strategy that is currently in play by the opposing entities, or in the case of faulty entities, to avoid those that prove to be more faulty than others. Like the selection algorithm, the exploration algorithm is designed to utilize context information about the state of the system to further improve performance.

1.3.5 Evaluation Framework for Trust Mechanisms

Trust mechanisms accumulate opinions of other peers in order to reduce uncertainty and enhance reliable interaction. While multitudes of trust mechanisms have been proposed in the literature, there is no published work that deals with a general framework for quantitative comparison of trust mechanisms. This dissertation proposes and investigates a novel framework for the quantitative analysis of trust mechanisms. The proposed framework enables researchers and developers to easily determine the suitability of a trust mechanism for a particular application, compare trust mechanisms, and predict their performance. The proposed framework is validated through analysis and comparison with simulation results.

It is likely that the overwhelming number of mechanisms have been created due to the lack of a framework to evaluate existing ones. Given such a framework, appli-

cation developers can focus on their development and simply pick a trust mechanism that is most suitable for their needs.

Many surveys of trust mechanisms [14, 15, 16] exist that compare, analyze, and classify reputation mechanisms and their characteristics. Unfortunately, these surveys only include qualitative assessments. Qualitative assessments are good for summarizing how a trust mechanism works or, in some cases, seeing if a trust mechanism can be implemented in a particular system. These surveys fail to take a quantitative perspective and thus preventing researchers from using them to make statements about how much better one trust mechanism is over another or make predictions regarding the effect of changing the system parameters or attacker models. Many trust mechanisms provide simulation results demonstrating their effectiveness, but it is difficult to discern how these results will scale to other environments and attacker models, as these simulations are performed with little consistency between trust mechanisms. This analytical framework eases the compatibility of comparisons between trust mechanisms.

1.3.6 Social Augmentation of Trust Mechanisms

The use of social computing and social networking has recently flourished as social networking sites have grown in popularity and pervasive computing resources have allowed for people to stay well-connected with access to social networking resources through mobile phones and other similar devices. This dissertation proposes to utilize information produced by relationships within social networks to assist in the establishment of trust for other pervasive computing applications. This dissertation proposes Social Trust (SoTru), a mechanism for utilizing social network relations to assist in the establishment of trust. SoTru permits trust mechanisms to utilize both implicit and explicit social networking information. Furthermore, the trust mecha-

nism can determine how much to to trust the social network. In simulation studies of SoTru have shown significant improvement of AREX in terms of adaptation speed. SoTru simulations also show that it can tolerate unreliable social networks where attackers have infiltrated the network.

CHAPTER 2

BACKGROUND

This chapter describes the relationship between trust and security in dynamic systems. The requirements necessitated by dynamic pervasive systems are discussed in this chapter. The work in this dissertation is motivated by examining the insufficiency of prior work for dynamic systems.

2.1 Challenges in Dynamic Systems

Developing and deploying dynamic systems poses challenges that involve mobility, device constraints (battery life, processing power, memory, etc.), heterogeneity, openness, and a goal of fading into the background of human experience. Dynamic systems share many properties with other types of systems. Specifically, dynamic systems include the problems of distributed, mobile, and pervasive systems and adds challenges in openness and churn (the turnover of nodes entering and exiting the system). This dissertation now discusses each of these challenges.

2.1.1 Mobility

Mobility in dynamic systems most commonly refers to mobility of the nodes in the system. While many underlying communication protocols have been designed to ease the burden of mobility, it can still cause disruptions in service and disruptions in security mechanisms. As a result, security mechanisms in dynamic systems must be able to protect the user against mobility-related vulnerabilities. Since mobility

(amongst other factors) can introduce intermittent connectivity, security mechanisms cannot require rigid connectivity structures such as a distributed hash table [17].

Mobility does not necessarily have to refer just to mobile devices, but can include mobile users or agents. Agents may move from device to device, even if the devices themselves are not moving, and still maintain the same security requirements. Also, to protect against a malicious mobile agent, other security mechanisms should not bind their protection only to the device. Likewise, users may use multiple devices in disparate locations. As users are less able to explicitly carry with them information to augment the trust process (such as access control lists or reputation information) as a software agent might be able to when it migrates, the security mechanism should be able to cope with this possibility.

2.1.2 Device Constraints

Device constraints in dynamic systems include typical sources of constraints such as energy, processor, memory, and bandwidth. Algorithms for these devices must be able to operate in a reasonable amount of time and not require so many resources that it affects other functions of the device. In dynamic systems, small mobile devices are often running applications. These devices are limited in performance and battery life. Any of these constraints can limit the performance of a system, so a security mechanism that requires too much of a particular resource, such as energy, will cause the application to become unusable.

In addition to typical constraint issues, devices in dynamic systems may suffer from constraints related to heterogeneity and invisibility. In particular, security mechanisms may use, but not require that devices have specialized hardware such as cameras, microphones, or even keyboards. While a security mechanism may make use of these features when available, it should not be required for secure operations.

2.1.3 Heterogeneity

Heterogeneity in dynamic systems includes a wide variety of factors such as operating systems, hardware availability, hardware architectures, network connections, network overlays, attacker strategies, and types of failures. As a result, security mechanisms must be able to execute in these operating environments. Security mechanisms should be easily portable between devices and not rely on specific aspects of the operating environment. Security mechanisms should also be able to address attacks such as individual attackers, colluding attackers, and sybil attacks (when a single attacker spawns many copies of an attacker) – whether the attacker controls a large portion of the system or a small portion. The mechanisms should also be able to operate when nodes are not actively attacking but fail occasionally. This means that if all nodes are potentially unreliable, the security mechanism should still permit the application to find necessary services.

2.1.4 Openness

Openness is the challenge caused by a low or non-existent admission cost into the system. In dynamic systems, nodes may freely enter the system without any cost imposed by a central controlling authority. As a result, approaches such as certificate authorities become difficult or impossible to use in dynamic systems. Openness also makes dynamic systems more susceptible to sybil attacks, so techniques such as relying on a majority vote become difficult or detrimental. Openness also means that security mechanisms cannot rely on banning entities from the system since no central control mechanism to do so exists and nodes can exit and reenter the system with different IDs.

2.1.5 Invisibility

Invisibility in dynamic systems is a goal shared with pervasive computing. The user should be able to take for granted that the security mechanism exists. The security mechanism should be autonomic, minimizing the amount of interaction required by the user, especially since the user may not be directly accessing the device (for example, a person walking down the street with a cell phone or PDA in his pocket may be performing actions requiring security, but it should not require the user's attention).

2.2 Security and Trust

Security in computing systems has traditionally focused on access control and authentication. In such systems, a user can only access a resource (for instance, edit a file) if that user has the appropriate rights to make the access, and has been authenticated as that user. This approach requires several characteristics of the system using it. First, the system must have a central authority. This is a simple requirement in a system such as a server with remote login – the server itself is the central authority. Second, it must be reasonable to authenticate the users of the system. This means that there must exist some secure mechanism that prevents the forgery of user accounts on the system. Additionally, the users must be known beforehand so that the system can assign them appropriate rights when they join or return to the system.

While these approaches can provide significant security benefits, they are not possible, at least not in their original forms, in many pervasive systems. As systems become more open and dynamic, supporting a rigid security infrastructure becomes increasingly difficult. Since the goal of pervasive computing is to create systems

that work so seamlessly that the user does not have to think about them, security mechanisms that require the user's time and effort are undesirable. A system in which the user must stop to create an account, make a request to an administrator to receive access rights, log in to the system with a username and password, and then access a resource does not follow the ideals of pervasive computing.

In addition to dynamic systems where many users may join or leave the system frequently, security mechanisms for pervasive systems must be able to handle characteristics such as mobility, heterogeneity, and interoperability.

To increase the flexibility of security systems, many researchers have now proposed trust-based approaches to security in distributed, and, in particular, pervasive systems. Trust can be used for security in many different ways and take many different forms. Some trust mechanisms more closely resemble traditional systems while others add a significant amount of flexibility to the system. As with many research areas in computer science, there are trade-offs associated with the various types of trust. Trust mechanisms that resemble traditional security mechanisms tend to be more rigid and require more infrastructure, but they also have the ability to provide stronger security guarantees. More flexible trust mechanisms often only mitigate some vulnerabilities in a system, and cannot entirely prevent the success of some classes of attacks. In general, the purpose of trust is to utilize relationships between entities to accomplish a task that would otherwise be more costly or impossible. These relationships may manifest themselves in many different ways.

2.2.1 Types of Trust

Trust can be established in many different ways. Some mechanisms require that trusted infrastructure be established prior to the assessment of trust, while others can be assessed in the absence of formal infrastructure. While many types of trust can

reduced to versions of each other, there are several well-established types of trust that are commonly used in the literature and in practice. This section discusses several well-established types of trust.

2.2.1.1 Authoritative

Trust is established through reliance on some authority to determine that the entity should be trusted. This is most often accomplished through a digitally signed certificate, such as a X.509 certificate [18], that is presented as evidence of trustworthiness. A defining feature of authoritative trust is that the authority produces a definitive decision as to whether or not an entity is to be trusted. Authoritative trust more closely resembles traditional security mechanisms than other types of trust. In such mechanisms, there is a central authority that verifies access rights and authenticates users [5]. The difference between an authoritative approach and that of traditional security mechanisms is that there does not need to be a central or coordinated authority. For example, several authorities may exist, perhaps each controlling an administrative domain, thus allowing for system extensibility and delegation of responsibilities. As a result, the authoritative approach still retains much of the security benefits of traditional security, but with additional flexibility.

2.2.1.2 Role-Based

In role-based systems, trust is established based on the role that an entity acts in. The role that an entity acts with can be established through a variety of means, including digitally signed certificates [7] or context aware mechanisms [19, 20]. The earliest research in role-based access control (RBAC) was in [21, 22]. It has since then been adapted to pervasive systems by utilizing trust in the context of active spaces [7]. Sets of permissions are assigned to each role in the system, and each user has a

set of roles. The roles are designed such that the permissions assigned to each role are only what are necessary to accomplish the tasks associate with that particular role. The advantage of using RBAC in pervasive systems is that it eliminates the need for individual user authentication. Each user only has to authenticate that they have a particular role. This eases the burden of administration after the initial establishment of roles.

2.2.1.3 Reputation

In reputation-based systems, trust is established based on the opinion of other entities. In reputation systems, other entities are queried for their opinions about resources or other entities. The results from these queries are then processed and a decision is then made based on the reported values. Reputation mechanisms permit a completely decentralized method of determining whom to trust. Reputation mechanisms tend to be used to determine whom to access resources from rather than who to allow access to. The advantage of this approach is that it makes the entry and exit of new users very easy as no single entity has to track their entrance and authenticate them. Many reputation mechanisms exist without any central authority or pre-established infrastructure (although some do exist that take advantage of centralization where it makes sense for the system [23]). The downside of reputation is that it is susceptible to many classes of attacks. As a result of the collaboration required for reputation-based trust, it is possible for attackers to control a significant portion of the system. Furthermore, nodes may collaborate to disrupt reputation mechanisms by having a set of nodes that attain high reputation and then report that other nodes controlled by the attacker should have high reputations. Reputation mechanisms are also vulnerable to "one time" attacks where a node acquires a high reputation then betrays the trust placed in it, or is compromised by an attacker who then uses the

high reputation to perform attacks. Thus, the attacker can attack without incurring the cost of building a reputation as a benign node.

2.2.1.4 Recommendation

In recommendation systems, trust is established through nodes' recommendations of other entities. Recommendation-based trust is similar to reputation-based trust and as a result share many of the same properties. The significant difference is that instead of querying entities for information about a particular entity, the initiating entity inquires about the most highly rated entity (or set of highly rated entities) to provide the desired service or resource. Rather than definitely relying on an authority, the entity queries other entities in the system to collect their recommendations, then processes the recommendations and makes a decision based on the reported values.

Table 2.1. Comparison of Trust Mechanisms

System	**Mobility & Connectivity**	**Open**	**Type**	**Central**
EigenTrust [17]	DHT prevents mobility	Yes	Rep	No
Credence [9]	Not designed for mobility	Yes	Rep	No
CORE [24]	Designed to handle mobility	Yes	Rep	No
PolicyMaker [25]	Requires connectivity	No	Auth	Yes
Keynote [26]	Requires connectivity	No	Auth	Yes
Vigil [5]	Connected to components	No	Auth	Yes
Gaia [7]	In service spaces	No	Auth	Yes
SECURE [6]	Designed for mobility	No	Rec	No
CTB [27]	Not Addressed	No	Auth	Yes
SAHARA [10]	Not Addressed	No	Auth	Yes
QUEST [28]	Designed to handle connectivity	No	Auth	Yes
JIA [29]	Designed to handle mobility	Yes	Rep	No

2.3 Existing Trust Systems in P2P

Much research has been done on the topic of trust in P2P systems. P2P communities often bring together large numbers of peers without requiring significant means of authentication or standards for entry into the system. Consequently P2P trust research has largely focused on reputation mechanisms. Existing trust mechanisms in P2P are of interest to this dissertation for their approaches to satisfy challenges of openness, heterogeneity, and sometimes mobility. A significant number of reputation systems have been designed, so only some of the more significant ones are described in this section.

2.3.1 EigenTrust

EigenTrust [17] establishes a reputation matrix that stores the experiences each peer has had with the other peers in the system. EigenTrust then uses the dominant eigenvector of a square reputation matrix to calculate a global trust value for each peer (where element i,j represents i's opinion of j). Trust is assumed to be transitive in order to perform this calculation. The reputation matrix is populated based on the number of positive transactions minus the number of negative transactions that peer i has with peer j. The values are then normalized such that each row sums to 1. The dominant eigenvector then gives the global trust value for each peer (and the probability with which a peer should choose to access resources from that peer). The authors then present variants of EigenTrust designed to operate in a decentralized manner (through distributed computation), and more securely (through managing the global reputation for each peer at several points in a DHT). The structured nature of the storage makes it difficult for EigenTrust to handle churn and mobility since the DHT would constantly have to be rebuilt.

2.3.2 Credence

Credence [9] is motivated by the observation that often trustworthy peers still propagate bad objects. Walsh, et al. approach the challenge that a peer sharing an object does not mean that the peer necessarily endorses that object. As a result, Credence operates on *object reputation* rather than peer reputation. Object reputation rates individual objects in the system rather than the peers that host them. Votes for or against the object (and in reality, components of the object) are weighted by the correlation of the voter. For instance, if A strongly disagrees with most of B's opinions, then a positive vote from B would be perceived as a negative vote by A. Credence communicates votes over the gnutella network, so its communications are handled in a decentralized and unstructured manner. Each vote is digitally signed so that it is not modified during the forwarding process.

2.3.3 PeerTrust

PeerTrust [2] is a Peer-to-Peer reputation-based trust system that also works in a client server architecture. While PeerTrust was designed with e-commerce application in mind, it will work as an overlay to a P2P network, so the concepts still fit with access control in smart spaces. The goal of this work is to produce an adaptable, effective, and generic trust metric that is resistant to corrupt feedback attacks.

The authors argue that problems with most current reputation mechanisms are that results rely solely on positive or negative feedback, they assume that agents are honest, there are no mechanisms for context sensitivity filters, there is no account for temporal adaptivity, and there are no incentives for peers to rate each other.

Furthermore, the authors list five important factors for evaluating trust.

- Feedback based on the amount of satisfaction
- Scope of the feedback

- The source's credibility
- Transaction Context (how important is the transaction)
- Community Context (how the transaction affects the community)

The trust factor is computed by combining these factors together. PeerTrust takes a weighted value of the sum of the products of the normalized satisfaction the peer receives from it's transaction, the credibility of the peer during a time period, and the transaction context factor, and normalizes that value by the number of transactions in the time period. The value is added to the weighted value of the adaptive community context factor. While context factors are specific to the application, this method of computing trust produces an average value of credible satisfaction which is modified based on the context of the community. By comparing this trust value to a set threshold, the computed trust value can be used to make confident decisions for access control.

Context factors allow for customization of the system to fit specific needs. For instance, cost or value of transaction can be incorporated into the equation by taking that as a context value of the transaction. Temporal adaptivity can be added by creating a weighting of past values. Incentives can be added as part of the community context. Thus, free riders can be punished by making the community context the ratio of resources provided to resources consumed.

2.4 Existing Trust Systems in MANETs

2.4.1 CORE

CORE [24] is a reputation mechanism designed for mobile ad-hoc networks. CORE uses three types of reputation and combines them together to produce a final reputation value. These three types are subjective reputation, indirect reputation,

and functional reputation. Subjective reputation is calculated directly from an entity's observations. Indirect reputation is reputation information reported by other entities. These values reported as indirect reputation can only take non-negative values as a defense to prevent nodes from propagating incorrect negative reputation. Functional reputation is the reputation an entity has for performing a specific function such as packet forwarding or routing.

2.4.2 Jiang, et al

Jiang, et al. [29] focus on reliable service composition in mobile ad-hoc networks (MANETs). The work is focused on minimizing MANET disruptions in the service composition process. The system is viewed in two tiers, a service layer and a network layer. A dynamic programming solution and a heuristic-based solution (to loosen the requirements of the dynamic programming solution) are presented to provide both network-level and service-level recoveries.

2.5 Existing Trust Systems in Pervasive Computing

Trust systems in pervasive computing tend to require more administrative control than systems in P2P and MANETs. Trust systems in pervasive computing tend to be less open and require a greater cost of administration. Some systems alleviate some of the administrative burden by allowing delegation of access rights. Typically these systems focus more on access control than on reliable and trustworthy access.

2.5.1 PolicyMaker & Keynote

PolicyMaker [25] is a distributed trust-management engine designed to handle authorization via compliance checking. A compliance checking algorithm in the system handles requests, credentials and policies in order to determine if the request is

allowed. Keynote [26] is designed with similar goals as PolicyMaker and maintains many of its features; however, it has two additional goals: standardization and ease of integration with applications. Keynote achieves this by placing responsibility on the compliance checker and standardizing the policy assertion language.

2.5.2 Gaia

As part of the Gaia project [7], researchers utilize mandatory and discretionary access policies, along with role-based access control to ease the process of administration and to provide security to their active space system.

The goals of this access control system are:

- Provide security to allow only authorized access to resources in both physical and virtual worlds
- Allow seamless transition between spaces without sacrificing security
- Ease of configuration, enforcement, and administration
- Support dynamic roles for explicit cooperation

To accomplish this, the authors present an access control model that supports customizable policies, virtual and physical world considerations in access control, and based on the user and activity set, dynamic support of different policies for different spaces.

The authors create a version of role-based access control that recognized three types of roles: system, space, and application. System roles are created with the user account and specify generic permissions for resources access. Space roles are dependent on which space they currently exist in and are used to access resources via the policies of the space administrator. The system automatically maps generic system roles into space roles. Finally, application roles permit even more customization by allowing the user to gain access based on the role of the task he is performing. These

roles are then mapped into space roles. From this point, the system has access rights for the user. Using both application and system mapping of roles into space roles prevents the user from gaining more access than merited by his system role, while still allowing necessary access to perform tasks.

To satisfy the need to explicitly allow cooperation, they also incorporate four space modes: individual, shared, supervised-use, and collaborative. Individual mode occurs when only one entity is in the space. This mode causes the system to act based completely on the space access control as defined by the space, application and, system roles. When a second entity enters the space, the system switches to shared mode. As a default, shared mode only allows actions that all users in the spaces are allowed (in other words, the intersection of the set of all allowed actions of the users). In order to reduce restrictions, the supervised-use mode allows a supervisor to perform actions that would not normally be allowed under shared mode. Finally, collaborative mode is designed for a trusted group where, if any entity can perform an action, then all can perform the action (the union of the allowed set of actions of each user).

The Gaia Access Control System works by the space administrator setting up the access policies for all of the space services. Users then authenticate themselves in order to gain entrance to the space and the system adapts it's mode based on the current users. Users then begin to interact with the space and each other and the system utilizes interceptors to check whether or not accesses to services conform to the access control policy.

2.5.3 Vigil

Vigil [5] is another role-based access control architecture. The architecture is designed primarily for smart spaces where clients have the potential to be dynamic

and mobile. Vigil contains six functional components: Service Manager, Communications Manager, Certificate Controller, Security Agent, Role Assignment Manager, and Clients.

The service manager handles service location by brokering between users and services that register with it. The communications manager is a gateway that translates between different protocols so the service manager can communicate with the services and users. The certificate controller handles authentication by generating and verifying X.509 certificates. The Role Assignment Manager contains a list of roles and rules for roles for the entities in the system. The Security Agent handles the access rights of users. The Clients are defined as both the users and services in the smart space system.

Upon entering the smart space, the client is authenticated and is assigned rights as based on his role. This role can change during the course of the interactions of the system. The user can gain more access rights by having those rights delegated to them by another user with the right to delegate. These delegations may or may not be constrained, meaning that it is possible to form chains of delegation which must be traced back through the path the delegation occurred in order for the security manager to verify the right to access the desired service.

Ontologies are used to provide a system of access based on the properties resulting from a user's role. Not only does the system use role hierarchies, but additional properties and constraints can be created in order to establish rights dynamically without having to create a new role.

2.5.4 SECURE

In the SECURE [6] project, trust is presented as part of an interaction mechanism for a self-organizing system. The authors note that trust satisfies Prigogine's [30] four requirements for a self-organizing system as such:

- Mutual Causality – The exchange of recommendations between agents affects the behavior of each agent
- Auto Catalysis – Positive or Negative recommendations will cause either an increase or decrease in interactions as a reaction
- Far-From Equilibrium – The system is dynamic and trust values are constantly changing, so the system will not reach and maintain an equilibrium
- Morphogenetic Changes – The services, users, and environment of the system are affected by random changes in condition.

The two main considerations of the model which is based on human behavior are communication through semantic information and decision-making with uncertainty. Tags are defined as the semantic information regarding a potential transaction between agents. The information in these tags are characteristics of the resource. For instance, a printer tag might include information such as pages per minute, color or black and white, and whether it has a duplexer or not. At the beginning of an interaction, tags are exchanged in order for both agents to learn about each other's capabilities. Then, based on previous experience, recommendations, risk, and the ability for the other agent to satisfy the user's needs, a decision to interact is made. Following an interaction, an evaluation takes place and the agents update their trust values with regard to each other. Through this process, recommendations propagate through a system to provide a distributed, trust-based method for access decisions.

2.6 Trust in Service Composition

Significant research has recently taken place in service composition. While much of the work in service composition has been focused on composing web-services, there has also been some work in the dynamic environments encountered in pervasive computing.

Composition Trust Bindings [27] verify the integrity of composed services using the service composition equivalent of digitally-signed software. A CTB policy requires that all nodes, services, and composition paths be known.

SAHARA [10] is a service composition framework that provides authorization control based on local rules and credentials from other domains. SAHARA also relies on a central Authentication, Authorizing, and Accounting (AAA) server and a requirement for credentials.

Bartoletti et al. [31] model service composition with security constraints using an extension of λ-calculus. This approach requires a statically-determined abstraction of service behavior which may not be feasible in dynamic systems. The system must be aware of what services will be available in an environment, the nodes that will be providing these services, and the method of composition until run-time.

QUEST [28] is a service composition infrastructure designed to assure QoS constraints. While it is not focused on security, it shares a common goal with this dissertation: to improve the reliability and performance of dynamic service composition. While QUEST does handle multiple nodes providing each lower-level service through a modified version of the Dijkstra algorithm [32] weighted with values of availability and response time, it does not address multiple different paths that could be used to achieve a compositions.

2.7 Observations of Prior Work

Most of the trust frameworks for dynamic systems described above require the distribution and checking of certificates against policies to establish trust. This approach has several significant drawbacks. First, it limits the use of trust in the system to that of certificate-based trust. Other types of trust exist in other areas of distributed computing that are useful in a dynamic systems, such as reputation-based [8, 17, 33, 34, 9] or recommendation-based [6], would be difficult or impossible to implement in these trust-management systems. Second, the type of trust is largely tied to the trust distribution mechanism. As a result, many components of the systems described have limited reusability, especially if the use of trust is not for authentication and access control, as is the case with most previous approaches.

Many of these trust mechanisms rely on some pre-existing list of trusted peers; however, it is not always possible to identify such peers in uncertain and potentially malicious systems. For example, in a file-sharing application in an urban setting, users may continuously come and go, making it difficult to identify peers that can be trusted in advance. In hostile environments, such pre-trusted peers could be captured and corrupted, so it can be dangerous to assign trust management tasks to only a few nodes. Without reliable pre-trusted peers, there is no guarantee that reputation values provided by any peer are legitimate. Consequently, trust mechanisms that accumulate the preferences of the majority of peers to calculate reputations will fail to provide protection for benign peers when they are in the minority.

Since reputation mechanisms are cooperative, they require communication among peers. In systems where consistent connectivity cannot be assumed, such as a mobile P2P system, a reputation mechanism will degrade in effectiveness when the portion of the system available to communicate with at any given time decreases [17]. The reduced ability to acquire reputation information may result in less reliable reputation

results. The unreliability of reputation in these cases is caused because changes in reputation values that otherwise would have propagated quickly through the system now take longer, so peers make decisions based on degraded reputation information. Additionally, attackers may take advantage of wireless, peer-routed communication to selectively disrupt communications, thereby taking control of the propagation of reputation information.

Another limitation of existing approaches to reputation-based security in existing P2P systems is the requirement of prior experience to make decisions [2]. As a result, peers are vulnerable against attacks when they enter the network. In a foreign system with no known trusted peers, an entering peer is vulnerable to attack as it has no means to determine the trustworthiness of any other peers in the system. This fact can be exploited by an individual malicious peer or by a set of collaborating peers. Furthermore, a peer can initially behave benignly, be recognized as such, and then act maliciously (either intentionally or due to being compromised). These attacks are especially dangerous for a peer that is sensitive to attacks (or to a particular type of attack) and reputation does little to prevent such attacks.

CHAPTER 3

TRUST FRAMEWORK

3.1 Introduction

Dynamic systems are characterized by dynamicities in terms user/device mobility, resource constraints, and high rate of churn. On the other hand, establishing trust among users and creating reliable services are paramount to meeting application requirements. This problem is further exasperated by the lack of flexibility, modularity, and scalability in existing trust mechanisms. The Distributed Trust Toolkit (DTT) described in this chapter envisages to address this challenge. Even though pervasive computing environments are considered in this chapter, the framework described here is applicable to dynamic systems in general.

Closed pervasive environments such as smart homes, banks, laboratories, clinics, vehicles, and personal area networks have important differences that prevent monolithic security solutions from being adopted universally. For example, the pervasive system in a public bus is completely different from that in an assisted-living home in terms of challenges, services, privacy, number of users, etc. Even within the same physical environment, applications may have different requirements. For example, a gaming application service and a video surveillance service within a public bus have different challenges and issues. Despite these differences, it is desirable to maximize the amount of work that can be reused when deploying such a diverse set of systems.

A popular approach to addressing security challenges in these environments is the use of distributed trust mechanisms. Trust allows users to use previously gathered information, such as certificates or reputation scores, to interact with peers in the

system with some assurance. However, most trust mechanisms currently available for pervasive computing lack flexibility, modularity and scalability. There is a need to develop mechanisms that are adaptable to mobility and disconnection, portable to different pervasive systems, and scalable with respect to number of devices, users and applications. The proposed DTT envisages to address these critical issues to make trust management more usable and easily extensible to a wide variety of pervasive environments.

The DTT is a set of tools that enables the implementation, deployment, and sharing of trust-related components for pervasive systems. The primary component type is called a *Trust Block*, and it contains everything needed for an application to use trust-based decision-making. Thus, the DTT has only two programming interfaces: an Application Programmer Interface (API), which facilitates the development of applications using Trust Blocks, and a Trust Block Programmer Interface (TPI), which facilitates the development of the Trust Block modules themselves. Additionally, the DTT includes a DTT daemon that manages the Trust Blocks. As illustrated in Figure 3.1, applications may use a variety of remote interfaces, such as our XML-RPC interface, to connect to the daemon, which maintains local copies of Trust Blocks and any data the Trust Blocks have stored. Also, a node may have several applications that access a daemon. Each application may make use of one or more Trust Blocks. Each Trust block may acquire trust information from one or more network interfaces that provide a common interface for portability purposes. A Trust Block may additionally query the local trust database for previously cached trust information.

While the DTT works for any distributed system, it is particularly designed for pervasive systems. The DTT uses Trust Groups to improve the performance of existing algorithms by utilizing a users set of devices such as cell phones and PDAs. Furthermore, DTT is designed to mask heterogeneity and ease porting to new devices

and networks through a set of simple, generic interfaces. Finally, the DTT is designed to ease the development of trust mechanisms for new environments through a modular approach to the design of trust mechanisms (refer to Section 3.7.2 for an example of transition Credence from an internet-based implementation to one more suitable for a MANET by only changing the Protocol component).

There are three goals of the DTT: enabling trust sharing, encouraging algorithm reuse, and easing trust system deployment. To meet our first goal of facilitating the sharing of trust information, we developed Trust Groups. Trust Groups are a natural consequence of the DTT design, as the nodes in a group will share a single DTT daemon. This makes it easier to share trust information, by simply storing it in a single host. The DTT daemon thereby acts similarly to *super-nodes* in P2P file-sharing systems like Kazaa [35], except that the DTT daemon serves trust information instead of directory information. Trust information can also be shared within a single node, since multiple applications running on a single entity can utilize a single instance of the DTT. For instance, consider a laptop that is part of both a Smart Home environment and part of the Adaptive Media System . Since these two applications may utilize resources from overlapping sets of nodes, trust information can be shared between applications.

To meet our second goal of encouraging code sharing and reuse, we have two design features. First, the use of Trust Blocks creates a clearly defined module structure for which the API can provide a generic interface. This means that applications need not change much to use different Trust Blocks. Second, the TPI is designed to facilitate composition of modules within Trust Blocks. Each Trust Block is comprised of three functional components, each of which may be mixed-and-matched with other components to encourage sharing of code.

Finally, to meet our third goal of facilitating deployment and use of all trust mechanisms, we have taken measures to ensure platform-independence and interoperability. In particular, we chose to make the DTT daemon cross-platform. We also ensure that the DTT may interoperate with other platforms not using the DTT and that different Trust Blocks can be used at the same time. Interoperability means that DTT and various Trust Blocks can be deployed incrementally without disrupting the ongoing operations of a system.

In the rest of this section we will describe the design of Trust Blocks, the core component of our system, the design of Trust Groups, and how the DTT ensures platform-independence and interoperability.

3.2 Operating Environment

Traditional approaches to security management and deployment typically fail to scale to the varied, mutable environments of typical pervasive computing systems. Instead, pervasive computing environments must rely on distributed trust [5, 7, 8, 17, 25]. While these distributed trust solutions scale usefully in pervasive environments, the infrastructure of these systems is tied to the single, specific trust mechanism explored by the researchers. This makes it difficult to reuse, combine, and extend trust mechanisms in custom systems or protocols, thereby limiting both research in trust and the use of trust in real systems. In our examples, we use trust to establish secure resource access, but the Distributed Trust Toolkit (DTT) is not limited to that usage.

Pervasive applications using the DTT connect to a *DTT daemon* that manages trust for the application. The DTT daemon contains a set of pluggable *Trust Blocks*, each of which provides an implementation of some trust mechanism. Trust Blocks

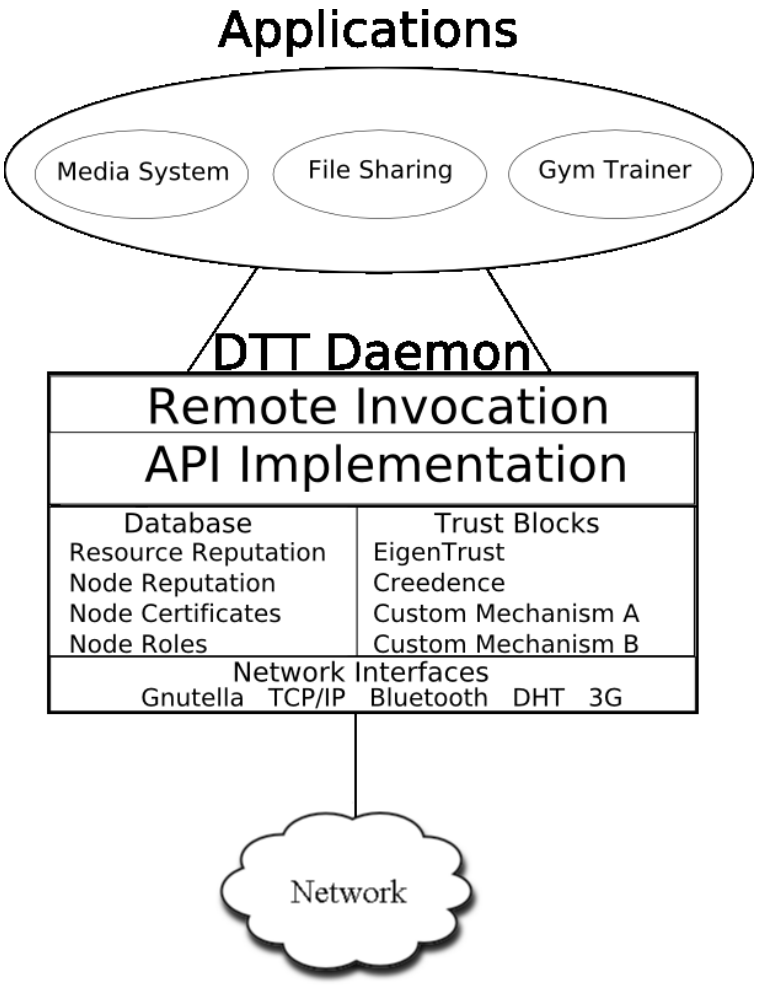

Figure 3.1. DTT Architecture.

are modular and may inherit individual modules from other Trust Blocks, leading to easy customization of trust mechanisms and improved code reuse.

3.2.1 Example

Suppose that Alice, a computer science student at a university, wishes to deploy an adaptive media system. Her users are other students with laptops and smartphones that move around the campus, yet expect their media streams to follow them. In

testing, Alice finds that students are often unhappy with her application because many of the data sources that her application chooses are slow, unreliable, or stream the wrong media. The users' mobility compounds the problem because the system must continually choose new sources without full knowledge of how trustworthy or reliable the sources are. Furthermore, Alice wishes to make use of the heterogeneous nature of her environment and utilize local networks formed by connections using local 802.11 and bluetooth in addition to internet and 3G connections to discover new trust information.

To remedy this problem, Alice uses the DTT to enable her system to choose better, trusted sources. First, she chooses a Trust Block that defines her trust policies. Initially, Alice decides to trust only sources that have X.509 certificates signed by her university's certificate authority. To implement this decision, she modifies the initialization code of her application to instantiate a CertificateTrustBlock. Later, in her discovery code, Alice only includes sources that are trusted by her Trust Block:

```
if (tb.isTrusted(source)) {
    validSources.add(source);
}
```

At runtime, the Trust Block contacts a discovered DTT daemon and takes care of the certificate exchange protocol between her application and the source. As network interfaces for her protocols are well-established, the Trust Blocks do not have to be modified to access the certificates through any of the networks her users have available at any time.

When Alice deploys the new DTT-based application, her users are thrilled with the reliability of the sources, but dissatisfied by the small number of "certified" sources — mostly official university lectures and seminars. To satisfy her users, Alice decides to include a new Trust Block based on a distributed reputation system, based on

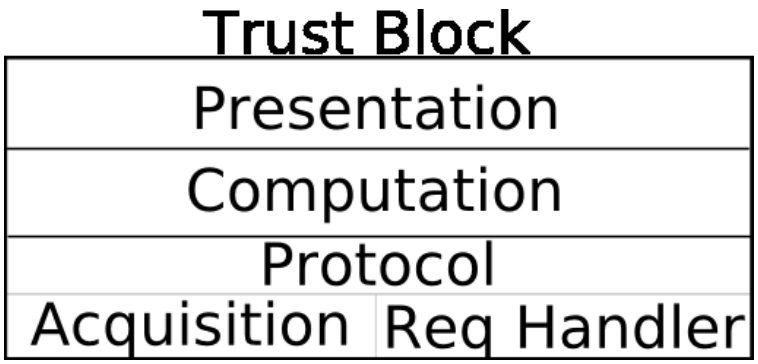

Figure 3.2. The Components of a Trust Block.

Credence [36], rather than certificates. To make the change, Alice simply instantiates a CredenceReputationTrustBlock rather than a CertificateTrustBlock and redistributes her application. At the application-level, Alice does not need to change her discovery code because the CredenceReputationTrustBlock defines a new policy for isTrusted() that her application can just use.

3.3 Trust Blocks

Each Trust Block consists of three layered components: the Protocol, the Computation, and the Presentation. Figure 3.2 illustrates the relationship between the components. An application accesses the Trust Block through its Presentation component. The Presentation component, in turn presents the trust information determined by the Computation component. Finally, at the lowest layer, the Protocol component interacts with other entities on the network to acquire the data that the Computation component needs.

The API provides access to a set of mixins and more complex Trust Blocks. The API can either be accessed as a local library or through remote invocation. The DTT mixins provide basic trust functionality that can be used for simple scenarios

without having to create more complex mechanisms. The mixin functionality consists of isReputable, isRecommended, isAllowedRole, and isCertified which provide reputation, recommendation, role-based decisions, and X.509 certification. Furthermore, the API allows access to specific Trust Blocks that are implemented with a common interface. This interface uses the eval method to initiate an evaluation of a user or resource and the getResults method to retrieve the up to date results of the evaluation for a particular evaluation request. The result of a request is of type TBPresentation which is discussed in Section 3.3.1.

3.3.1 Presentation

The Presentation component is the "application-facing" component of a Trust Block. It is responsible for implementing policy decisions — e.g., whether a certificate is "valid" or an object reputable enough — based on trust data from the Computation component. For example, in a reputation-based system, one Presentation component might report only the "most reputable" object, while another might report a list of the most reputable objects. In a certificate-based system, the Presentation component may ignore the expiration date of a certificate when looking at old data. In our scenario of Section 3.2.1, the tb.isTrusted() function is implemented by the Presentation component. The Presentation layer is connected to the Computation component through the getResult method and a Presentation object is returned by the API to the application.

3.3.2 Computation

The Computation component is responsible for implementing the algorithms involved in computing the trust values that will be interpreted by the Presentation component. For example, a simple reputation computation might count the votes for

the object it is evaluating to determine the object's reputation score. Likewise, an X.509 certificate [18] computation mechanism might verify that the certificate was signed by a trusted Certificate Authority, that it was within the valid time frame of the certificate, and that the certificate had not been revoked. However, in no case does the Computation component make a policy decision on the data it evaluates. This policy decision-making is left to the Presentation component. Computation components connect to the Presentation layer through the doComputation method and to the Acquisition component through an acquireData method. The Computation component can also acquire cached trust information from the database. The information is tagged based on type (ie, reputation or a certificate) and subject (ie, a particular user or type of resource). This is discussed further in Section 3.6.

3.3.3 Communication Protocol

The Trust Block Protocol component is the "network-facing" component of a Trust Block. The Protocol component is divided into two sub-components: an Acquisition module that makes outgoing requests to other entities on behalf of the Computation component through the getData method, and a Request Handler that handles protocol messages from other entities in the system. This split was created to allow the developer to customize the way the Trust Block acquires data without changing the Trust Block's interaction with other entities in network. For example, a broadcast Acquisition component may simply use the underlying network broadcast to make requests whereas a more complex Acquisition module might maintain a list of preferred peers to query for trust information. In either case, trust protocol messages received from other hosts are unaffected by the decisions made in the Acquisition module.

The Protocol components are implemented using the TPI. The TPI's interface to the network adapter consists of broadcast, multicast, and query along with forwarding versions each method as additional logic may be required for a forward that is not required in initiating a request, such as checking the TTL of a message. Every network interface implements its version of these functionalities for the particular protocol. The TPI also provides an interface for receiving and responding to messages from the network. The getMessage and sendMessage methods are implemented to receive incoming queries and to respond to the appropriate node as a result of the queries. As an example, a trust block designed for Gnutella over the Internet could be dropped into a local adhoc network that communicates over TCP/IP and utilize the network-layer broadcast to acquire trust information and unicast to respond to the query as implemented in the network interface in the DTT Daemon. As a result, none of the code for the trust block needs to change despite the fact that the application is no longer running on an overlay network.

3.4 Trust Information

Trust information is the evidence held by a node that another node should or should not be trusted. It may exist in many forms such as a digitally signed certificate or a reputation value. Trust information is acquired through the communication protocol and personal experience. Then it is stored in the trust database where it can later be used by trust mechanisms to make decisions.

3.4.1 Trust Database

The trust information stored in the database is tagged with a type (e.g., an X.509 certificate or a User Recommendation). The tags allow DTT Members to retrieve any trust information that they can utilize from the existing cache of trust

information stored in the database. As some Trust Blocks may use similar, but not exactly the same trust information, some additional trust information may be used from the database if the Trust Block has the means to convert it into a usable form. For example, if a Trust Block collects boolean reputation values (*trusted* or *untrusted*), but there were votes from another Trust Block that were in the continuous range of [0,1], then the boolean reputation system might translate any vote with a value under 0.25 to count as *untrusted* and any vote over 0.75 to count as *trusted* (ignoring any votes in between as being too noisy). There is no assumption that this trust information is transitive, it is only made available to assist the entities in quickly coming to a decision if the trust mechanism chooses to use the information.

3.5 Interoperability

3.5.1 Independence

The DTT is designed to achieve several types of platform independence. The DTT's implementation is largely language independent. Our implementation of the DTT is written in Java to increase its independence from specific operating systems and hardware. The DTT Trust Blocks are also independent of the network that they acquire information from.

The DTT daemon is not tied specifically to Java, and additional implementations can be written in other languages. Likewise, we have implemented our Trust Blocks in Java, but the Trust Blocks can be implemented in other languages. It is also possible to execute Trust Blocks written in one language with a DTT written in another, given appropriate wrappers or exported methods.

To achieve independence from the network from which trust information is acquired, the DTT uses two interfaces, a Net Requester and a Net Responder. These

two interfaces form the core of the Trust Programmer Interface (TPI), and they are implemented to hide the details of the underlying network from the creator of a Trust Block. This means that Trust Blocks are portable to any network for which there is an implementation of the Net Requester and Net Responder interfaces available for the Trust Block's Protocol components to access. For instance, if a reputation Trust Block originally acquired trust information from a Gnutella P2P network on the Internet, the same Trust Block could be used to determine reputations on a local bluetooth network. A Trust Block thus can have a collection of Net Requesters and Net Responders that it uses to acquire trust information. This allows the Trust Block to acquire information from multiple networks if available or necessary.

3.5.2 Incremental Deployment

Since the DTT is a client-side library with pluggable Protocol components, the DTT can be incrementally deployed into a pervasive computing environment already running existing communication or distributed trust protocols as long as the appropriate Protocol components are loaded into the DTT daemon. Moreover, if the DTT is widely deployed, the DTT may aid in protocol upgrades, as a single DTT can engage in multiple versions of a protocol.

3.6 Trust Groups

The DTT design encourages the formation of Trust Groups. The main purpose of a Trust Group is to enable peers to share trust information and enhance performance in calculating trust values.

Trust Groups consist of a *DTT Host*, which runs a DTT daemon, and several *DTT Members*. Trust information is stored in a database on the DTT Host and can be used by any member attached to the Host. Since the Members attached to the

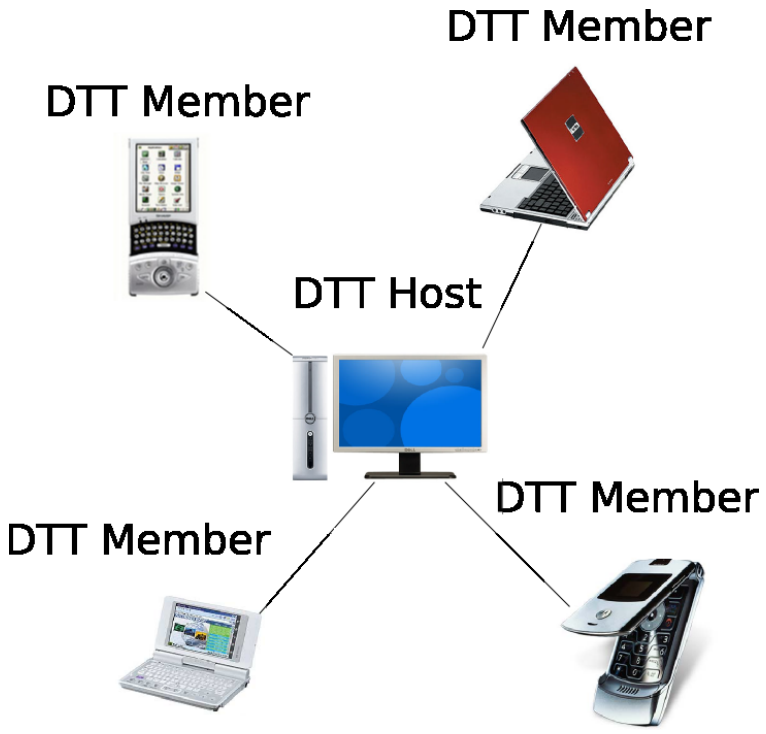

Figure 3.3. Devices in a Personal Area Network Using a Trust Group.

Host may be running several different applications that utilize different Trust Blocks. In Section 5.3, we evaluate the effect of shared trust information in an adaptive media system using a Trust Block that implements the Credence reputation system.

Trust Groups in our simulations are formed out of pre-trusted entities, such as the Personal Area Network (PAN) shown in Figure 3.3. However, we also permit the dynamic formation of Trust Groups. These groups can be formed opportunistically using individual DTT daemons to bootstrap into a Trust Group based on both mutual trust and the expectation that they will benefit from joining the group. Furthermore, since entities in pervasive computing environments may either sleep or move, groups may morph over time. In such a case, the most stable and powerful entities should host the DTT daemon for the group. In dynamic Trust Group formation, each entity is initially a host. However, an entity can discover other hosts in the area, establish a

level of trust with one of them, and attach as a member of the host's Trust Group. If a host breaks its connection from the Trust Group for any reason, each disconnected member will respawn a DTT daemon and can either seek to find another host to attach to or act as its own host. Trust information stored by the original host can be transfered in a graceful exit or some saved locally by members of the trust group in the event of a random failure by the host. In this way, Trust Groups are flexible enough to be deployed in a variety of pervasive computing environments. The optimal methods for determining which hosts to attach to and what constitutes necessary trust for doing so are largely application- and system-dependent and, as a result, outside the scope of this paper; however, many of these issues could be addressed with a scheme similar to Seamless Service Composition (SeSCo) [37] in dynamic environments or the Utility Based Clustering Architecture (UBCA) [38] for decentralized clustering.

3.7 Results

To show the value of DTT as a toolkit for research and to demonstrate the value of Trust Groups, we have simulated a sample pervasive computing environment with an implementation of the Credence reputation system [36]. Credence is an object-reputation system designed to operate on the Gnutella network. A peer running Credence accumulates votes about a particular object (identified by a hash of the file contents) via a Gnutella broadcast. The peer maintains a database of accumulated votes and the votes of the peer making the request. It then computes a correlation coefficient between itself and each peer it has a vote from. The coefficient is used to weight the the vote the peer has received from that particular peer. The total weighted result is then used to determine if the object is reputable and should be downloaded.

Table 3.1. Default simulation parameters

Number of Peers	50
Number of Media Objects	4000
Number of Genres	20
Number of Genres of Interest	4
Number of Groups	5
Group Similarity	Random
Peer Rating Accuracy	90%
Portion of Bad Media Objects	50%
New Media Objects per Day	16
Simulation Days	100
Average Number of Connections	5
Average Requests per Day	5
Media Object Density	80%

The use of Credence has three important benefits. First, it serves as a proof of concept to show that prior work in trust mechanisms can be implemented within DTT. Second, we have selected Credence in particular because it is both deployed as a real system [9] and as a simulated system [36], which makes it both practical and easy to compare to. Finally, it enables us to see the benefits of using Trust Groups to enhance the performance of an established system.

3.7.1 Simulation Setup

The simulation portrays an adaptive media system similar to that described in Section 3.2.1. In this system, entities make requests for media, which is discovered and linked together by the planner. To improve the quality of the media that is being supplied, the system uses Credence to determine the most reputable media sources. Each day, every entity requests an average of five media objects (uniformly distributed between 0 and 10 requests). The media requests are broken into separate genres that each entity prefers. Example genres may include “Action Movies” or “Jazz Music.”

As the simulation resulted in similar results in terms of the ability to identify bad media objects to those shown in the Credence simulations [36], we focus primarily on the reduced cost to achieve these results that is accomplished by using Trust Groups. For convenience, Table 3.1 contains the values used for simulations unless otherwise noted. Messages are tallied as individual votes and requests that are sent. This means that aggregate messages (such as reporting several votes at once) still count as the number of votes that are sent. We define a *trust message* as initial requests, forwarded requests, and responses, and count the overhead in terms of the number of trust messages. For the simulations with Trust Groups, the trust message count also includes all votes cast, since the DTT may not be running locally. Each data point represents the average over 100 runs of the simulation. The current implementation of the DTT, on which these simulations were run, is written in Java.

3.7.2 Simulation Results

Our first simulations were designed to demonstrate the effect of increasing the number of peers in the system. Figure 3.4 shows the effect of increasing the number of peers in the system in terms of trust messages sent per peer on both Credence and Credence implemented with Trust Groups. Furthermore, the simulation also includes the effect on peers attached to the groups. As attached entities are often resource-constrained, such as a PDA being used for mobile viewing of a movie, it is important that they spend less energy transmitting messages for computing trust. The decrease in trust messages sent by each peer in Credence occurs because the resources (and hence votes) become more abundant, so trust queries take less forwards to find new votes. Likewise, the DTT peers see the same effect, but by utilizing DTT trust groups the effect is more pronounced since votes are shared at a DTT, thus requiring less communication between DTT nodes. The average DTT peer and the average attached

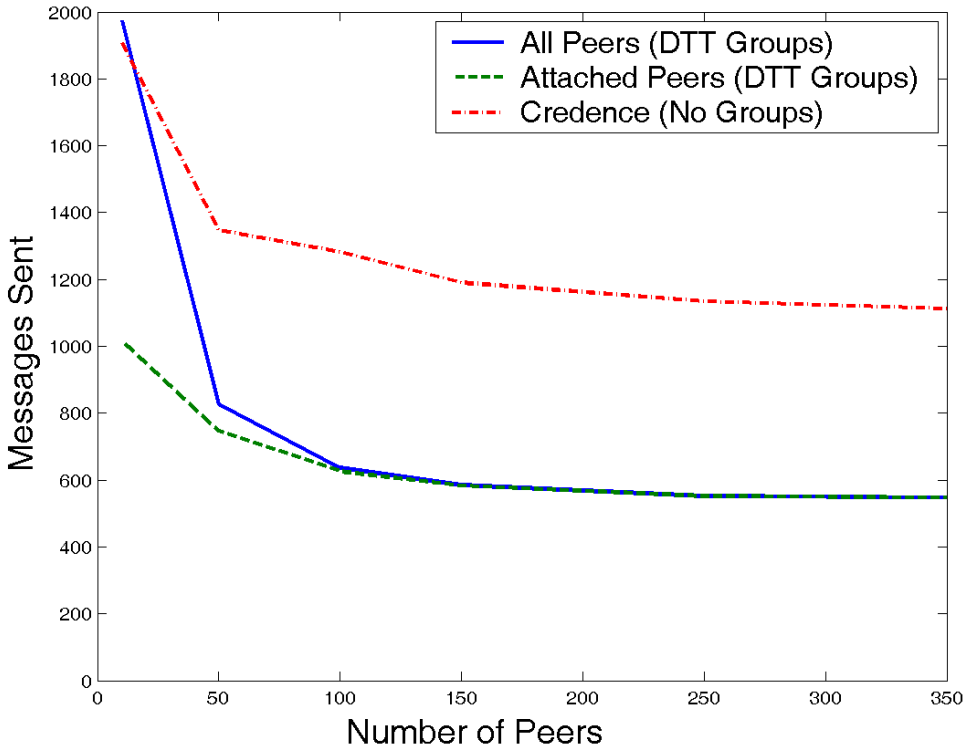

Figure 3.4. The Effect of the Number of Peers on Trust Messages per Peer.

peer eventually converge because the groups become large enough that most of the votes that need to be acquired can be found within the DTT group. At that point, the number of messages sent is approximately the same as the number of requests made plus the number of votes cast.

Continuing with our examination of modular Trust Blocks, Figure 3.5 shows the results of changing the acquisition component used in the simulation. The Broadcast plot is the result of acquiring trust information for Credence via a Broadcast over a Gnutella network with a TTL of 5 hops. The Random Walker plot is the result of plugging in a Random Walker Acquisition component on the same Gnutella network.

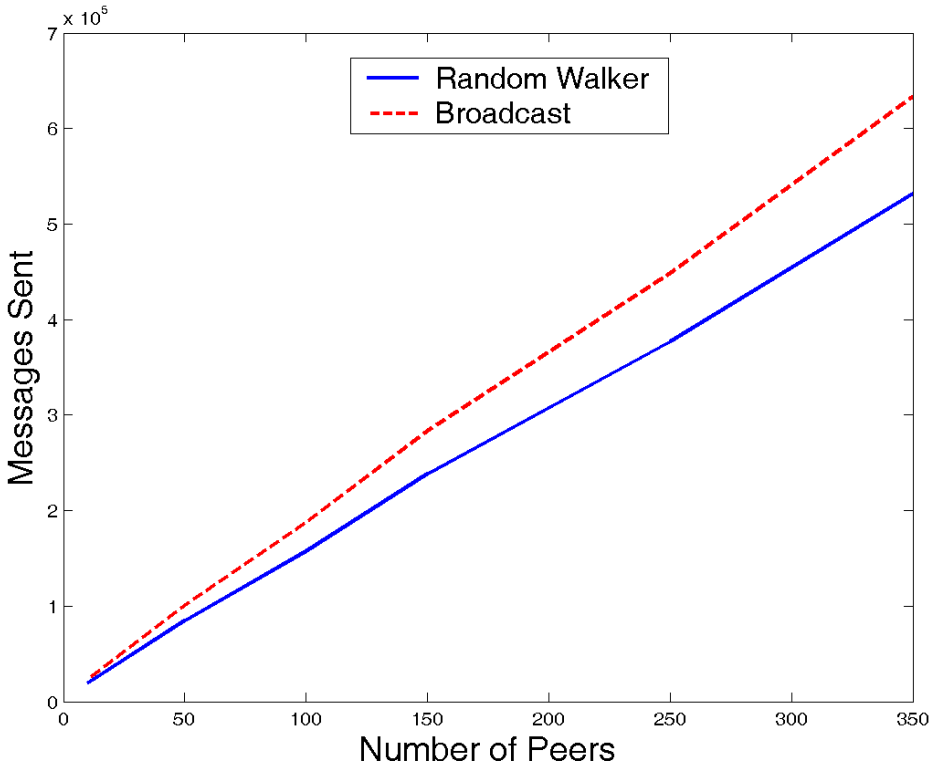

Figure 3.5. Comparison of Acquisition Components.

We examine the energy consumption of resource-constrained members of a Trust Group when they are able to enter sleep-mode once they have offloaded a trust request to the DTT Host, rather than waiting for responses as they would need to do if not in the Trust Group. The energy simulation assumes a cost of 1 energy unit per time step to stay awake and 0.1 energy units per time unit to sleep. Furthermore, a sent message costs 1 additional energy unit. Each non-Host entity sleeps for the 5 time steps after the trust request. Each resource access takes one time step. The result of this simulation is presented in Figure 3.6. The dominating portion of the energy consumption in this simulation setup comes from the messages sent. In scenarios where

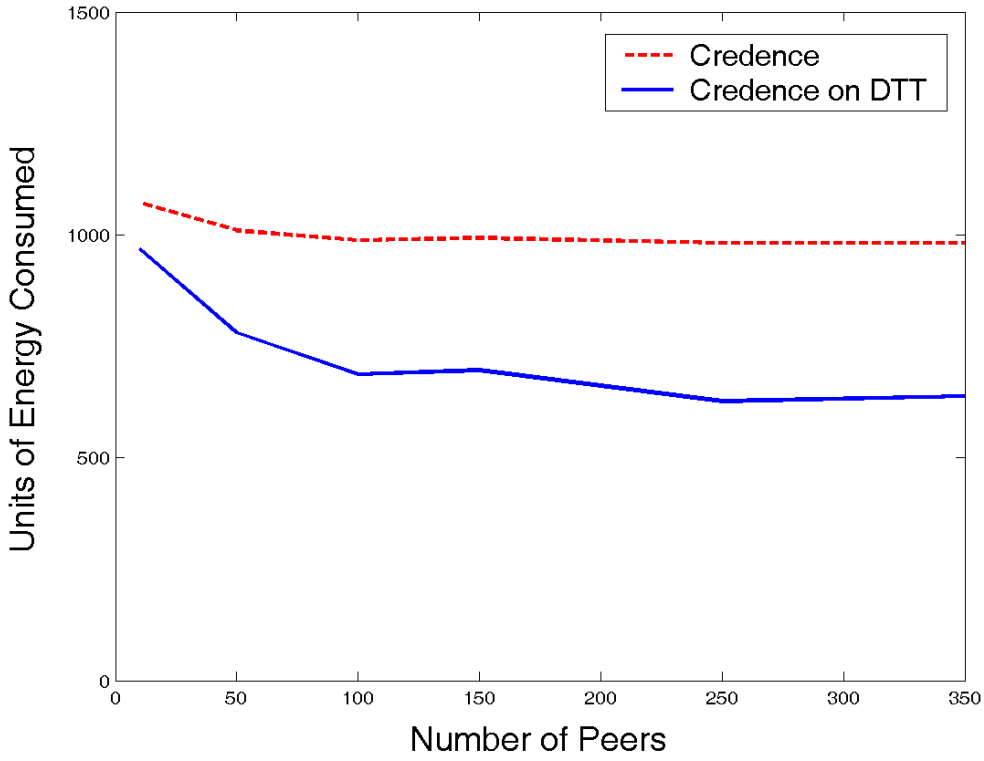

Figure 3.6. Energy Consumption per Peer.

less messages are sent relative to time in the system, for example, an opportunistic networking application, then benefits of the DTT become even more pronounced.

Our last simulation examines the modular Trust Block feature of the DTT. To do this, we break from our previous simulation scenario. We take a simple average vote Trust Block (A peer asks for votes on the reliability of a particular object and if average vote rates the object above 0.50 then it is accessed, otherwise it is not). Votes are in the continuous range of [0,1]. Additionally, this is performed in a mobile environment, which decreases the number of available votes.

Since we have fewer votes, we are interested in determining the average range of the 95% confidence interval over the average of the requests (using the same param-

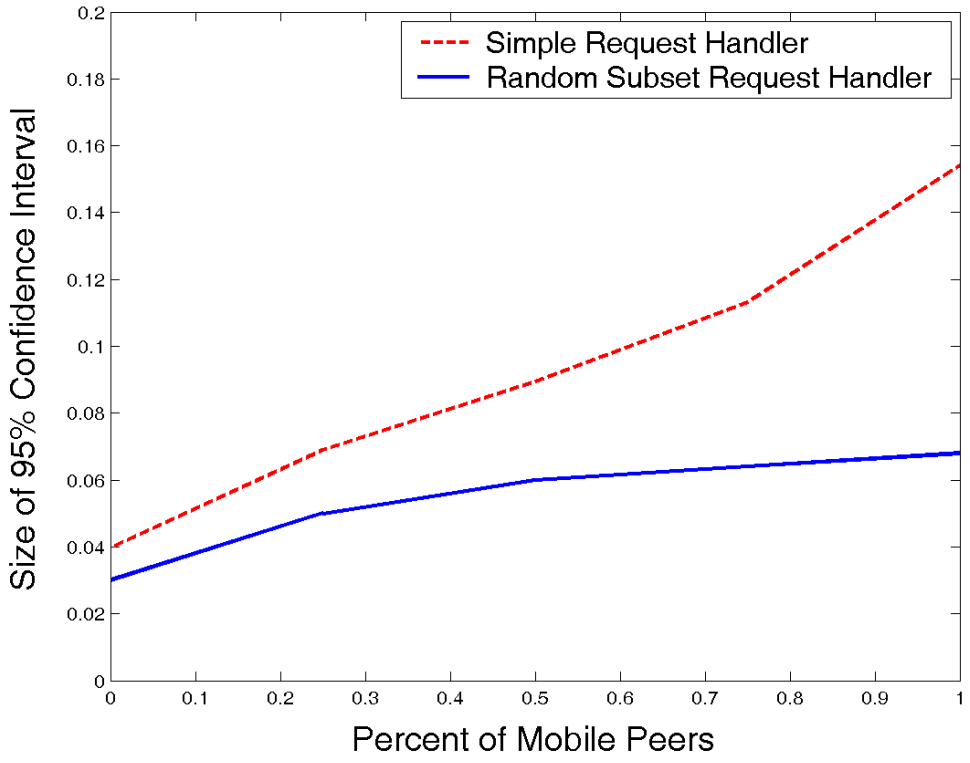

Figure 3.7. Effect of Protocol Components on Confidence Intervals.

eters in Table 3.1). To obtain the confidence results in Figure 3.7, we first examined the original Request Handler component, then exchanged it with the Random Subset Request Handler components. In the simplistic Request Handler, requests were only answered with personal experience; however, in the Random Subset Request Handler, the handler answers requests with a random subset of the votes that the entity has accumulated regarding the object. The trade-off between the two Request Handler components is that the simplistic one sends less data over the network, and the Random Subset handler allows votes to persist even after mobile peers have become unavailable.

3.8 Summary

The Distributed Trust Toolkit provides a ready-made framework for implementing and evaluating trust mechanisms in pervasive computing systems. The DTT introduces two new abstractions: Trust Groups and Trust Blocks. Trust Groups allow associated applications devices to share recorded trust data and trust computations. Trust Blocks divide the implementation of trust mechanisms into three modular components: the Presentation, Computation, and Protocol. Applications interact with the Trust Block Presentation, which makes policy decisions based on data gathered by the Computation component. The Protocol component implements network-based trust protocols and allows the DTT to interoperate with legacy trust systems.

The pluggable Trust Blocks of the DTT framework enable the use of trust mechanisms developed for developed for different systems and requirements. The remainder of this dissertation discusses three novel trust mechanisms, AREX, ReSCo, and SoTru, each designed to meet specific goals and requirements. Any or a combination of these trust mechanisms can plugged into a DTT framework to evaluate trust.

CHAPTER 4

ADAPTIVE RESOURCE EXPLORATION

4.1 Introduction

In dynamic systems it is often difficult to establish the reliability of a third party to assist in the establishment of trust between two parties. Because these systems often lack intrinsically trusted infrastructure and the open nature of the system makes collaborative techniques for establishing trust difficult, techniques must be established to establish trust without the requirement of third parties. This chapter introduces trust mechanisms that do not rely on reliable or honest third parties to establish trust. These mechanisms can be augmented with collaborative or centralized mechanisms if it is discovered that such techniques can be securely used, but they do not require such mechanisms to operate securely.

4.1.1 Motivating Scenario

Consider a user, Bob, who arrives at a new city. Bob does not know his way around this new city, but is able to retrieve some maps and use a mapping service that he discovers are hosted by other users on the ad-hoc network. In querying these resources and services, Bob quickly discovers that the information he has access to is not consistent. As there are several different sets of information that are offered to him multiple times, Bob cannot reliably select the most commonly provided information since it is obvious that the majority of the system may be providing incorrect information. He does not have any basis for extracting the truth from these resources

and services. This chapter introduces mechanisms that Bob could utilize to more securely and reliably access the resources he needs to navigate through this new city.

4.2 Resource Exploration

This section provides the details of Resources Exploration, a technique that is used to detect potential attackers or unreliable nodes. Resource Exploration can also, when used strategically, motivate malicious nodes to provide benign resources at a higher rate that they otherwise would.

4.2.1 Nash Equilibrium Overview

A Nash equilibrium[39, 40] is a solution to a multiplayer game in terms of a set of strategies for the players partaking in the game. The Nash equilibrium solution implies that no player in the game can improve its utility by changing its strategy whilst the other players continue playing their strategies. A game may have multiple Nash equilibria.

In order to determine a Nash equilibrium, the parameters of the game must be set as shown in Figure 4.1. By setting the expected value of each action a peer could take equal to the alternative action, the mixed-strategy equilibrium can be determined. As a result, the serving peer should attack with a probability defined by Equation 4.2 and the requesting peer should use exploratory messages with a probability defined by Equation 4.1.

$$P_{exp} = \frac{B_m}{C_{dis} + B_m} \tag{4.1}$$

$$P_{attack} = \frac{B_b}{C_v + B_b} \tag{4.2}$$

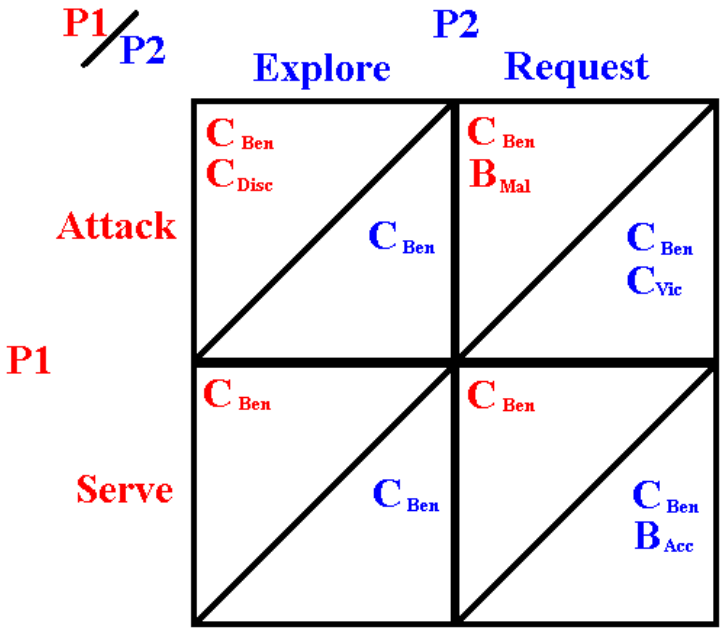

Figure 4.1. Payoff Matrix for a Malicious Peer (P1) and a Benign Peer (P2).

An obvious downside to this approach is that it requires a knowledge of the opponent's preferences.

4.2.2 Attack Minimization

The utility of a benign peer and that of a malicious peer are modeled by Equations 4.3 and 4.4 respectively where P_{exp} is the probability of sending an exploratory message.

$$U_{benign} = Access - \frac{C_b}{1 - P_{exp}} - (1 - P_{exp}) \times C_v \tag{4.3}$$

$$U_{attacker} = (1 - P_{exp}) \times B_m - \frac{C_m}{1 - P_{exp}} - C_b \tag{4.4}$$

These equations provide the basis for our next Section. By selecting a P_{exp} such that $Utility_{benign}$ is non-negative, a peer minimizes the amount of attacks suffered while still expecting to not lose utility by participating.

4.2.3 Utility Model

To measure the performance of the security mechanism, performance metrics must be defined. To do this, a utility model for peers is defined. The utility model allows us to model a variety of peers analytically and in simulation. The model also provides a means for conveying the effect of our mechanisms on the variety of peers, so that performance can be validated.

The relationships between benefits and costs defines the behavior of a generic peer inn the system. Generic peers are not required to be purely malicious or purely benign. Instead, a peer's actions will be evaluated based on its utility function. The following terms are used in the relationships.

The utility function is normalized to unit-less, non-negative values. Since most of the low-level components that make up the utility relationships are preferences, such as an aversion to being subjected to a denial of service attack, no formal method is provided for determining the values of those costs and benefits, though in many cases these benefits and costs could be described financially or in terms of energy costs.

The Victim Cost, C_{vic}, is a relation that captures the negative effect on a peer when it becomes the victim of an attack. It allows us to describe a peer's aversion to being attacked and plays a large role in determining how much effort should go into avoiding attacks or whether to participate in a system at all.

Table 4.1. Table of Utility Terms

U_{ben}	Utility Model for a Benign Peer
U_{mal}	Utility Model for a Malicious Peer
U_{hyb}	Utility Model for a Hybrid Peer
U	Total Utility
B	Total Benefit
C	Total Cost
B_{ben}	Benign Benefit
B_{acc}	Access Benefit
B_{mech}	Mechanism Benefit
B_{mal}	Malicious Benefit
B_s	Benefit from Spying
B_d	Benefit from Denying Service
B_f	Benefit from Serving Faulty Resources
C_{ben}	Benign Cost
C_{mal}	Malicious Cost
C_{vic}	Cost from being a Victim
C_s	Cost from being Spied On
C_d	Cost from being Denied Service
C_f	Cost from being Served Faulty Resources
C_{conn}	Cost being Connected to the System
C_{res}	Cost of Providing Resources
C_{mech}	Mechanism Cost
C_{ms}	Cost of Spying
C_{md}	Cost of Denying Service
C_{mf}	Cost of Serving Faulty Resources
C_{disc}	Cost of being Discovered as Attacker

$$C_{vic} = C_s + C_d + C_f \tag{4.5}$$

Benign benefit, B_{ben}, captures the benefit gained by legitimate participation in a P2P system. It consists of the benefit a peer perceives from accessing resources and any benefit that is derived from mechanisms in the system (for example, incentives for sharing useful resources).

$$B_{ben} = B_{acc} + B_{mech} \tag{4.6}$$

Malicious benefit, B_{mal}, captures the benefit gained from acting maliciously. This is described by the actions of spying on a peer, denying access to a peer, and providing faulty information to a peer.

$$B_{mal} = B_s + B_d + B_f \tag{4.7}$$

Benign cost, C_{ben}, is the cost of participating in the system. This is the overhead cost of staying in the system (as derived and normalized from energy, memory, bandwidth, etc.) in addition to the costs incurred from providing resources and any costs from mechanisms incorporated in the system (such as punishments to prevent freeloading).

$$C_{ben} = C_{conn} + C_{res} + C_{mech} \tag{4.8}$$

Malicious costs, C_{mal}, are costs associated with malicious actions, and include bandwidth costs or processing costs. While these are likely to be relatively small for a malicious peer, they do exist and are incorporated into the relationships.

$$C_{mal} = C_{ms} + C_{md} + C_{mf} \tag{4.9}$$

All of these benefits are tied together and then related to each other to provide the overall utility as described in Equation 4.12. Also included in the cost is the Discovery Cost, C_{disc}, which is the cost of an attacker being discovered (which may take the form of having to exit and re-enter the system or even just a decrease in available peers to attack).

$$B = B_{ben} + B_{mal} \tag{4.10}$$

$$C = C_{ben} + C_{mal} + C_{vic} + C_{disc} \tag{4.11}$$

$$U = B - C \tag{4.12}$$

For the remainder of this chapter some of the details of specific attacks will be simplified by using only with Benign Benefit, Malicious Benefit, Benign Cost, Malicious Cost, Discovery Cost and Victim Cost. Mechanism Benefits and Costs have been omitted since our mechanism does not explicitly involve payments or incentives. Handling the relationships at this level of details provides sufficient information without sacrificing simplicity and generality. Furthermore, this approach allows us to speak generally about attacks rather than identifying specific attacks during the analysis.

For simplicity in analysis, only purely benign versus purely malicious peers are examined, hybrid peers could also be examined, as modeled by Equation 4.13 which represents a peer that uses the system both as it is intended and maliciously. Purely benign peers, as modeled by Equation 4.14, only gain utility from successful transactions and can be modeled with the components Benign Benefit, Benign Cost, and Victim Cost. Malicious peers, as modeled by Equation 4.15 only gain utility from successfully attacking other peers and can be modeled by using the components Malicious Benefit, Benign Cost, Discovery Cost and Malicious Cost.

$$U_{hyb} = (B_{acc} + B_{mal}) - (C_{ben} + C_{vic} + C_{disc} + C_{mal}) \tag{4.13}$$

$$U_{ben} = B_{acc} - (C_{ben} + C_{vic}) \tag{4.14}$$

$$U_{mal} = B_{mal} - (C_{ben} + C_{disc} + C_{mal}) \tag{4.15}$$

4.2.4 Exploratory Requests

The main idea of the resource exploration is to send out exploratory requests in addition to real requests. This process is detailed in Algorithm 1. These exploratory

Input: Peer Preferences

Output: Resource Access Results

while *Resource Not Accessed* **do**
  Calculate P_{exp};
  Generate Request (P_{exp} are exploratory);
  Send Request;
  if *Request Is Exploratory* **then**
    if *Attacked* **then**
      Blacklist Attacker;
    end
  end
end

Algorithm 1: Exploratory Requests

messages are designed to reveal the nature of the peers resulting in an increase in utility for the requesting peer and a decrease in utility for attacking peers. The requesting peer can either send an exploratory message or a request message. The serving peer can either respond with an attack or with a legitimate response. Peers will incur a cost by sending the exploratory messages in terms of a greater amount of Benign Cost, C_{ben}, but exploratory messages reduce the likelihood of being attacked. The decreased likelihood of attack is due to the increasing cost incurred by the attacker if discovered as a malicious peer, C_{disc}. In all cases, each peer will incur a cost of C_{ben}.

At this point, some assumptions must be stated about performing resource exploration. First, it is assumed that the peer responding to the request cannot differentiate between exploratory and regular requests. This assumption is justified by noting that a peer can reuse previously obtained results, self-generated results, or

pre-programmed results depending on the specific application. Second, it must be assumed that the peer sending the requests can verify whether or not an attack has occurred. This assumption is also made in other P2P reputation research [17, 2, 9], since to provide opinions, peers must know whether they were attacked. In many cases, this is manually determined by human users, but due to the cost of human intervention, our work is most useful if the attacks can be automatically determined (for instance, comparing the known checksum of a file to a generated checksum of the file sent by another peer).

Before a peer can decide to utilize resource exploration, it needs to determine at what rate to send out exploratory messages. In our Adaptive Resource Exploration framework presented in Section 4.3.7 it is shown how a peer can improve its utility by learning to play a Nash equilibrium strategy.

4.2.5 Utility Bounds

The utility equations provide a means for selecting to select a value for P_{exp} involving two equations. Equation 4.16 describes the benign peer's average utility per interaction, $Avg_{U_{ben}}$, and Equation 4.17 describes the attacker's average utility per interaction, $Avg_{U_{mal}}$.

$$Avg_{U_{ben}} = B_{acc} - \frac{C_{ben}}{(1 - P_{exp}) \times P_{att}} \ (1 - P_{exp}) \times P_{att} \times C_{vic} \tag{4.16}$$

$$\begin{aligned} Avg_{U_{mal}} = (1 - P_{exp}) \times P_{att} \times B_{mal} - \frac{C_{mal}}{(1 - P_{exp}) \times P_{att}} \\ -P_{exp} \times P_{att} \times C_{disc} - C_{ben} \end{aligned} \tag{4.17}$$

To maximize utility, the peer takes the derivative of Equation 4.16 with respect to P_{exp}, sets the equation equal to 0 and uses the P_{exp} value that produces the maxi-

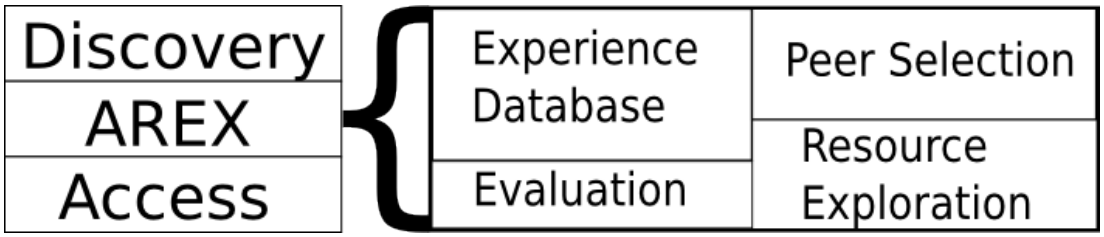

Figure 4.2. AREX Architecture.

mum utility point (since Equation 4.16 is quadratic in terms of P_{exp} there is only one maximum).

Equation 4.16 allows the benign peer to calculate bounds for how high of an exploratory rate it can withstand for the utility it intends to achieve. Furthermore, Equation 4.17 also allows the peer to predict how much its exploratory rate will reduce the utility of the attacker for a given set of attacker preferences and attack rate. However, these equations do not fully incorporate the game that is played between the attacker and benign peer, thus necessitating AREX.

4.3 Adaptive Resource Exploration

There are three main contribution from AREX presented in this section. By implementing the proposed resource exploration algorithms, the AREX peer adapts itself to perform effectively in benign, faulty, and hostile environments. Figure 4.2 shows the architecture of AREX. Based on the available resources and values in the experience database, AREX decides which peer to access and whether to explore or to access the resource. After the request is answered, AREX determines whether an attack occurred. AREX then updates the experience database based on the heuristic function. AREX is independent of both the discovery and access mechanism. It can be plugged into a variety of systems and networks.

4.3.1 Faulty Benign Peers

Since Algorithm 1 has no tolerance for inadvertent errors by benign peers, it can reach a deadlock state in which all peers are blacklisted. To overcome this limitation, an enhanced version of the algorithm is presented in this section. Rather than blacklisting a peer after any action perceived as an attack, Algorithm 2 is reactive but forgiving. Instead of strictly requiring that a peer determine if an attack has occurred in all cases, an indeterminable case is permitted. The indeterminable case permits the user to exert an alternative preference of using a peer in the future. Since AREX does not require that each interaction be recorded and maintained, the amount of memory required for each peer in the system is almost negligible – effectively enough to store an integer (or real number, depending on the implementation). Since the trustworthiness of the system as a whole is also maintained, peers that have not been in range recently can be dropped from the database without a significant effect on the system performance.

Algorithm 2 allows the peer to define how tolerant it is to attack through the punishment function, $\alpha(S)$ or the amount of credibility it gives to valid resources through the reward function, $\beta(S)$. Both of these values are non-negative. If the result of an access is indeterminable, then a tolerance function $\chi(S)$, which represents uncertainty, is used to evaluate the experience. This value can be zero, positive or negative depending on the disposition of the peer. A negative value of $\chi(S)$ is appropriate when a peer can tolerate little risk.

The following terms are used in Algorithm 2:

- K: vector containing experience values for the set of available peers
- k_i: experience value for peer i
- $\alpha(S)$: punishment function
- $\beta(S)$: preference function

Input: Peer Preferences, Known Peers Experience Vector

Output: Resource Access Results

while *Resource Not Accessed* **do**

Select Peer i from K with probability $\frac{k_i+1}{\sum_{j=0}^{|K|} k_j}$;

Calculate P_{exp};

Generate Request (P_{exp} are exploratory);

Send Request;

if *Attacked* **then**

$k_i- = \alpha(S)$;

end

if *Success* **then**

$k_i+ = \beta(S)$;

end

if *Indeterminable* **then**

$k_i+ = \chi(S)$;

end

end

Algorithm 2: Experience Values

- $\chi(S)$: tolerance function

4.3.2 Achieving Nash Equilibrium

The second contribution of AREX is a methodology for playing a Nash equilibrium strategy when there is insufficient information available to calculate the mixed-strategy Nash equilibrium described in Equation 4.1 (Incomplete information game). As a result this section presents an adaptive method for approximating the Nash equilibrium when all peers in the system are strategic attackers and then adapts to

perform better when there exist non-strategic attackers. Hence, the approximation for the Nash equilibrium will be the worst-case performance as a result of AREX.

The peer starts by calculating the opponent's Nash equilibrium point for P_{att} by using Equation 4.2. Then the peer uses Equation 4.16 to calculate the P_{exp} that will result in the highest initial expected utility; however, a peer cannot maintain this strategy, because a strategic peer will adapt its strategy to exploit naivety. Additionally, if the opponent is not playing a Nash equilibrium, and is instead playing a sub-optimal strategy (or is benign), the peer wants to exploit that information to its advantage.

To adapt to the environment, the peer continually adjusts P'_{att}, the estimation of P_{att}, by using Equation 4.18 where ϕ is a discount value (to prevent overreaction) and γ is the indeterminable discount in the range of $0 \leq \gamma \leq 1 - \phi$. Based on the new value of P'_{att}, the strategy is recalculated.

$$P'_{att} = \begin{cases} \phi \times P'_{att} + (1 - \phi) & \text{if attacked} \\ \phi \times P'_{att} & \text{if not attacked} \\ \phi \times P'_{att} + \gamma & \text{if indeterminable} \end{cases} \tag{4.18}$$

4.3.3 Alternate Strategies

It is reasonable to expect strategic attackers may employ different strategies aside from the basic mixed-strategy discussed up until this point. A strategy deployed against reputation mechanisms that could be deployed against AREX involves building a high reputation value in order to later perform attacks later. This is easier to perform in reputation mechanisms since the use of external reputation information allows peers to collude and build reputation without the cost of actually performing those actions. In AREX the actions must be performed for them to have effect on

experience values. This section analyzes an analogous attack on the AREX system by both colluding and individual peers.

In this strategy, the attacker performs benignly to increase the likelihood that that a request is sent rather than an exploratory request. Additionally, the benign behavior will increase the likelihood that the attacker will be selected in the future. After a sufficient period of time, the attacker betrays the trust of the AREX peer and begins to "cash in" its accumulated experience value by attacking. While AREX does not completely prevent this type of attack, it does allow the AREX peer to know under what conditions the attack would benefit the attacker. For the attack to benefit the attacker, the utility received from the attack must be greater than the cost of acting as a benign peer for the duration required to successfully perform the attacks. Equation 4.20 describes the utility derived from this type of attack.

$$P_{sel}(i) = \frac{k_i}{\sum_{j=0}^{|K|} k_j} \times P_{att}^{i} \tag{4.19}$$

$$U = B_{mal} - \frac{C_{ben}}{(1 - P_{exp}) \times P_{sel} \times P_{att}} - \frac{C_{conn}}{(1 - P_{sel}) \times P_{att} \times (1 - P_{exp})} \tag{4.20}$$

P_{sel} and P_{exp} are both functions of the behavior of the peers in the system and the values assigned for $\alpha(s)$, $\beta(s)$, and $\chi(s)$. As a result, manipulating the values of these variables allows the AREX peer to predict what B_{mal} has to be for an attacker to use this strategy. Intuitively, if a peer incurs a cost building up a high experience value, and if $\alpha(s)$ depletes that experience value significantly quicker (such as by halving the value each time) per an attack attempt (so that P_{sel} decreases for the attacker) and P_{exp} simultaneously rises quickly to decrease the likelihood of a successful attack. One way to counter this approach is to allow non-attacking peers to take care of decreasing P_{exp} by acting benignly at no cost to the attacker. The

problem with this approach is that then P_{sel} for the attacker-controlled peers is low, so the opportunity to attack is minimal.

Section 4.5 discusses the effectiveness of this approach in approximating a Nash equilibrium and strategies for sub-optimal attackers.

4.3.4 Strategy Selection

This section describes the third contribution of AREX: handling differences in strategy between individual peers and the system as a whole. In our preliminary work, each peer could be treated individually or the system could be treated as a whole when computing P_{exp}, but there was no methodology for deciding which approach was appropriate.

AREX may operate in systems of heterogeneous attackers. Some may adapt to the AREX strategy while others attack at a fixed rate. This situation begs the question: how can strategic attackers be motivated to attack less while allowing the AREX peer to simultaneously utilize benign peers and optimize against non-strategic attackers?

This question is actually addressed by the two previous solutions. If sub-optimal attackers and benign peers exist in the system, then the two-level approach described in Section 4.3.1 allows the AREX peer to decrease the probability of being exploited by strategic attackers and sub-optimal attackers with a propensity for attacking. This approach results in the system being transformed in such a way that a peer can still treat the system of other peers as a single attacker whose attack strategy is described by Equation 4.21 where P^i_{att} is the probability that peer i will attack.

$$P_{att} = \sum_{i=0}^{|K|} \frac{k_i}{\sum_{j=0}^{|K|} k_j} \times P^i_{att} \tag{4.21}$$

Consider the following three cases:

- The system is mostly benign
- The system consists of sub-optimal attackers
- The system consists of optimal attackers

In the first case, the probability of requesting from any sub-optimal attackers would approach 0, as would the probability of requesting from optimal attackers. Then the system would be modeled by an attacker similar to that of the benign peer. In the second case, the exploratory messages would identify the sub-optimal attackers with the least propensity for attacking. Then the attack strategy of the perceived system would tend toward the the malicious peers with the lowest attack rates when benign peers were not available. In the third case, the system would model the optimal attackers as a single attacker. If heterogeneity was introduced in terms of types of peers in the last two cases, Algorithm 2 would tend toward accessing resources from peers with the lowest attacking rates. Finally, in the case that no peer was distinguishable from another (or no peer was ever accessed multiple times), our approach will treat the entire system as the average of its members.

4.3.5 Experience Heuristics

The original AREX approach to evaluating other peers in a system was to modify the current existing experience value with a constant value. Rather than limiting AREX to linear modifications, $\alpha(S)$, $\beta(S)$, and $\chi(S)$ are defined to be heuristic functions of the current state as perceived by the AREX peer. The heuristic functions provide better control of AREX's behavior such as allowing AREX to take a context-driven approach to resource access. For example, if node churn is high, the reward function can be defined as $\beta(S) = \frac{1}{1-S[ChurnRate]}$ where S is the current state (in this case only node churn rate is included as the state). The contextual reward function causes AREX to provide high rewards to good peers when the churn rate is high.

This causes the peer to utilize resources from good nodes more rapidly since they are unlikely to be available for long and high levels of exploration would deplete energy quickly. Simulations results displaying the effect of this approach can be found in Section 4.5.4. By using general heuristic functions, AREX can additionally take advantage of non-linear functions and piece-wise functions to gain customized control of its behavior.

4.3.6 Redundancy

Another issue that is present in some applications that may use AREX is an inability to populate the AREX peer with sufficient ground truth that can be used to carry out the resource exploration strategy. If the AREX peer cannot generate its own results or cannot do so quick enough, reusing exploratory requests can leak information to colluding attackers resulting in degraded performance. The simplest way to approach this problem is to make multiple requests and to take the most common result as the actual result. This approach can be enhanced by only accepting the result if the number of responses of the most common result is above a threshold value, τ.

The question of how to select an appropriate τ and appropriate redundancy level must also be considered. If either value is too high then resources will be wasted trying to achieve the requirements. If either value is too low, then the risk of failure is increased which will also result in a loss of utility. τ is selected using the estimated value of the attack rate of the system, P'_{att}, defined previously in Equation 4.18. The estimate of the attack rate provides an efficient basis for decided τ since the value is already being calculated, so it does not cause the system any additional computational or memory load. It provides protection from benign failures and it forces a collusive attacker to increase its attack rate to break the security which will

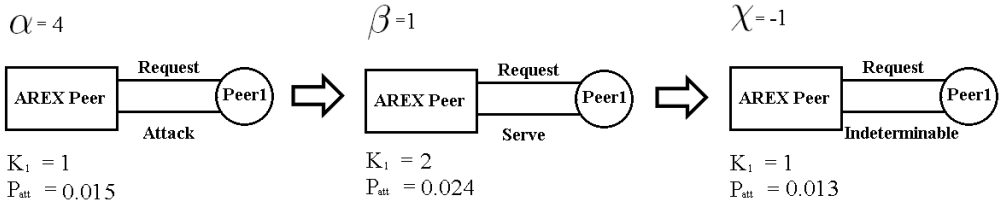

Figure 4.3. AREX Example Behavior.

then lower their expected utility. To determine the level of redundancy, the peer compares the expected cost of an attack to the cost of issuing a request as shown in Equation 4.22.

$$Redundancy = \lfloor \frac{P'_{att} \times C_{vic}}{C_{req}} \rfloor \tag{4.22}$$

While a collusive set of attackers cannot be entirely defended against, the attacking the threshold strategy would be costly to the collusive attacker. To subvert the threshold strategy, the collusive attacker must control a portion of the set that is being queried greater than or equal to τ. This is different from requiring that the attacker possesses τ portion of the system since the probability of selection is defined as $\sum_{i=0}^{|K|} \frac{k_i}{\sum_{j=0}^{|K|} k_j}$. This means that to gain a high likelihood of being able to successfully complete the attack, the peers would have to behave in a predictable manner that is beneficial to the AREX peer similarly to the process required by the original formulation of AREX that has access to ground truth for generating exploratory messages.

4.3.7 Example

Figure 4.3 shows AREX in operation. In this example we view the effects of Peer1's actions of attacking, serving, and an indeterminable response. These actions

occur in a system where the AREX peer is connected to three other peers. At the beginning of this scenario, all three peers have $K_i = 5$ (5 valid responses) and $P^i_{att} = 0$ (no attacks). When Peer1 attacks the AREX peer, Peer1's K_i value decreases by $\alpha(S)$, decreasing its probability of being used for the next access from $\frac{1}{3}$ to $\frac{1}{11}$. Upon successful service of the next request, Peer1's K_i value increases and its P^i_{att} value decreases; however, the overall P_{att} estimate for the system increases because of the increased chance of selecting Peer1. Finally, after the indeterminable result, the probability of selecting Peer1 decreases, and its P^i_{att} value remains the same, thus reducing the system's estimated P_{att}.

To explain this further, consider a more concrete example. Alice has an application on her PDA that allows her to query nearby peers for directions to buildings on a university campus. Since Alice is new to the university, she does not know who she can trust to give her correct directions to certain events on campus. Her PDA has been able to acquire some official information from the university website, but she must request some details from other peers in the system. Alice's application will then use the knowledge that she has to intermix exploratory messages in with her normal requests. The results of these exploratory requests can be verified against the knowledge her application already possesses. From these results, she is able to determine the peers she is more likely to be able to trust future results from.

4.4 Simulation Setup

A discrete, time-stepped, simulator at the level of resource accesses is used to test AREX. At each step, the AREX peer executes Algorithm 2 and sends a request. After receiving the result, the peer recalculates the values k_i and $P^{'}_{att}$. If mobility is enabled, then mobile peers are modeled as moving randomly.

Table 4.2. Default simulation parameters

Number of Peers	1000
Mobility	0%
Connectivity	100%
Avg Benign Reliability	95%
Execution Time (seconds)	1000
Attack Rate of Malicious Peers	100%
C_{ben}	1
C_{vic}	100
C_{mal}	0
C_{disc}	1
B_{acc} (static peers)	120
B_{acc} (mobile peers)	150
B_{mal}	100
$\alpha(S)$	1
$\beta(S)$	1
$\chi(S)$	1
ϕ	.95
γ	0

All attackers have the same preferences, as listed in the chart above. Benign peers always try to return the proper response, but may fail or lose connectivity during service. Note that peers who are unwilling to provide service, i.e. *freeriders*, are a different problem and beyond the scope of this work. Thus the simulation models the reliability of benign peers as normally distributed with a mean of 95% and a standard deviation of 1%.

In some simulations, an alternate version of AREX labeled AREX-BL is used that is the version of AREX which blacklists any faulty or malicious peers rather than using the tolerance built into AREX in order to demonstrate the need for tolerance.

4.4.1 System Model

The simulation first executes as a decentralized and unstructured Peer-to-Peer (P2P) system. This system is static, meaning that the peers are completely immobile, and stable, meaning there is no node churn.

The simulations also include a mobility model, presented in Section 4.5.2. This simulation varies the peer mobility of both moving in and out of range at rates varying between 1% to 100%. The peers operate within a frame of reference relative to the AREX peer. A rate of 1% represents a peer that is unlikely to change its current position relative to the AREX peer and a rate of 100% represents a peer that will always change its current position relative to the AREX peer at each time step. The peer direction traveled is randomly selected from a uniform distribution, and the distance traveled is always a distance great enough to cause disconnectivity from the AREX peer if the moving peer was currently connected to the AREX peer. The choice to model the mobility of other peers randomly instead of with a travel pattern was made in order to simulate a situation in which it would be more challenging for the AREX peer to adapt.

4.4.2 User Model

Two user models are used for comparison. Our first model depicts a naive user who always attempts to access resources. The second model applies the AREX mechanism to access resources.

4.4.3 Attacker Models

The attacker model defines the benefits and costs to the attacker. In our attacker model, the system attacks some percentage of the time ($f\%$). In addition to

assigning an arbitrary percentage for f, in our experiments f can have a special value representing the following cases:

- Always attack
- Attack with a Nash equilibrium
- Attack at variable rates
- Never attack

Attackers are assumed to be consistent in terms of preferences. For example, if an attacker has a B_{mal} of 50, then its B_{mal} will be 50 throughout the entire time of the simulation. The attackers used, as noted by the table above, are based on a powerful and motivated attacker in order to show that our mechanism works against a strong opponent. If an attacker was modeled with less reward for being malicious and more cost for being malicious, then the performance of the mechanism would further improve.

4.5 Results

In the remainder of this section presents and discusses the simulation results of AREX-based resource access. This section presents results over the simulation time that demonstrate AREX's ability to perform with increasing effectiveness as i) parameters vary and ii) user preferences vary. Unless otherwise noted, the simulation parameters are as shown in Section 4.4. In all cases, each simulation was run 1000 times and the average of those runs is presented.

4.5.1 Time-Based Results

First AREX is examined for the average energy cost at a given time during the execution of our system. In Figure 4.4, each point on the plot represents the average cost at the respective timestep. For reference, the average cost of a naive approach

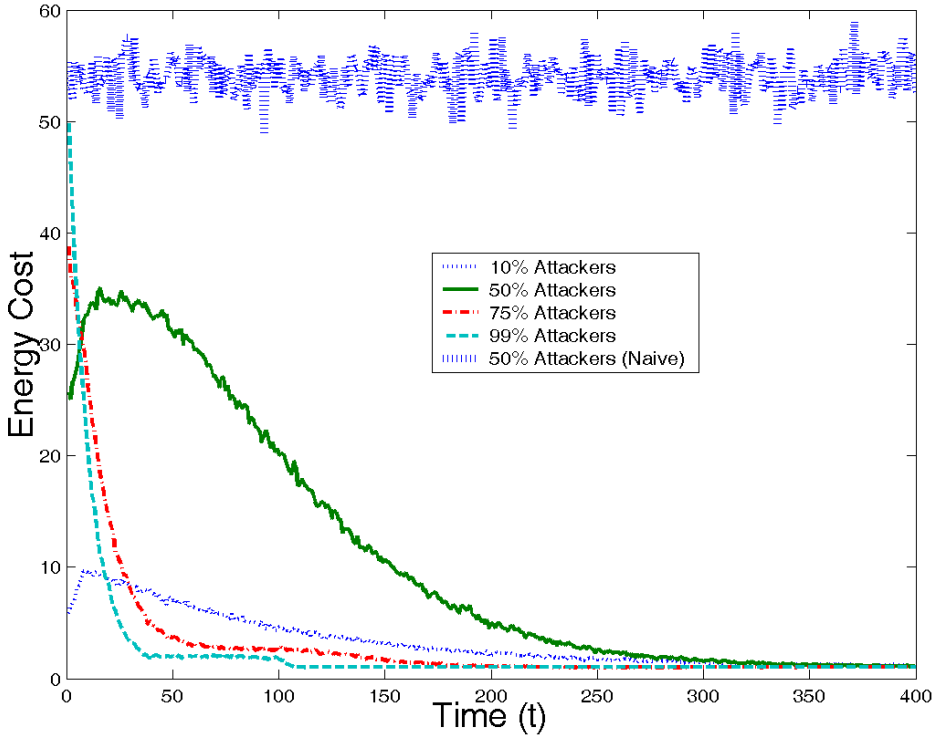

Figure 4.4. Effect of AREX Adaptation Against Various Attack Rates.

when 50% of the system is attacking is also given. As time increases, the average cost approaches the benign cost of participating in the system, even when 99% of the system is attacking. This means that AREX adapts to attackers and learns to decrease the expected cost as the system persists. The reason for this is that as AREX determines who the attackers are, they receive less opportunity to attack. In the 99% case, these attackers are quickly identified and receive only a minimal number of requests, leaving the bulk of the requests to be directed toward reliable, benign peers. Hence, AREX results in a low cost steady-state system.

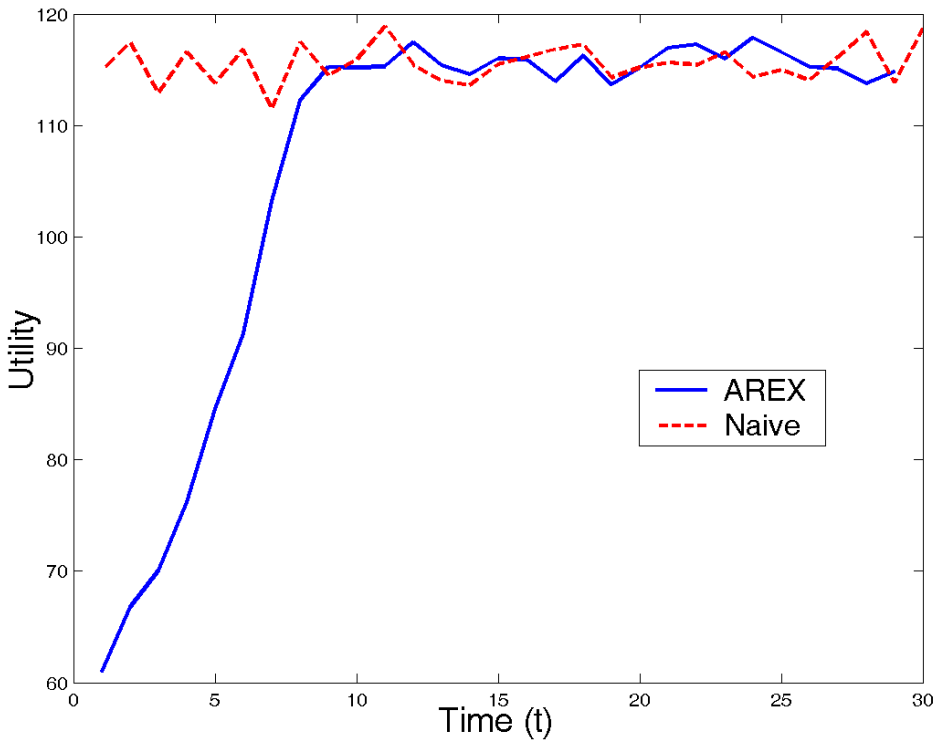

Figure 4.5. AREX Adapting to a Mostly Benign System.

4.5.2 Resiliency Results

As the percentage of attackers (hostility) varies, the number and set of peers known at any given time (mobility), and the number of peers in the system. Mobility was simulated by varying the rate of mobility per time step. At each time step, each peer randomly moves relative to the AREX peer at the rates shown in Figure 4.7.

Figure 4.6 shows the average cumulative energy consumed to access a first resource. The x-axis shows the ratio of malicious peers to benign peers in the system. If a peer is not malicious then it is unreliable on average, 5% of the time. The plot of AREX-BL stops before the other two plots because in some simulations the peer fails to access a resource in the unplotted situations, thus demonstrating the need

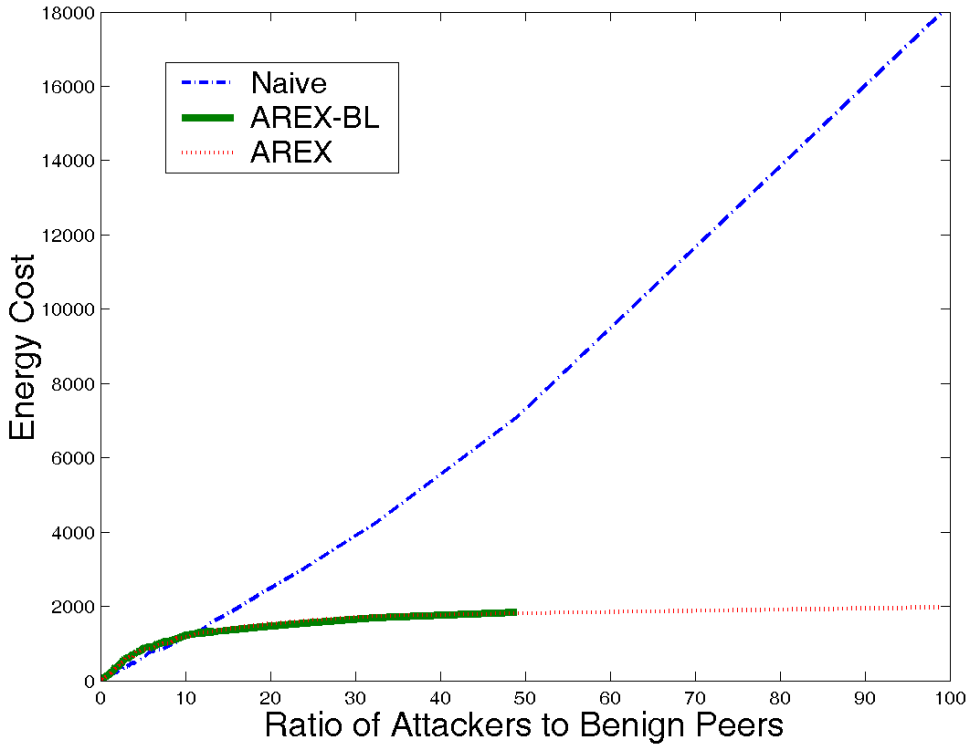

Figure 4.6. Average Cumulative Cost to Access First Resource.

for the tolerant version of AREX. The overhead associated with resource exploration only increases the energy costs a small amount over a naive access strategy when the system is mostly benign, and when the system becomes overwhelmingly malicious, the energy savings of AREX become immense.

In the simulations used to obtain the data in Figure 4.7 and Figure 4.8, the location of each mobile peer was updated at each time step. The results of the mobility simulations reveal that AREX is not negatively affected by mobility, as shown in Figure 4.7. This simulation was carried out with half of the peers as attackers. More interesting results were obtained when the ratio of peers leaving to those entering the range of the AREX peer as shown in Figure 4.8. This figure shows AREX's

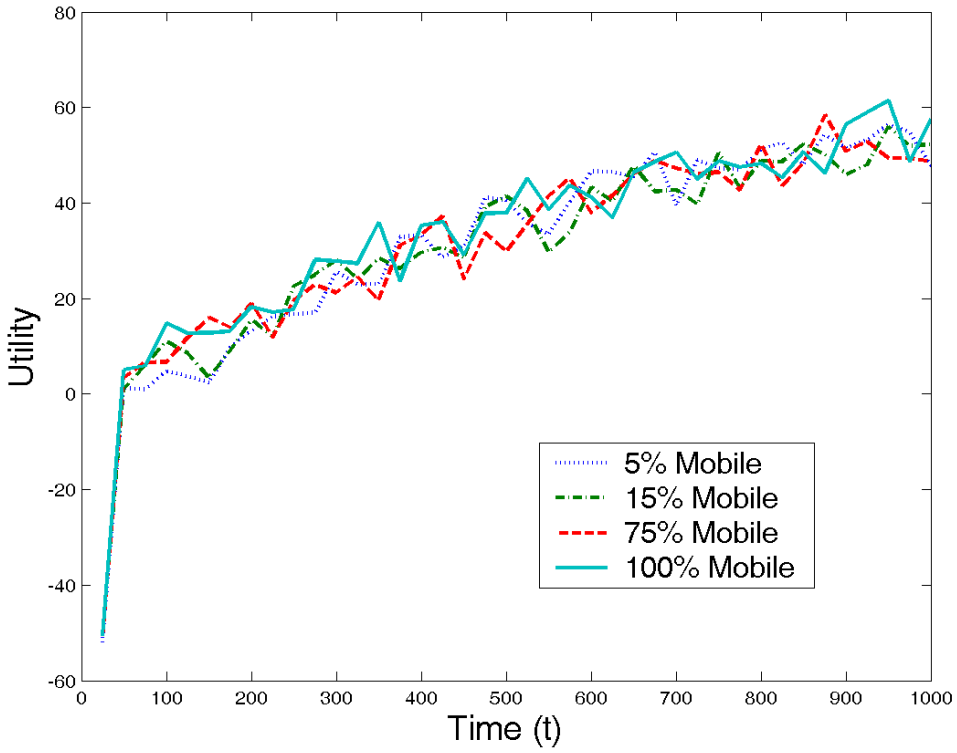

Figure 4.7. Average Utility Over Time for Mobile Peers.

performance is not diminished for similar arrival and departure rates; however, when the number of peers returning to the range of the AREX peer is insignificant compared to the number leaving, utility is diminished, but it must be an extreme case as demonstrated by the 100 : 1 ratio. The reason for the diminished utility is that AREX has fewer benign peers to access at any given time.

AREX's resilience to attack as the number of peers in the system increases is shown next. Intuitively, as the number of peers increases, the longer it takes AREX to adapt. This intuition is shown to be true in Figure 4.9 which shows the steady state utility per request of systems with varying number of peers. The AREX peer reaches a steady state after it adapts to utilizing a small group of reliable peers. Because

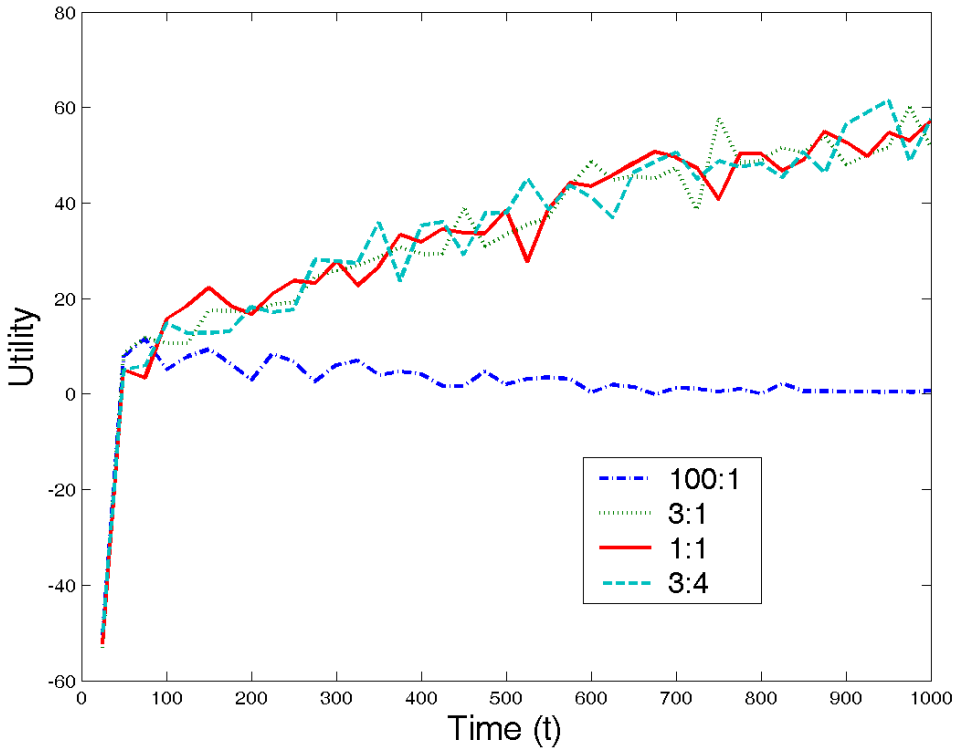

Figure 4.8. Effect of Arrival and Departure on Average Utility Over Time for Mobile Peers.

AREX does not rely solely on that group, but instead randomly selects peers outside of the group (though less often than in the group), steady state utility takes longer to converge when there are more peers outside the group to explore. By varying the $\beta(S)$ parameter, the AREX peer could use a stronger preference for the early members of the group and cause a quicker convergence to steady state utility. It is also feasible to limit the number of peers that AREX is tracking to reduce the time to convergence. In this case, rather than maintaining a database of 10000 peers, AREX would maintain a database of a random subset of those 10000 and only add new peers when the original subset fell below a particular threshold as a result of

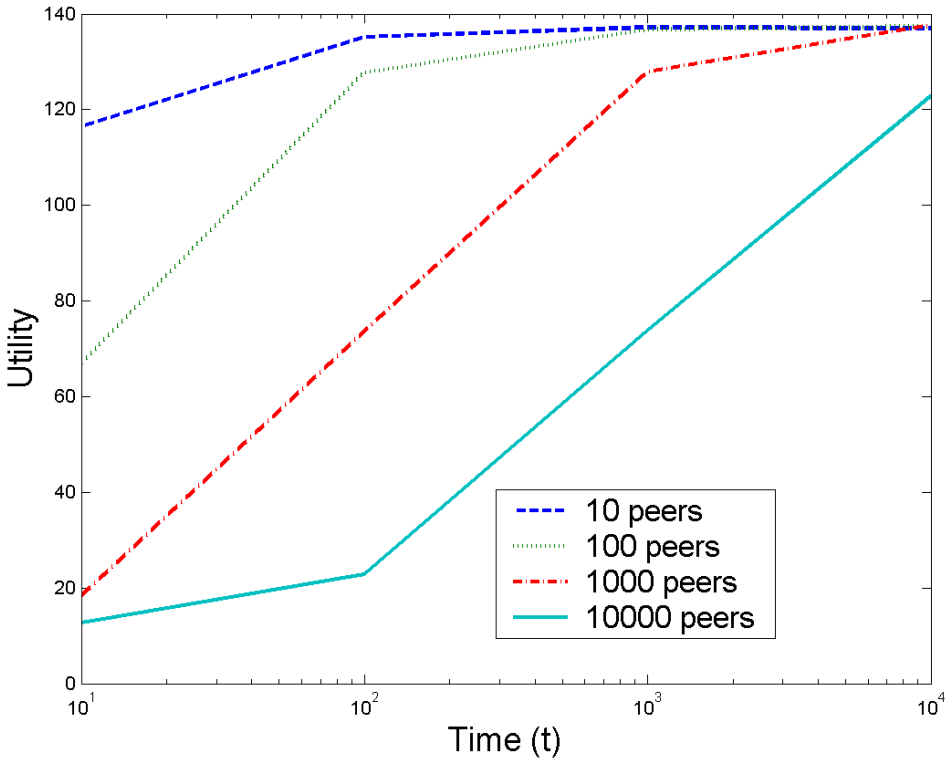

Figure 4.9. Effect of Number of Peers Being Tracked.

mobility, disconnectivity, or a large proportion of malicious peers within that subset. Assuming that the subset was a representative selection of the entire system, it would converge similarly to a system of the subset's size.

4.5.3 Preferential Results

The results discussed here assist us in determining what situations it is appropriate to use AREX and to what extent it will be effective. The ratio of B_{acc} to C_{vic} (AV ratio) and the ratio of B_{ben} to C_{vic} (BV ratio) are the two important preference factors for AREX. the use of AREX decreases a strategic attacker's preference for attacking.

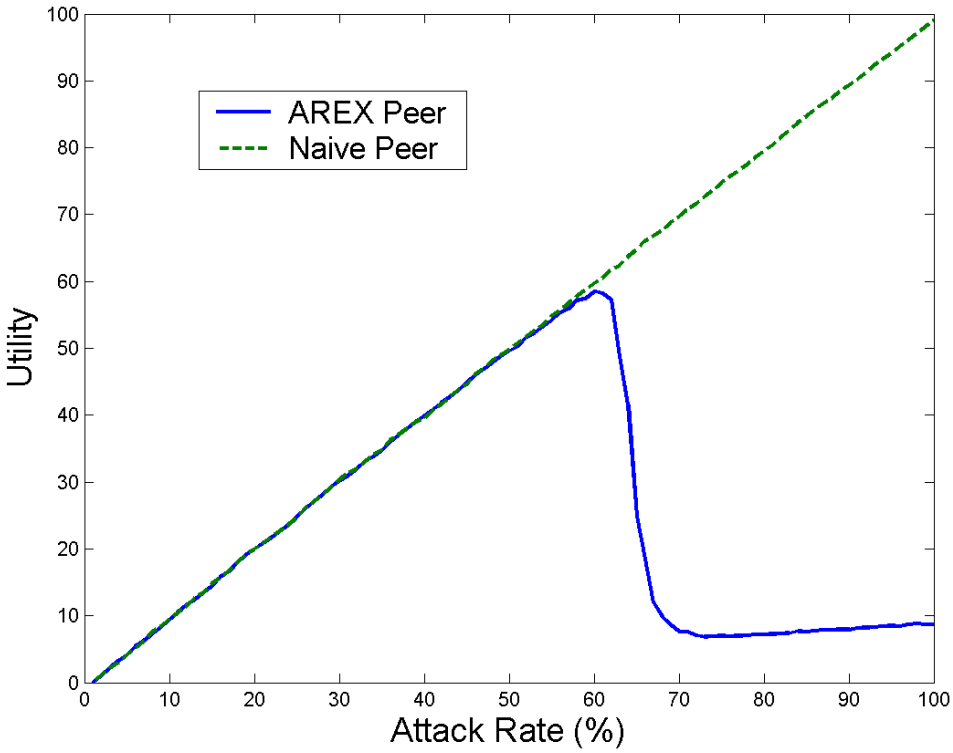

Figure 4.10. Effect of AREX on Opponent Preference to Attack.

Figure 4.10 shows the effect of AREX on an attacker's utility. The graph shows that while AREX has an insignificant effect on the attacker when the attacker attacks less than the Nash equilibrium strategy. When the attacker attacks more often than Nash equilibrium, its utility is greatly diminished. Hence the attacker is motivated to attack significantly less as a result of AREX. The analytically computed Nash equilibrium point for the attacker is approximately 0.643 and the simulation results show the optimal utility to be within a few percentage points of this value. The values differ as a result of the benign peer adjusting to learn its optimal strategy since it does not know the attacker's strategy *a priori*. This allows the attacker to achieve a maximum utility with a rate that is slightly higher than the Nash equilibrium.

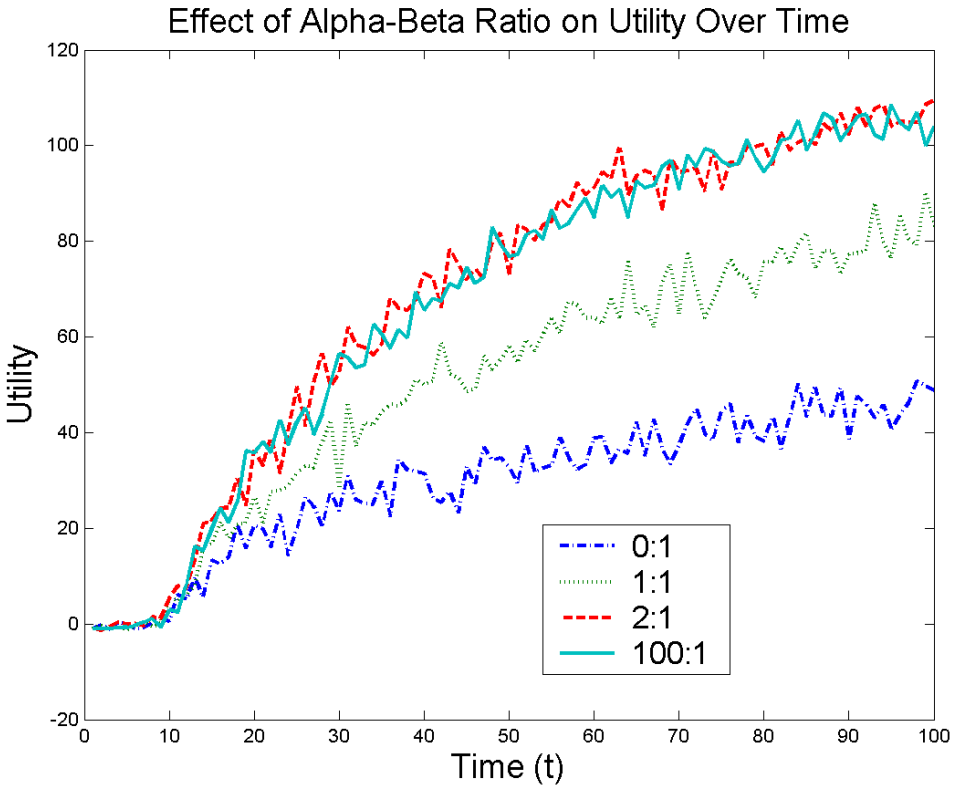

Figure 4.11. Effect of $\alpha(S)$:$\beta(S)$ Ratio on AREX.

The results shown in Figure 4.6 vary as the simulator changes preferences as noted in Section 4.5.2. The change in results occurs based on the ratio of C_{ben} to C_{vic} (RV ratio). As the RV ratio increases, the intersection point (the point that defines when it is in the peer's best interest to change the rate of exploratory messages) also increases with respect to the percentage of the system that is malicious. The RV ratio also affects the difference in costs between any two exploration rates. As the RV ratio approaches 1, the difference in cost between any two strategies as the percent of the system is attacking changes approaches 0. As the RV ratio approaches either zero or infinity the cost difference between any two exploration strategies approaches infinity.

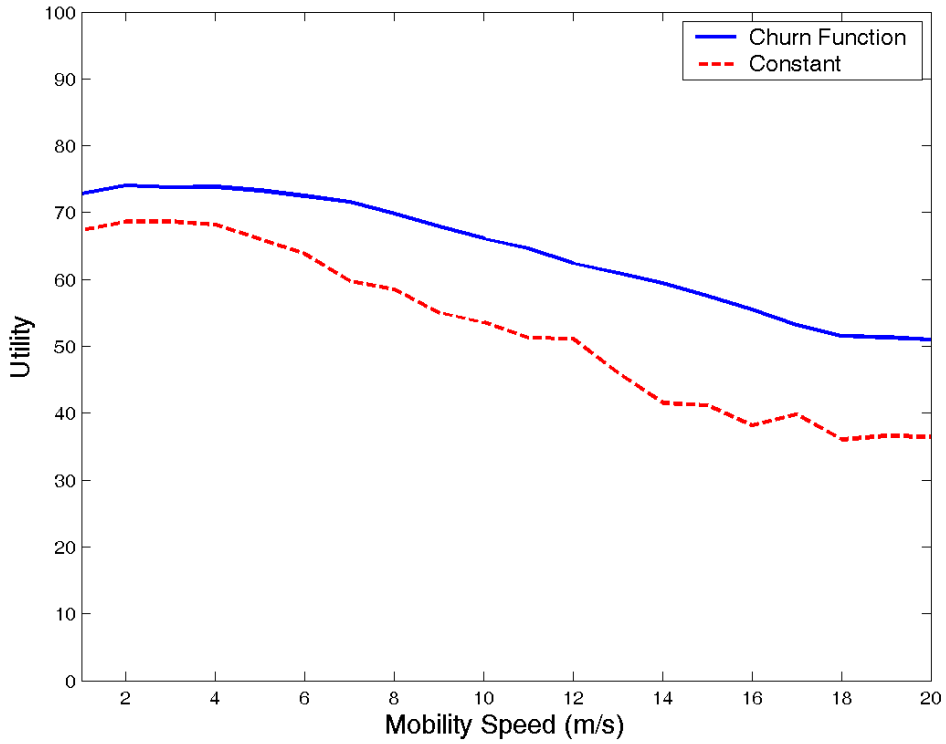

Figure 4.12. Use of a Churn Heuristic Function with AREX.

Figure 4.11 shows that the ratio of $\alpha(S)$ to $\beta(S)$ has little effect on the performance of AREX pending that the ratio is greater than 1. When the $\alpha(S)$:$\beta(S)$ ratio was greater than one, the simulations converged to the same average utility; however, when the ratio was 1 (meaning punishment and reward are the same) or 0 (meaning there is no punishment, only reward), the utility derived by the AREX peer was greatly diminished.

4.5.4 Heuristics

In this section, simulations demonstrate the effectiveness of using heuristic functions as opposed to simple constant values to customize AREX to take advantage of

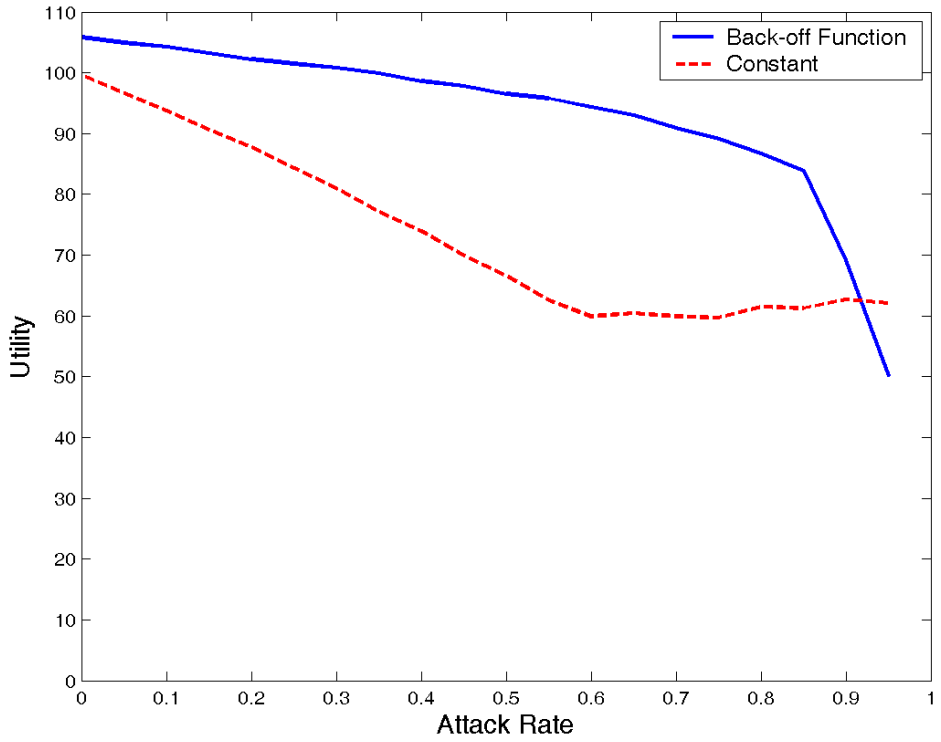

Figure 4.13. Use of a Backoff Heuristic Function with AREX.

contextual awareness. Figure 4.12 shows the use of a heuristic function in comparison to a constant increments. By utilizing the heuristic function, AREX is able to incorporate context into the decision-making process. In this particular example, AREX calculates the churn rate of the peers it knows about and adjusts the $\beta(S)$ value as a function of churn. As churn increases, the peer predicts that it will have less opportunity to utilize a resource, so it places more value on a good interaction and will be more likely to access that resource again. As the speed of movement increases, the amount of time that peers stay in range of the AREX peer decreases. Since AREX requires interactions to adapt, this decreases the effect of AREX; however, that de-

crease can be mitigated by utilizing churn as context and placing more emphasis on interactions that occur when churn is high.

Simulations also demonstrate a heuristic function that uses a backoff function similar to how TCP Tahoe adjusts the congestion window after a packet loss. In this case, any detected attack results in a peer's experience value dropping to a preset (non-zero) minimum level. The result of this approach (shown in Figure 4.13) resulted in more successful results than the constant increment and decrement approach when the percent of the system that is attacking is not too high (under 90%), but when most of the system is attacking, inadvertent errors by benign peers causes the AREX peer to not favor the benign peers as strongly.

4.5.5 Application Specific Results

In this section, simulations demonstrate the effectiveness of AREX for specific applications. In particular, two applications: distributed computation and filesharing.

In the distributed computation application, the AREX peer made requests for a computation process to be run on different peers. Included in the request was a payload of data to be operated upon. The data and operations to be performed were selected such that they would take approximately 1 second to transmit the request, 11 seconds to perform each computation on the AREX peer (an emulated cell phone), and 1 second to perform the computation on the peer receiving the request (a high-end desktop). Similar to the simulations shown earlier in this section, 1000 tasks were performed, there were 100 peers available in the system, and benign peers made mistakes on average 5% of the time. Figure 4.14 shows the average amount of time saved per a task by performing distributed computation using AREX. Even when most of the system is attacking, AREX is able to quickly identify the good peers in

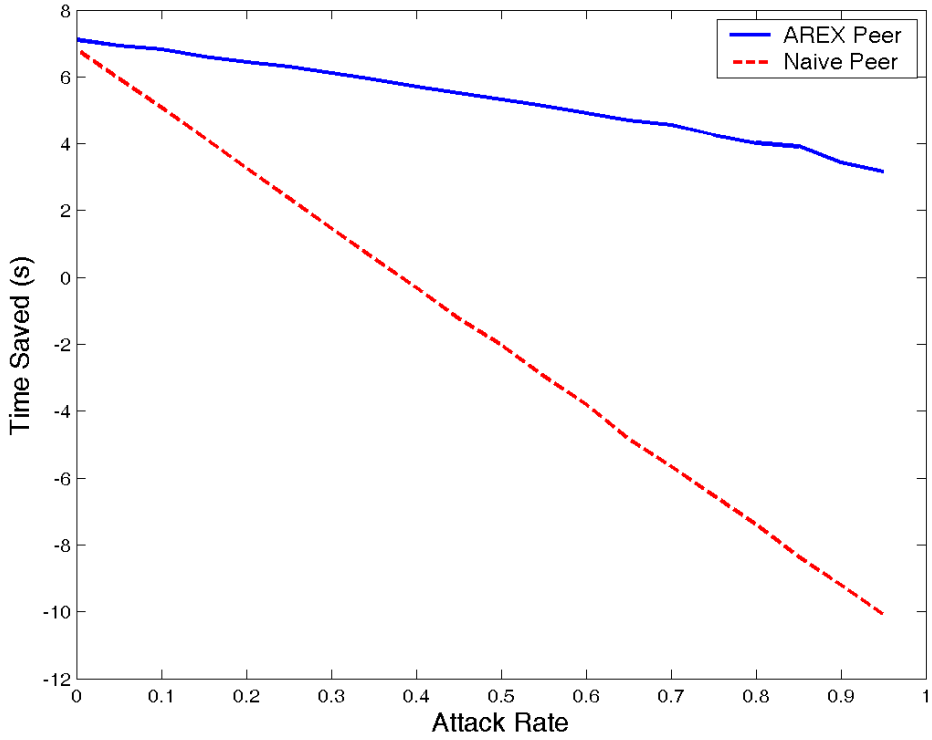

Figure 4.14. Average Time Savings in Distributed Computation.

the system and utilize them to bring the average time savings back into the positive range over the course of 1000 tasks.

In the file-sharing application, the AREX peer made requests for files available from up to 100 different peers. In this experiment, success is defined by the number of correct files that the AREX peer was able to acquire in 1000 requests. Similar to previous experiments, benign peers were unreliable an average of 5% of the time. Figure 4.15 shows AREX's success at acquiring correct files as the number of attackers increases. Even when 95% the system is attacking, the AREX peer was able to acquire approximately more than half of the files it requested.

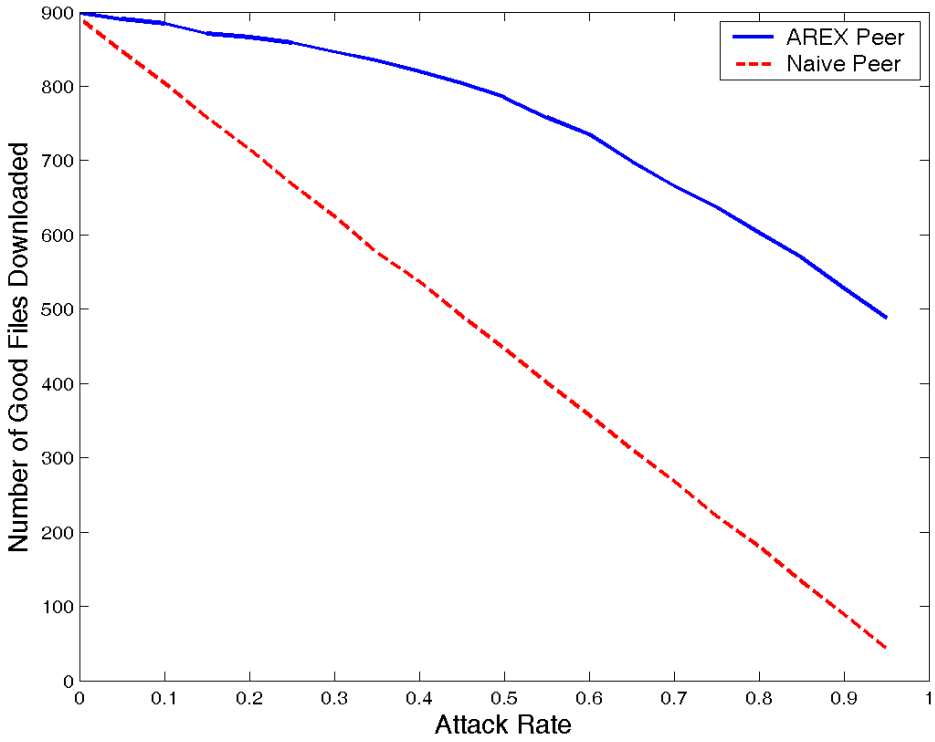

Figure 4.15. File-Sharing with AREX.

4.6 Summary

This chapter presents a novel adaptive mechanism called AREX for secure resource access in uncertain or hostile P2P environments. In AREX, benign peers send exploratory messages to assess the actions of untrusted peers and quickly adapt based on their actions. AREX adaptively balances the trade-off between exploration and utilization of resources to protect the peer running it with minimal energy costs. AREX performs well in dynamic environments where previous work was found to be inadequate. Furthermore, AREX not only benefits the peer running it, but it also reduces rational attacker's motivation to attack by playing an approximate Nash equilibrium strategy against the attacker.

Simulation studies validate our findings and demonstrate the superior performance of AREX in terms of protecting benign peers, rendering malicious peers ineffective, and energy costs. In experiments of specific applications, AREX provided significant improvement, even when the number of attackers constituted 95% of the system. Peers using AREX are able to accomplish such results through identifying the benign peers during the initial adaptation phase and then utilizing the resources of those benign peers to accomplish desired tasks and reduce the amount of resources spent sending exploratory messages. AREX has been applied to two specific tasks that have commonly used a P2P architecture to implement, File Sharing and Distributed Computation. In these applications AREX has shown significant improvement over naive approaches.

CHAPTER 5

RELIABLE SERVICE COMPOSITION

5.1 Introduction

Service composition is the process of combining available low-level services in a system to create an application(or high)-level service. In dynamic systems, such as in mobile or P2P settings, the low-level services that make up an application-level service may be available on nodes with variable connectivity or availability. While nodes that become unavailable can make it harder to compose a service, new nodes may become available that increase the possible composition possibilities. Adding further to the complexity, multiple compositions for a single high-level service may exist. For example, some services may accept or produce a wide variety of data formats, leading to a range of paths that might be possible, even when only a few data transformations are performed.

Ideally, service composition in mobile and uncertain environments should be: i) adaptive to the dynamic nature of connections between mobile nodes, so that it can overcome loss of services due to disconnecting nodes and exploit newly available services due to incoming nodes; and ii) cognizant of the trustworthiness and reliability of nodes. These requirements are salient even in somewhat static and well-connected systems, as unreliable and attacking nodes can pose a threat to the success of accessing composed services.

As a motivating example, let us consider a mobile ad-hoc system in a town where a large number of entities provide services. A police force wants to utilize these services to monitor a large gathering that is taking place and has the potential

to turn into a riot. While there is not any single service that will allow the police to accomplish this, embedded in the environment are traffic cameras, video cameras, microphones, and other such devices. Furthermore, the police have access to a variety of software services available to them to perform tasks such as fusing information, converting information formats, and encryption. In this situation, the police can compose a high-level service by gathering information from camera and microphone services and fusing them together. The audio service can be connected with an audio-to-text converter and the resulting text file can be cross-linked with other information or even translated if the voices are in different languages. Finally, the resulting information can then be encrypted prior to transmission to enhance privacy and security. As one can see from the example, the composed high-level service has many points where it could break if attacked or if a node fails. There are also many possible ways to compose a satisfactory high-level service. The goal of ReSCo is to provide a mechanism for utilizing possible nodes and paths to compose reliable high-level services in the presence of unreliable and possibly malicious nodes.

Service composition frameworks employ a variety of practices to build high-level services from the basic services available in the environment. Some of these techniques require exact matching of services [11], [41], [12], and others perform dynamic matching [37]. All of the existing service composition frameworks evaluate composition choices without considering the trustworthiness of nodes providing the available low-level services.

This chapter proposes ReSCo, a middleware component designed to augment these existing systems with trust information. ReSCo does not attempt to compose services, but rather it takes a list of viable compositions available in the environment as an input and selects which composition to use and which nodes to request the low-level services from to accomplish the composition with a high level of trust. To the

best of our knowledge, no other system currently exists that improves reliability of on-the-fly service composition in open, dynamic systems without relying on pre-existing infrastructure to establish authentication and access control.

There is a substantial body of related works, but nothing that accomplishes the goals of ReSCo. Trust and reputation research in peer-to-peer (P2P) systems focuses heavily on the selection of individual nodes for getting reliable service [8], [17], [9], [42]. As evidenced by the results of research in these systems, the selection of individual nodes makes a significant difference to the reliability and performance of a system, especially when a system contains malicious nodes. Research in service composition systems often focuses on discovering services and determining which services can be combined, rather than on which nodes should be combined to increase reliability. In service composition systems, the presence of unreliable or malicious nodes is magnified by the fact that a composed service requires that all services function correctly at the same time. The research that does focus on providing secure or reliable service composition typically focuses on systems that can utilize a centralized infrastructure to ensure that services can provide valid authentication information. Research in service composition is similar to that in MANET QoS routing, but it differs in that service composition is a two-tiered selection whereas QoS routing is only a one-tier process. The service composition problem determines which possible service path to use to compose a high level service and the underlying nodes that should be used to complete the composition.

ReSCo abstracts the service composition problem into two sub-problems: composition path selection and node selection along a chosen path. ReSCo utilizes local experience information and reputation information to identify unreliable nodes and adapts to select reliable nodes more often. Figure 5.1 shows an example of composing a path and selecting the underlying nodes that provide the service. ReSCo is not

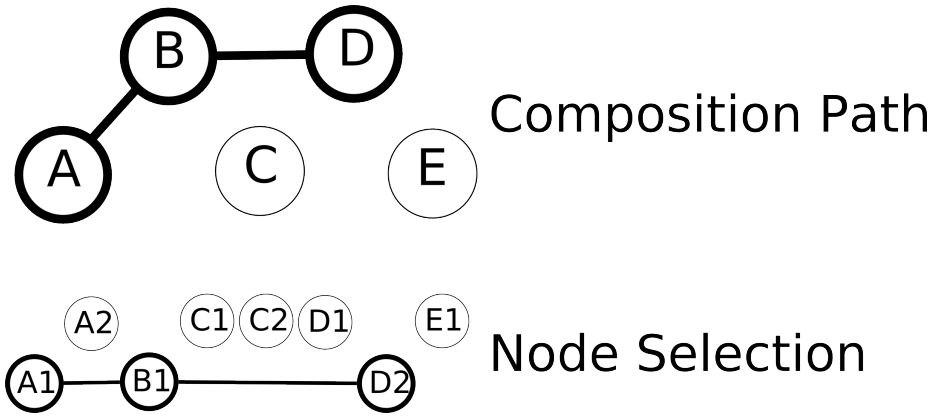

Figure 5.1. An Example of a Service Composition Problem.

limited to this approach, however. ReSCo is designed to be modular and capable of running a variety of strategies to select paths and nodes. Section 5.3 examines several such strategies.

Because of its modular design, ReSCo is independent of the service composition framework that decides which high-level services are composable. This means that ReSCo can provide greater reliability for both existing and future service composition system. In technology-rich environments, such as pervasive and P2P systems, ReSCo can take advantage of redundant services and adapt to select the most reliable nodes. Even in more sparsely-populated systems, though, ReSCo benefits the user. In simulation studies, ReSCo show significant improvement against attackers and faulty nodes in static and mobile environments compared to previousnaive solutions. For example, results show that ReSCo adapts to provide a nearly 90% success rate after 1000 time steps, where a naive approach could only achieve an approximately 50% success rate.

5.2 Design

The objective of ReSCo is to minimize the effect of unreliable and malicious nodes in a service composition system. To accomplish this, a ReSCo instance running on node u keeps track of u's experiences with other nodes in the system in an *experiences database*. In this way, ReSCo can serve node u without relying on collaboration with any other nodes to achieve its goals. At the same time, the experiences database can be extended to use reputation information in addition to u's direct experiences. This allows ReSCo to adapt more quickly in scenarios where most nodes are honest but potentially unreliable.

Input: N, π, Nodes

Output: Ordered List of Nodes to Access

Compute path probabilities (Equation 5.5);

Select path;

foreach *ServiceInPath* s **do**

  Compute node probabilities for s (Equation 5.2);

  Select node and add to access set;

end

Algorithm 3: Selecting the Node Set to Access

Input: π, Set of Requests

foreach *request* r **do**

  Create Access Set from π (Algorithm 3);

  Use the service composed by Access Set;

  Evaluate Result;

  Update evaluation information to database;

end

Algorithm 4: Composing a Service

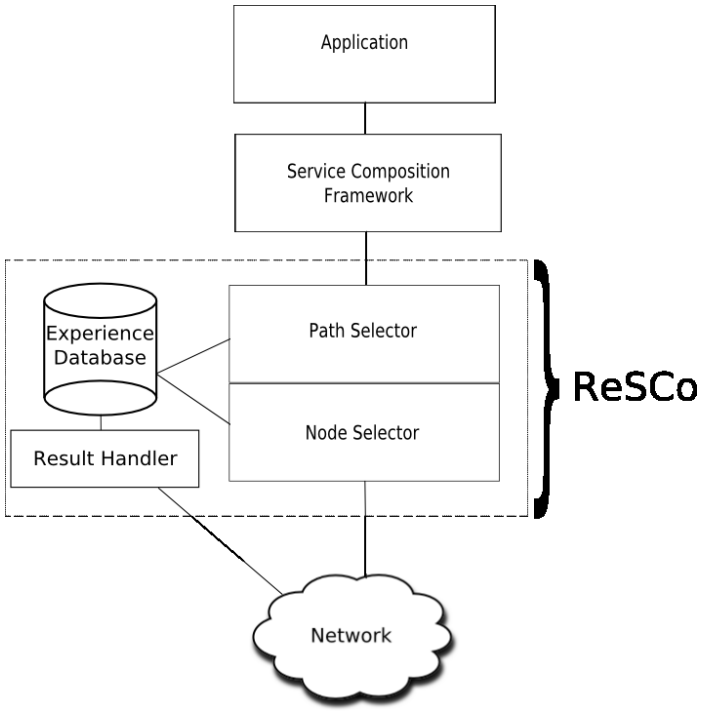

Figure 5.2. ReSCo Architecture.

Algorithm 4 describes the entire process of selecting the set of nodes to be used for the construction of a high-level service. More precisely, Algorithm 4 shows the *default implementation* of ReSCo, in which ReSCo adapts to the reliability of nearby nodes to select trustworthy nodes more often. For simplicity, discussions of ReSCo's path selection and node selection algorithms in the remainder of this paper will be referring to the default implementation unless otherwise noted.

ReSCo provides reliable path and node selection as part of a complete service composition system (as shown in Figure 5.2). ReSCo is designed so that it acts as a layer between the service composition framework and the network. We designed ReSCo to be modular, so that it is independent of the mechanism that determines valid compositions. This allows ReSCo to remain useful as new techniques for creating

high-level services continue to improve. This evaluation of ReSCo utilizes SeSCo [37] for determining which high level service compositions are available because its graph-based composition provides a higher number of successful compositions than discover and match techniques. In particular, SeSCo provides a benefit in mobile environments by managing the available services may not be obvious from only examining the nodes that are within range. This gives ReSCo an opportunity to find more reliable paths than would otherwise be available.

The ReSCo middleware component consists of four primary parts: a Path Selector, a Node Selector, a Request Evaluator, and an Experience Database. The Request Evaluator determines if a service composition was successful. Ideally this should be an automated process, e.g. based on verifying a public key signature or checking a file format. In some applications, however, the Request Evaluator may rely on a user determining whether or not the result is satisfactory and providing feedback, such as viewing a video and clicking on a "thumbs-up" or "thumbs-down" icon. The Experience Database is a storage location for the results of local experience and the experiences reported from other users (via reputation). The database contains cumulative values based on an application-defined weighting for positive and negative experiences, both local and from reputation values. ReSCo can leverage techniques from other systems, such as AREX [42] to populate the experience database through exploration while increasing protection for the node utilizing it. Section 5.2.4 and Section 5.2.5 discuss the Path Selector and Node Selector components in more detail and explain the algorithms that underly their operations.

5.2.1 Request Evaluator

The ReSCo request evaluator defines two heuristic functions, $\alpha(S)$ and $\beta(S)$, of the current state as perceived by the ReSCo peer. The heuristic functions provide

better control of ReSCo's behavior, allowing ReSCo to take a context-driven approach to service composition. ReSCo can track aspects of the computing environment that are defined as the state, S. Examples of which might be the speed of the current user, the node churn in the system, etc. For example, if node churn is high, the reward function can be defined as $\beta(S) = \frac{1}{1-S[ChurnRate]}$ where S is the current state (this case considers only node churn rate as the state). The contextual reward function causes ReSCo to provide high rewards to good peers when the churn rate is high. This causes the peer to utilize resources from good nodes more rapidly since they are unlikely to be available for a substantial amount of time. By using general heuristic functions, ReSCo can additionally take advantage of custom functions to adapt to different contexts instead of simply using constant values.

5.2.2 Experience Database

The ReSCo database keeps track of the current experience values for the known services. Each database entry contains an identifier for the service and an experience value. Because the experiences are compressed into an integer value, the individual entries for each node do not consume a significant amount of memory and the database scales linearly with the number of nodes in the system. Even though the database does not use a large amount of memory, it is beneficial to manage the size of the database. As the number of known nodes grows, good nodes gain trust more slowly because their probability of selection grows more slowly. The database is reduced to the maximum cache size by removing the entries that are closest to the initial value of the database once the database reaches a threshold size and continues until the database is below the threshold size.

This approach has two important implications. First, it means that upon re-entrance, the value in the database for a node that was previously purged will be

close to what it was when the node was in the database, thus mitigating the "white-washing" effect that might occur by removing peers (in other words, the approach seeks to minimize the information that is lost by purging database entries). Second, an important effect in mobile or other dynamic systems is that nodes that have not been seen recently will be purged, since their experience values will not change as quickly as the rest of the system's experience values grow.

Table 5.1. Table of Commonly Used Terms

π	Set of possible paths
$P_{prune}(x, node)$	Probability of pruning a service from the database
$Services$	Set of services used to compose paths
$Nodes$	Set of nodes that provide a service
$TE(x)$	Total Experience value for a service x
$N(x)$	Set of values of nodes with service x
$S(x)$	Set of services composing path x
$E_\sigma(x)$	Expected success for a service x
$E_\rho(x)$	Expected success of a path x
P_x	Probability of success for service x
P_σ	Probability of selecting a service
P_ν	Probability of selecting a node
P_ρ	Probability of selecting a path
P_α	Probability that a node attacks
P_s	Probability that a request was successful
t_{conn}	Amount of time a node is connectible
P_{avail}	Probability that a node is available
P_{awake}	Probability that a node is awake
$P_{inRange}$	Probability that a node is in range
$\alpha(S)$	Punishment heuristic function for unreliable service
$\beta(S)$	Reward heuristic function for reliable service

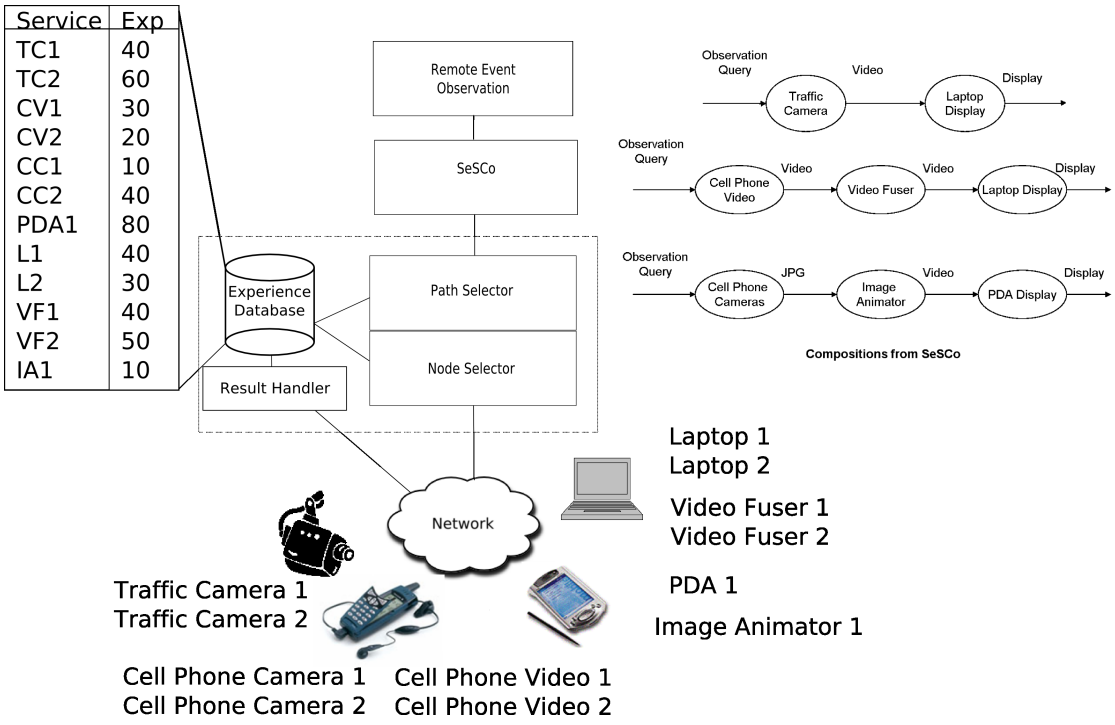

Figure 5.3. Example Composition Scenario.

5.2.3 Example

An example scenario of how ReSCo is used to compose an application-level service is now presented. For the details on the methodology of the calculations, see Sections 5.2.4 and 5.2.5. In this example, the user is trying to compose a service to get a video observation of a remote event (possible compositions, the experience database, and available services are depicted in Figure 5.3). SeSCo is able to determine three ways to accomplish this task: i) Using a traffic video camera and displaying it on the laptop ii) Using a predefined set of cell phone video cameras in the area of the event and fusing the information to create a comprehensive picture then displaying it on the laptop and iii) Using a predefined set of still photo cell phone cameras in the area of the event and using an animator service to turn the sequential images into a video, then showing the animation on the PDA screen.

Table 5.2. Selection Rate for Paths in Example Application (Figure 5.3)

$S(x)$	$TE(x)$	$E_\sigma(x)$	$E_\rho(x)$	$P_\rho(x)$
TC – L	50 – 35	0.185 – 0.130	0.02405	88.8%
CV – VF – L	25 – 45 – 35	0.093 – 0.167 – 0.130	0.00201903	7.5%
CC – IA – PDA	25 – 10 – 80	0.093 – 0.037 – 0.296	0.001018536	3.8%

Table 5.3. Selection Rate for Nodes in Example Application (Figure 5.3)

$N(x)$	$P_\nu(x)$
TC1	40.0%
TC2	60.0%
L1	57.1%
L2	42.9%

These possible compositions are passed from SeSCo to ReSCo. ReSCo then looks up services in the experience database and uses the information to perform the composition path calculations (shown in Table 5.2). After the calculations are performed, ReSCo selects a path and then performs the calculations (shown in Table 5.3) to select the nodes to request for the low-level services. Note that this is the default implementation. Alternate selection algorithms (discussed in Section 5.2.6) would be utilized the same way by ReSCo, except the calculations would differ.

5.2.4 Path Selector

The first task of ReSCo is to decide which composition path to use. A composition path refers only to the types of services that will be used, not the individual nodes that provide those services. To do this, ReSCo randomly selects a path with a probability computed by the portion of the expected reliability of the services that compose each path. The expected reliability is determined by the product of the expected reliability of a service. The reason ReSCo takes the product of the reliability

percentage is twofold. Because the values are between 0 and 1, it means that longer compositions will be chosen less often than shorter compositions, unless they are significantly more reliable. Second, it means that if there is a choke point (a service that is significantly less reliable than other services), then that path will be chosen less often. The expected reliability of a service can be described by the expected reliability of the nodes that compose the service (which is discussed in Section 5.2.5).

The service experience is calculated with Equation 5.1 and is the total sum of the experiences that the node running ReSCo has had with other nodes that provide that particular service. Table 5.2.2 provides a list of commonly used terms.

$$TE(x) = \sum_{i=0}^{|N(x)|} N(x)_i \tag{5.1}$$

The probability of selecting a node for a service is given by Equation 5.2. It is equivalent to taking the ratio of the experience values of a node to that of the total experience value of that particular service.

$$P_\nu(x, node) = \frac{N(x)_{node}}{TE(x)} \tag{5.2}$$

The expected experience value of a service (Equation 5.3) describes the weighted average of all the nodes that provide a particular service. In particular it is calculated by summing the probability that a node will be selected by the current experience value of that node over all the known nodes that provide the particular service.

$$E_\sigma(x) = \sum_{i=0}^{|N(x)|} P_\nu(x, N(x)_i) \times N(x)_i \tag{5.3}$$

A path is composed of one or more services. To determine an expected experience value for the entire path (Equation 5.4), ReSCo takes the product of the expected values of each individual service. While this value has little meaning in

isolation, it is used in Equation 5.5 to calculate the probability of selecting a path. Similar to the computation for selecting a node, a path is selected with a probability equivalent to the ratio of its expected experience value to the sum of the expected values of all possible paths.

$$E_\rho(x) = \prod_{i=0}^{|S(x)|} E_\sigma(i) \tag{5.4}$$

$$P_\rho(x) = \frac{E_\rho(x)}{\sum_{i=0}^{|\pi|} E_\rho(i)} \tag{5.5}$$

ReSCo makes the selection randomly instead of only choosing the best path for several reasons. First, it results in a natural load balancing. The most trustworthy paths will obviously receive more of the traffic, but among reasonably trusted paths, the traffic will be balanced relative to their trustworthiness. Next, randomly selecting paths allows us to explore the service-space. This point is particularly important as ReSCo is designed for dynamic environments. Since nodes may be mobile (both the node accessing services and the service nodes themselves), the best service selected by a composition scheme may not be on a static node and may vary significantly throughout the time in the system and random selection allows us to transition as the available system changes.

5.2.5 Node Selector

Once a path has been chosen to construct, ReSCo must then choose the individual nodes that will provide the services that are required to construct the high-level service. The selection process is similar to that of Path Selection. The node that will be accessed for each service is selected randomly with the probability equal to the

proportion of its contribution to the total experience value (Equation 5.1) for that service as described in Equation 5.2.

Algorithm 4 is the operating procedure for the architecture shown in Figure 5.2. The composition layer presents the ReSCo layer possible compositions for a request, then ReSCo computes which composition to use and which nodes to access for each service. After the services have been accessed, ReSCo adds any available evaluation information from the result to the experience database.

5.2.6 Variations on the Selectors

In addition to the default composition path and node selectors, ReSCo has several additional selectors that can be accessed by applications through the middleware component. These selectors can be called specifically by the application or set as the default selector.

The primary alternative selector is the best node selector. It is similar to the deterministic selector except it differs in how the path is chosen. The deterministic selector chooses the path that has the best average reliability. The best node selector performs the best, but completely eliminates any exploration. It is also completely vulnerable to attacks where the attackers build up a high reliability score and then use it to perform a large number of attacks. In a static and consistent system, the best node selector is the optimal choice once the experience database has been populated with a sufficient amount of information. The best node selector is best used for critical compositions.

5.3 Evaluation

This section presents a mathematical reliability analysis to evaluate the effect of ReSCo on service composition systems. Furthermore, this section includes de-

tailed simulation results of ReSCo for such varying factors as attackers, mobility, and composition requirements.

5.3.1 Analysis

This section details an analysis of the success rate of a service composition system and the effect of ReSCo on the success rate. In general, the success rate of a composition system is the probability that each peer in the path correctly provides the requested service. This is shown in Equation 5.6

$$Success = \prod_{i=0}^{|Path|} P_s(i) \tag{5.6}$$

If it can be assumed that all services are provided with the same average reliability, the resulting success rate when randomly selected will be the average reliability of the peers in the system to the power of the path length as given by Equation 5.7.

$$Success_{Rnd} = (P_s)^{|Path|} \tag{5.7}$$

$$Success_{ReSCo} = \prod_{i=0}^{|Path|} \sum_{j=0}^{|Nodes_i|} P_\sigma(j) \times P_\alpha(j) \tag{5.8}$$

The probability of selecting a node (Equation 5.2) does not give any means for determining the expected experience value for a node. The experience value for node i that provides service s, $N(s)_i$, is a function of the attacker model. Thus, by modeling attackers mathematically, node behavior is defined through an equation that can be used to predict impact on the system. For example, this analysis models an attacker who attacks at a consistent rate over time with Equation 5.9 where t is the amount of time that node i has been available. This equation accounts for the likelihood of

the node i) being part of the selected path, ii) being selected to provide a service, and iii) attacking or failing.

$$\begin{aligned} N(x)_i = P_\sigma(x,i) \times t \times P_\alpha(i) \times \sum_{k=0}^{|\pi|} P_\rho(k) \times \alpha(S) \\ + P_\sigma(x,i) \times t \times (1 - P_\alpha(i)) \times \sum_{k=0}^{|\pi|} P_\rho(k) \times \beta(S) \end{aligned} \tag{5.9}$$

In the case of the attacker modeled in Equation 5.9, as time goes to infinity the set of peers with the best performance will be the ones selected. This value will then become the expected performance of the system. Assuming that there exists some node in the system with a reliability above $\frac{\beta(S)}{\alpha(S)+\beta(S)}$, the expected value of any node that fails less often than $\frac{\beta(S)}{\alpha(S)+\beta(S)}$ will diminish to 0. A node with a reliability between $\frac{\beta(S)}{\alpha(S)+\beta(S)}$ and the maximum reliability that can provide that service will grow in experience value toward infinity, but at a rate much slower than that of the most reliable set of nodes.

The success of ReSCo predicted against the attacker modeled in Equation 5.9 can also be affected by factors such as mobility and connectivity of the attacker. Therefore, t_{conn} is presented as an alternate definition for time that accounts for intermittent connectivity in the system. t_{conn}, Equation 5.10, consists of the amount of time that the attacker is connected to the ReSCo peer.

$$t_{conn} = t \times P_{avail} \tag{5.10}$$

This work does not explicitly define P_{avail} since many unknown factors could affect it, but for these purposes it is modeled as the probability that a node is awake (as opposed to in sleep mode to conserve energy) and within range as shown in

Table 5.4. Default simulation parameters

Number of Simulation Executions	200
Number of Peers	100
Number of Peers in Mobile Simulations	300
Mobility	0%
Connectivity	100%
Average Benign Reliability	95%
Execution Time (seconds)	1000
Attack Rate of Malicious Peers	100%

Equation 5.11 (this equation makes the assumption that a node being awake and being in range are independent).

$$P_{avail} = P_{awake} + P_{inRange} - P_{awake} \times P_{inRange} \tag{5.11}$$

5.3.2 Simulation Setup

The simulator is designed to take an input consisting of the possible paths that can be used to create a service, and then create that service from the underlying nodes available in the network. For most simulation experiments, ReSCo is simulated and compared with a system in which all parameters are the same except that the service path and the nodes that compose it are chosen randomly from the set of available paths and available nodes that can successfully fulfill the composition request. Where relevant, ReSCo is compared to the default behavior of the service composition system that does not use ReSCo.

5.3.3 Results

The simulation results are grouped into several sections. This section examines ReSCo's response to varying numbers of attackers and unreliable nodes, how ReSCo

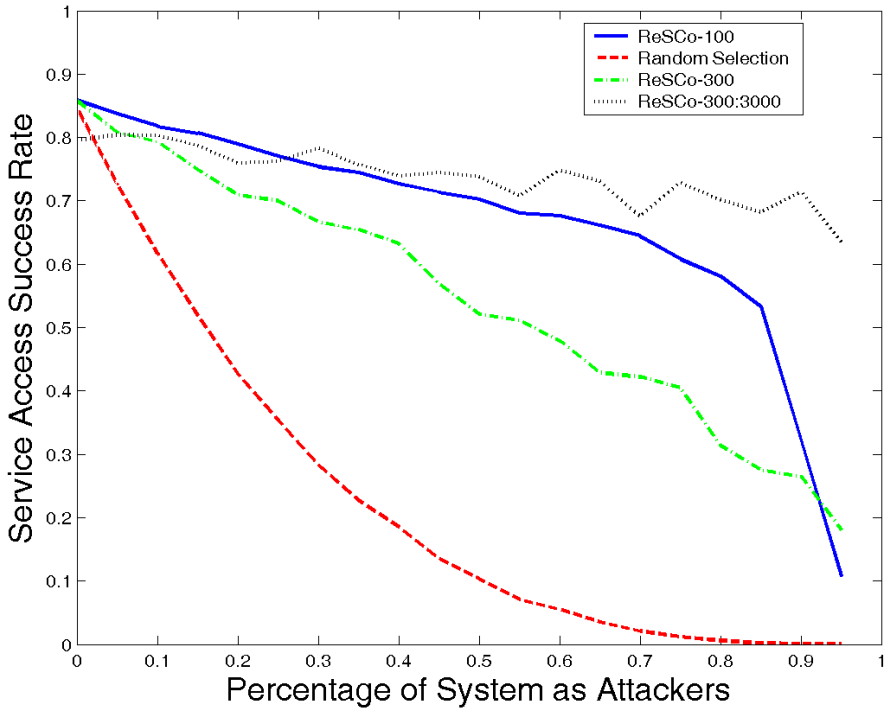

Figure 5.4. Effect of Attackers on Success (Approach:NumPeers:NumIterations when differing from the default).

adapts over time, the effect of differing composition requirements on ReSCo, the effect of mobility on ReSCo, and the effect of augmenting ReSCo with reputation.

5.3.4 Attackers and Unreliable Nodes

The first experiment is designed to show the resiliency of ReSCo against an increasing number of attackers in a system. Figure 5.4 shows that as the number of attackers increases, random selection shows an exponential decrease in the percentage of successful high level services it can access. ReSCo naturally decreases in the percentage of successful constructions accessed, but is successful over half the time as long as less than 85% of the system is attacking. The reason for the sharp decrease

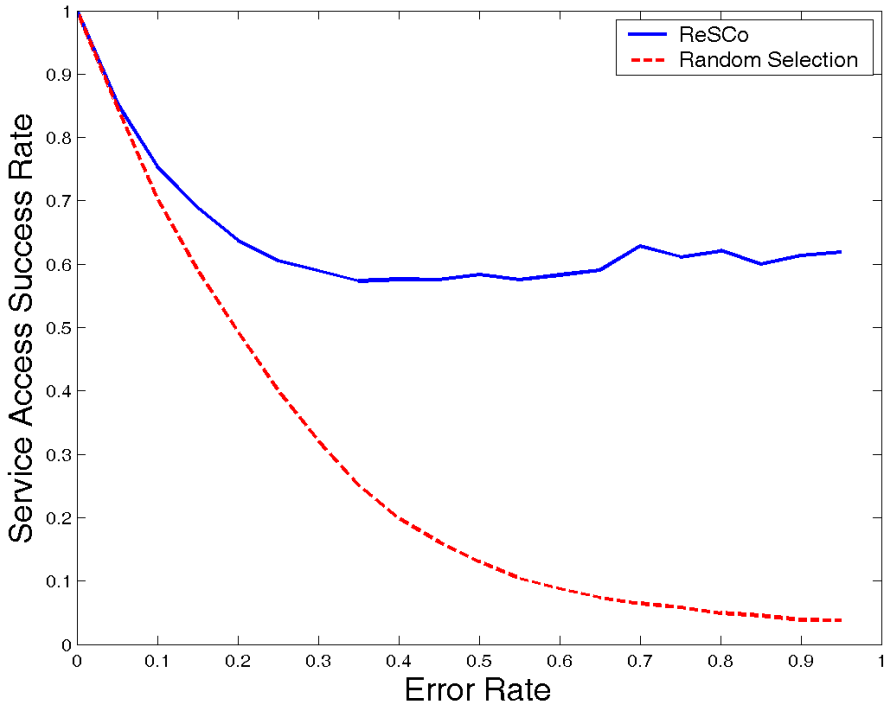

Figure 5.5. Effect of Unreliable Nodes on Success (with increasing standard deviations).

after this point in success is not a failure of ReSCo, but rather is a result of the fact that when simulating 100 peers in a system, there exist some cases in which there is no possible way to construct a path out of trustworthy nodes. To illustrate this, consider a system in which is trying to compose a high level service from two possible paths consisting of three services, $\{A, B, C\}$ and $\{D, E, F\}$. If 95% of the system is attacking and there are 100 nodes in the system, then there will be 5 nodes that are benign. This means there there may be no benign paths possible. To demonstrate that the effect is not present in larger systems, Figure 5.4 also shows ReSCo in a system with 300 nodes. Since more nodes are included, it takes more time to adapt, so a 300 node simulation plot is shown after 3000 time-steps (Labeled ReSCo-300:3000)

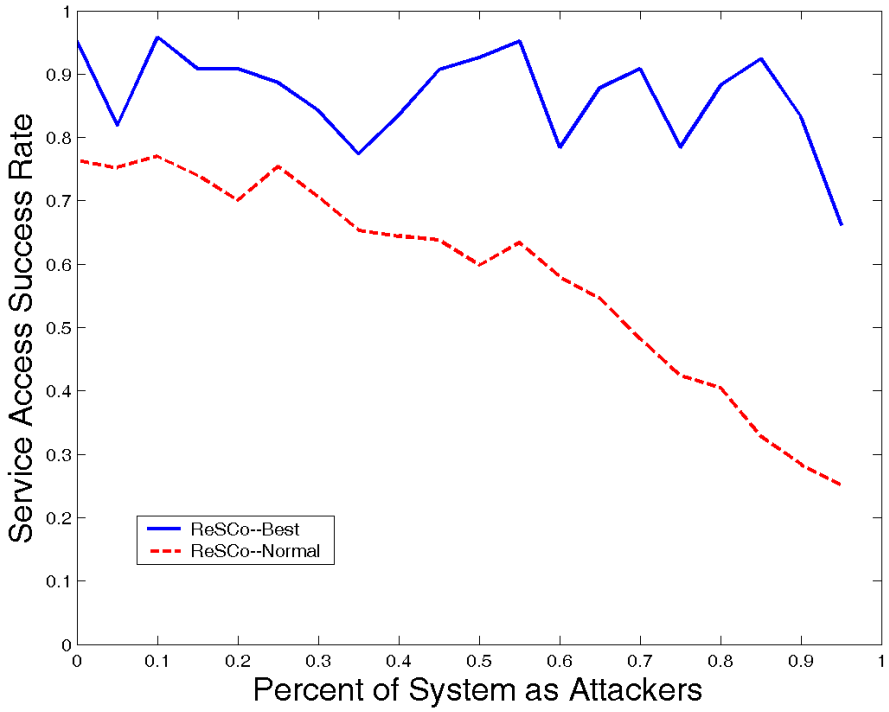

Figure 5.6. Comparison of Stochastic Selection and Best Node Selection.

in addition to the default of 1000. After this additional time is used, the adaptation produces favorable results that do not include a sharp decrease in performance with high levels of attackers.

In addition to attackers, some nodes may become increasingly unreliable (or be malicious and attack at a rate of less than 100%). Figure 5.5 shows ReSCo's resilience to unreliable nodes and its ability to determine which nodes are more reliable. For each data point, the simulation generated every node in the system based on a normal distribution with an average error rate as given by the x-axis and a standard deviation of half the error rate (and a minimum and maximum of 0% and 100%, respectively). If the error rate was not selected from a distribution, but instead was constant, then the ReSCO results would only contain statistically insignificant differences from random

selection. The reason for this is that ReSCo must adapt by finding more reliable nodes, and if all nodes are equally unreliable, then it cannot perform any better than random selection.

This section also examines the effect of the best node selection algorithm for service composition. Figure 5.6 shows the improvement provided by Best Node selection. In this simulation, the first 500 seconds are spent using the normal stochastic version of ReSCo's selection algorithms in order to populate the experience database (with 300 nodes in the system). The next 500 seconds produce the data shown in Figure 5.6. This result demonstrates that utilizing the Best Node selection can be highly effective, even with significant numbers of attackers. The approach still requires that the database be populated prior to use of the selection algorithm, but it is beneficial for critical compositions that need additional protection from failure. The Best Node selection can have unreliable results (as evidenced by the erratic plot in Figure 5.6) because it stops the adaptation process, so as changes occur in the system or if the time spent training does not result in a sufficient approximation of the state of the system, the use of the Best Node selection can produce poor results.

5.3.5 Adaptation over Time

This section shows the adaptation of ReSCo as time progresses. Figure 5.7 shows the cumulative success rate up to each given time step in a system with 20% attackers. The result shows how a standard setup of the system progressively adapts to its environment in order to provide better service over time. Also the results show user expected success rate over a given time. The third plot in Figure 5.7 shows ReSCo's cumulative success in a system that has 80% attackers and that after 100 time steps, ReSCo has had the same success as the naive approach in a system with 20% attackers.

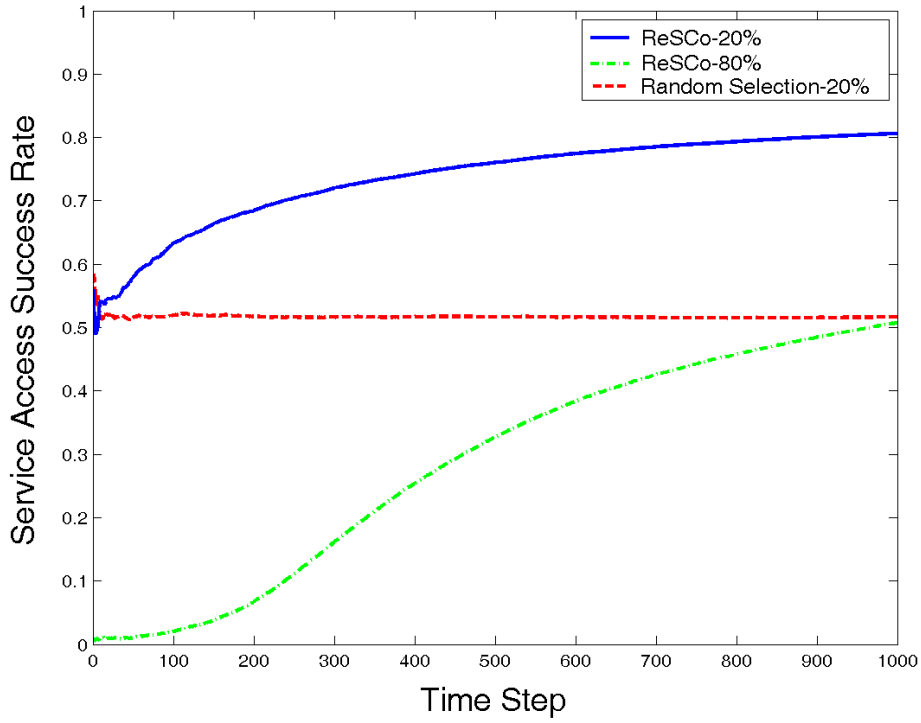

Figure 5.7. Average Cumulative Success at Each Timestep.

Similarly Figure 5.8 shows the average success rate at each time step in the same system setup. The difference between Figure 5.8 and Figure 5.7 is that the Figure 5.7 shows the total success rate to a give time step and Figure 5.8 shows only the success rate for a specific time step. This result provides a view of what a user can expect the success rate to be at any given time. In both cases, the randomly selected paths show no improvement over time. ReSCo shows a 60% improvement in the cumulative success rate over the course of 1000 time steps, and an average of a 74% improvement on the 1000^{th} time step. Figure 5.8 also shows ReSCo's adaptive properties with a system that is 80% attackers (the same one as shown Figure 5.7). This figure shows that a node using ReSCo sees improvement over naive approaches within 300 time-steps despite being in a system with 300% more attackers.

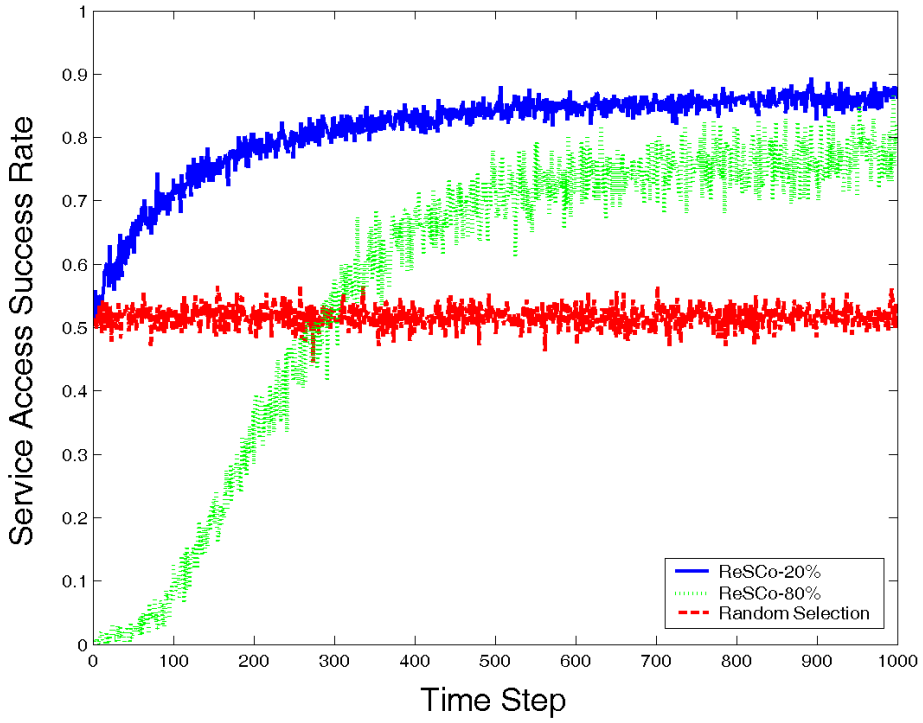

Figure 5.8. Average Success at Each Time Step.

5.3.6 Path Length

The length of a path can have significant effect on the ability to successfully create and access a high-level service. As the number of services needed to create a composed service increases, the opportunity for a single node in the service to fail increases. The effect is clearly shown by the exponential decrease of the Random Selection curve in Figure 5.9 with 20% of the system as attackers. Through its adaptive selection algorithm, ReSCo is able to mitigate this problem and still successfully complete requests 55% of the time, even when the requests are 10 services long.

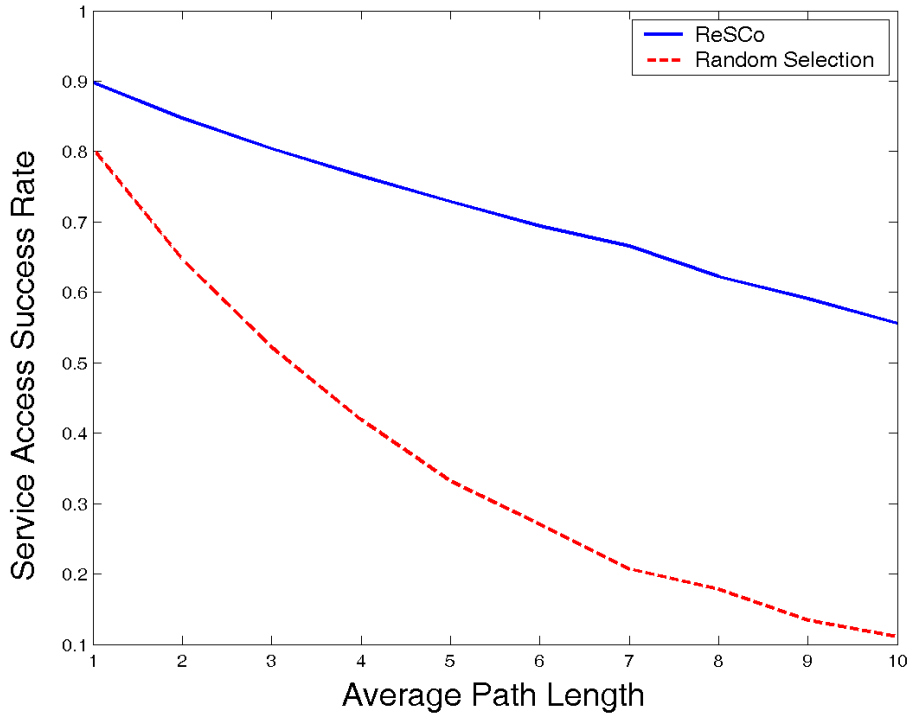

Figure 5.9. Effect of Path Length on Success.

5.3.7 Mobility

Since are being composed in dynamic environments, ReSCo is expected to perform successfully in mobile systems. The simulation uses a random movement model. Each peer has a circular wireless range of 10 meters and moves an average of 1 meter per time step (selected from a Gaussian distribution with a mean of 1 meter and standard deviation of 0.5 meters). The plots found in Figure 5.11 show that ReSCo still performs well in mobile environments, though not quite as well as in static environments. We also show the effect of increasing the percentage of nodes that are mobile (with 50% being attackers) in Figure 5.10. Within a level of tolerance, the success rate in mobile systems stays constant until approximately 90% of the nodes in the system are mobile, but even then the performance does not drop significantly.

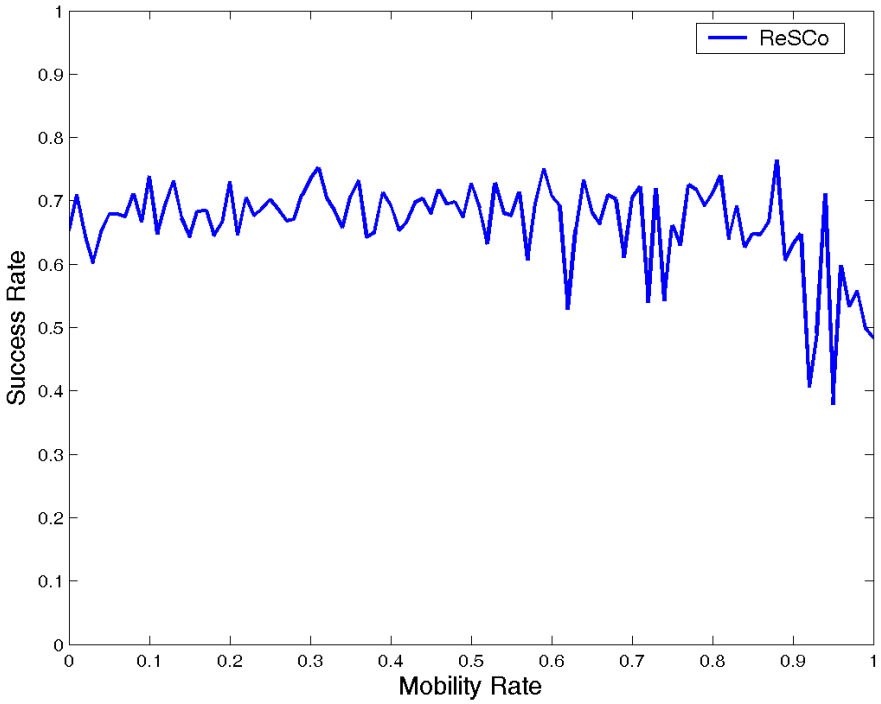

Figure 5.10. Effect of Mobility on Success.

We have also examined the effect of using the best node selection module in mobile systems. The best node module cannot be used exclusively since no exploration would take place in the mobile system, but when used occasionally it can enhance the performance over the default selection modules. In Figure 5.12 the best node module is used when node churn in the system is detected to be under 20% after 200 time steps (done to allow the database to populate first). On average, this approach yielded 6.9% better results throughout the tests. Exactly how to choose when to use the best node approach as opposed to the default approach is being examined as future work.

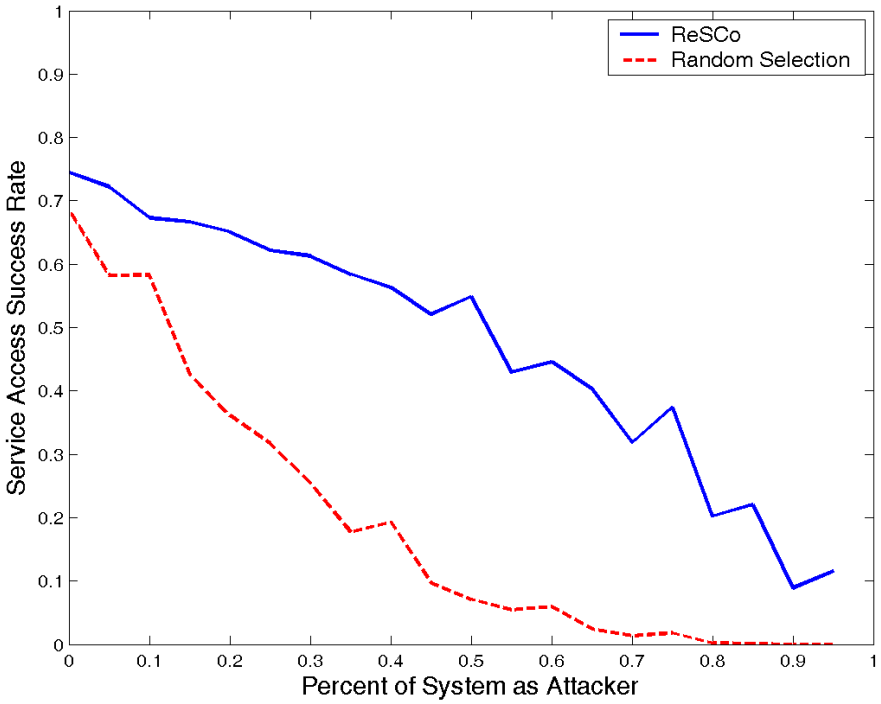

Figure 5.11. Effect of Attackers in a Completely Mobile System.

5.3.8 Reputation

We also examine the effect of exchanging reputation data between nodes. The advantage of including reputation in the system is that nodes can more quickly adapt to the system. Figure 5.13 shows the effect of reputation on ReSCo. When ReSCo is augmented with reputation, it still follows the same general trend as the original system, but is able to maintain a higher level of performance as the number of attackers in the system is increased up until the breaking point (which with the parameters of the system simulated is at 85%) as described above in regards to Figure 5.4. Finally, we added external reputation information to the mobile system. The addition of reputation information provided an average of 19.4% improvement in success as illustrated in Figure 5.14.

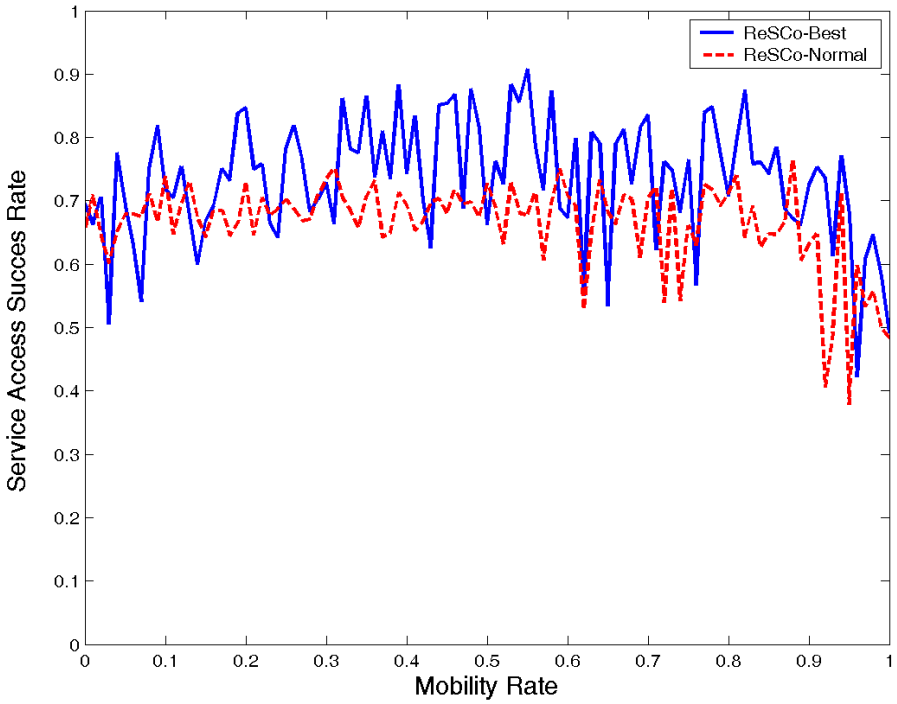

Figure 5.12. Comparison of Stochastic Selection and Best Node Selection in a Mobile System.

5.4 Summary

ReSCo is a modular middleware component for increasing the reliability of service composition in such dynamic systems as P2P, and mobile and pervasive systems. Through stochastic selection that adapts based on personal and reported experiences, ReSCo provides reliability to service composition systems despite the presence of unreliable nodes and attackers. ReSCo can handle selection of compositions paths despite differences in path length. Furthermore, ReSCo is resistant to choke points.

This chapter presented both an analytical model for the performance of ReSCo and simulation results that illustrate the effectiveness of ReSCo. In particular, ReSCo shows significant improvement in reliability over the naive approach against attackers

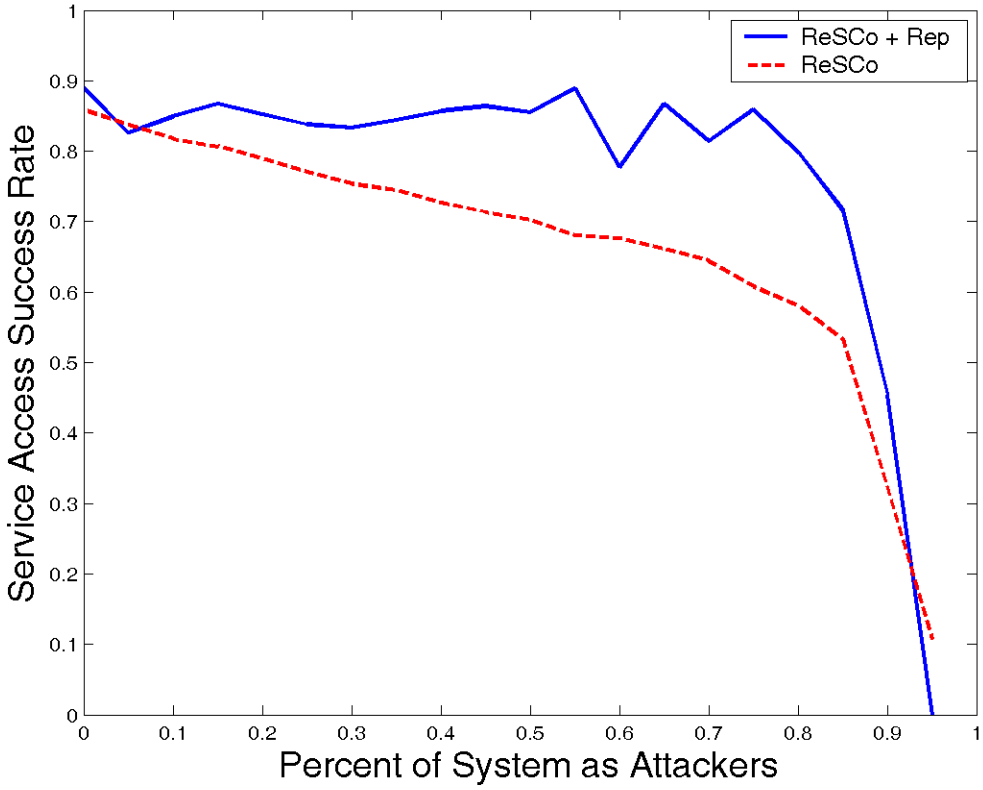

Figure 5.13. Effect of Utilizing Reputation on Success.

(about 700% improvement when half the system attacks), erroneous entities (about 500% improvement when nodes make errors 50% of the time), and in mobile environments (nearly 28% improvement when all nodes are mobile).

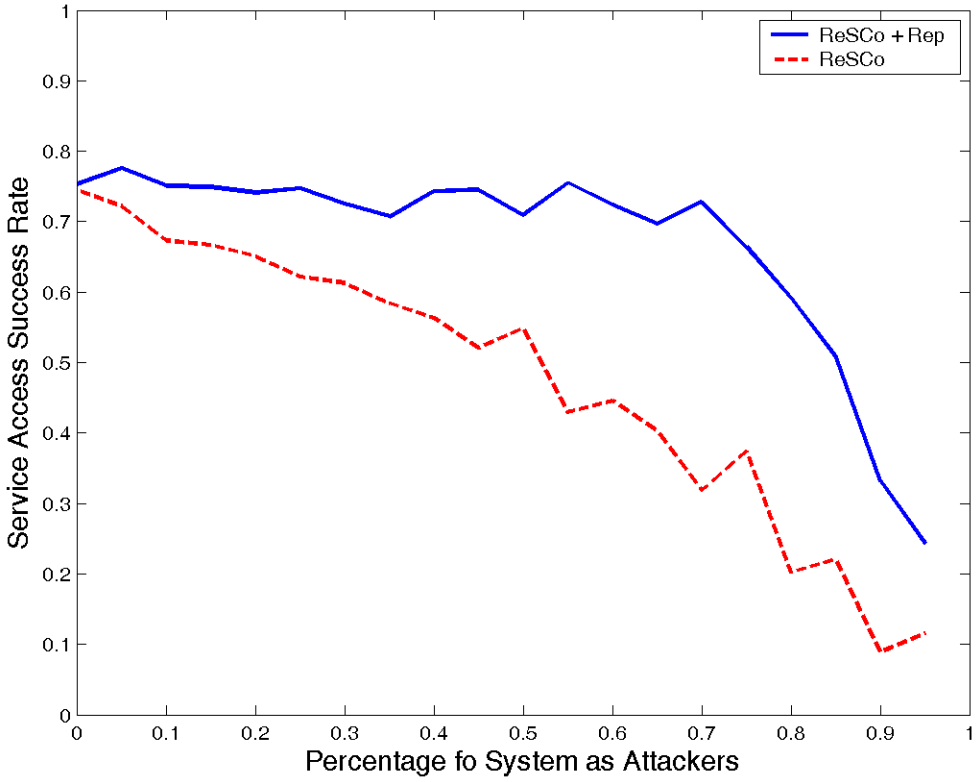

Figure 5.14. Effect of Reputation on Mobile Nodes.

CHAPTER 6

ANALYTICAL EVALUATION OF TRUST MECHANISMS

6.1 Introduction

Trust mechanisms are traditionally tested through simulation or experimentation. This section introduces an extensible framework for mathematical analysis of trust mechanisms. The goal of this section is to introduce metrics for trust mechanisms (not to be confused with metrics for trust). Metrics for trust mechanisms do not focus on how trust is described in the mechanism, but rather they focus on the effectiveness of the trust mechanism in selecting trustworthy resources. This chapter describes two metrics, *accuracy* and *convergence*, and demonstrates their analytical effectiveness by an example analysis of the EigenTrust reputation mechanism [17] in Section 6.4.2.

These trust metrics do not encompass the whole of a trust mechanism. There are many qualitative properties of trust mechanisms such as those discussed for reputation mechanisms in [15]. The purpose of this section is not to reduce the decision of which trust mechanism to use to a mathematical equation, but rather to provide metrics to quantify aspects of trust mechanisms and assist in the selection and development of trust mechanisms for use in different types of systems and against different types of attackers.

The method of evaluation described in this section is designed to analytically describe similar information currently obtained by simulations and experimentation. It is not a complete mathematical analysis of any possible attack or network configuration.

6.2 Evaluative Models

Evaluative models are used to produce a specific metric. This dissertation proposes two evaluative models. One model describes the nodes in the system and another describes the trust data in the system. The system is sufficiently generic so that it can be extended to evaluate complex behaviors and additional aspects of security.

6.2.1 Node Model

The node model consists of several sub-models. These models define the individual node behaviors and the population distributions of these models. There may be many different sub-models since a system may have many classes of attackers or benign nodes.

6.2.1.1 Benign Model

The benign model describes the behavior of benign nodes in the system. Generally these nodes will correctly describe their interactions and correctly answer queries about other nodes; however, this is not always the case and some benign nodes may be faulty in that they sometimes fail with no malicious intent.

6.2.1.2 Attacker Model

Attacker model nodes describe the behavior of malicious nodes in the system. These attackers may be colluding or acting individually. Attackers are described by their behavior in terms of their attack probability and (when relevant) how they report trust information about other nodes at a given system state.

6.2.1.3 Population Model

The population model describes how different classes of nodes compose the system. The population model may be dynamic and change over time. Mathematically, it is described by a distribution function over the size of the system.

6.2.2 Trust Data Model

The trust data model only describes whether or not the data is accessible within the necessary time frame. This means the model can be extended to include many factors of importance such as delay tolerance, mobility, and intermittent connectivity.

6.3 Metrics

Components of trust mechanisms are designed to either increase the accuracy of the mechanism or increase the speed at which the values converge. As a result, the two main metrics introduced are trust accuracy and trust convergence. A secondary metric, $effectiveness$, is also introduced in this section.

6.3.1 Trust Accuracy

Trust accuracy describes how accurately the trust mechanism labels and predicts the actions of other nodes (or resources). A trust mechanism should perform better than random selection for a given attack or it is worthless against that particular attack. Accuracy is a value within the range of [0..1]. A high accuracy value does not necessarily imply secure selection of nodes since all nodes in the system may be faulty or malicious. Rather a high accuracy value implies that the trust mechanism is able to accurately label the level of trustworthiness of nodes.

6.3.2 Trust Convergence

Trust convergence describes how close to the actual convergent trust value a trust value will be within a given time frame. Convergence is a value within the range of [0..1]. For example, in a centralized system such as EBay [23], the convergence value will typically be 1 since the values required for the trust computation are all available in the central node; however, in a system such as Credence [9], the trust information is distributed throughout the system on individual peers. The information must be retrieved and cross-correlated in Credence which means that connectivity disruptions can cause values to be unreliable at a given time. As the convergence of a distributed mechanism may never reach a final value, we will define convergence to describe the point when the residual change drops below a threshold τ. That is to say that a mechanism is τ-convergent when the the change consistently drops below $\tau\%$ of the previous value.

6.3.3 Effectiveness

Based on the two previous metrics, this dissertation proposes a metric of the the effectiveness of a trust mechanism as follows:

$$E = \frac{A \times C}{A_{rnd} \times C_{rnd}} \tag{6.1}$$

Where E is the effect a trust mechanism has, A is how accurately a trust mechanism labels peers, and C is how quickly the trust value converges (necessary for distributed computation).

The trust effect gives a quantitative means to analytically compare the effectiveness of trust mechanisms where no such metric previously existed. Section 6.4.2 provides example of how trust effect can be derived forEigenTrust.

6.3.4 Utility Modeling

Trust effect is useful for comparisons, but it does not provide a complete picture of a trust mechanism. In order to provide a comprehensive view of the trust mechanism, we use utility modeling. Through utility modeling, we can not only incorporate the effect of a trust mechanism into quantitative comparisons, but also model and compare the trust mechanism in different systems with different peer preferences. As a result, rather than saying that a particular trust mechanism fits a scenario better than another trust mechanism because of some qualitative properties, we can show that it fits that situation better quantitatively.

Our utility model of P2P systems comes from the general model described in Section 4.2.3. Since in a comparison between trust mechanisms the expected value of the utility from the system itself (such as cost of entering the system, cost of staying in the system, etc.) will be equal for all mechanisms, this section will only focus only on the utility contribution of the trust mechanism itself. The trust mechanism has four main effects:

1. decreased expected cost of being a victim
2. decreased expected benefit of a successful attack
3. reduction of available resources
4. increased overhead cost

The first two are similar in that they are the values from the general utility equation multiplied by the trust effect. The third factor is the result of error in the trust mechanism that causes a peer to incorrectly refuse resources from a benign peer. The fourth factor is the overhead cost for using the mechanism, expressed as a normalized form of the service, processing, memory, and communications costs added by the trust mechanism, weighted based on the preferences of the peer.

The utility equations for a benign peer and a malicious peer from Section 4.2.3 with the addition of a trust mechanism in the system can be seen in Equation 6.2 and 6.3 respectively, where $TrustAvail$ is the amount of a desired resources available after the trust mechanism is applied and $TotalAvail$ is the amount that would have been available without the trust mechanism.

$$\begin{aligned} Utility = BenignBenefit \times \frac{TrustAvail}{TotalAvail} \\ -BenignCost - VictimCost \times TrustEffect \\ -TrustOverhead \end{aligned} \tag{6.2}$$

$$\begin{aligned} Utility = MalBenefit \times TrustEffect - MalCost \\ -BenCost - TrustOverhead \end{aligned} \tag{6.3}$$

6.4 Example

In this section we present two examples of the use of these trust metrics. First is a mechanism presented in this dissertation, AREX, and second is EigenTrust. EigenTrust is examined in more detail than AREX to show how these metrics and utility equations can be used to evaluate and predict performance.

Table 6.1. Trust Metrics

TrustAcc	Accuracy of the trust mechanism
TrustConv	Convergence of the trust mechanism
TrustEff	Effectiveness of the trust mechanism

6.4.1 AREX

This example provides an examination of the metrics as they pertain to AREX. AREX can be examined more simply than typical trust mechanisms because it lacks the complexity involved with handling corrupted collaborative information. The accuracy of AREX is based on the estimate of the attack rate and the attack rate of the system at the time of the next request as shown in Equation 6.4. Since AREX does not rely on the communications of other nodes, it converges to its evaluative value instantly. As a result, AREX's estimation of the system's reliability is the only contributing factor to error, thus eliminating vulnerability to attacks that are based on disrupting information flow.

$$TrustAcc = \left| \frac{P_{att} - P'_{att}}{P_{att}} \right| \tag{6.4}$$

$$TrustConv = 1 \tag{6.5}$$

6.4.2 EigenTrust

In this example, we will analyze the EigenTrust[17] trust mechanism. We begin by modeling the $TrustAcc$ and $TrustConv$ components of the equation. Since trust is considered transitive in EigenTrust, the effect of any given peer's opinion on the calculation of the global trust value is equal to the difference from the real value times the trust that the peer has obtained, as shown in Equation 6.6. The rate of convergence of the trust value in EigenTrust can be described by the rate of convergence of the distributed eigenvector calculation. This is shown in Equation 6.7.

$$TrustAcc = \sum Eigv(R)_i \times |TrustR_{i,j} - ActR_{i,j}| \tag{6.6}$$

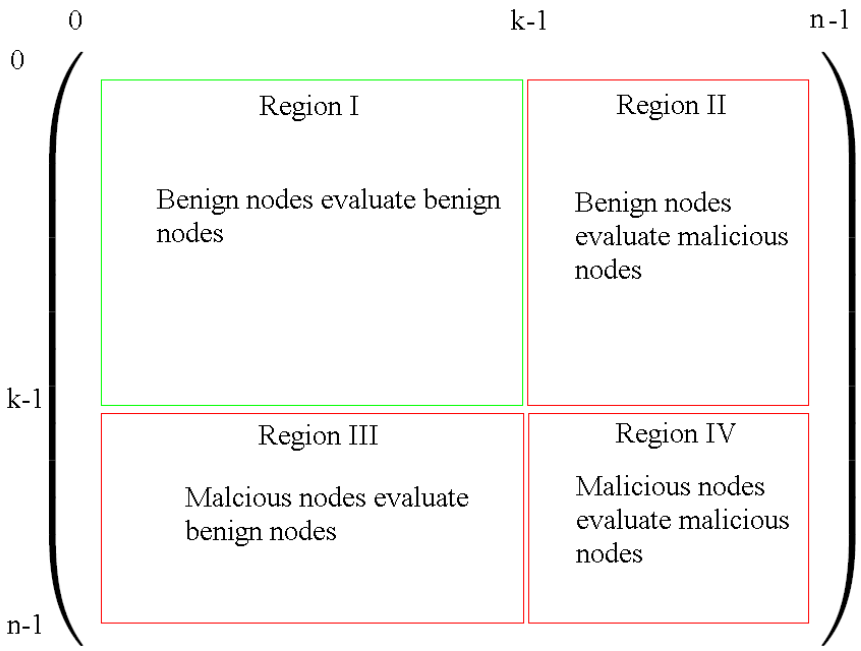

Figure 6.1. EigenTrust Matrix.

$$TrustConv = \frac{E}{N^{N-1}} \times \frac{\lambda_2}{\lambda_1} \tag{6.7}$$

Where the trust accuracy, $TrustAcc$, is computed with $Eigv(R)_i$ as the global reputation value for peer i which is the i^{th} element of the principal eigenvector of matrix R. $TrustR_{i,j}$ is the reported reputation of j by i and $ActR_{i,j}$ is the actual reputation that should have been reported for j by i.

The trust convergence, $TrustConv$, is computed with E being the number of unique peers previously interacted with (included pre-trusted peers) and N peers in the system to compute the average local network size per peer and λ_2 and λ_1 are the second and first eigenvalues of the normalized reputation matrix (since the matrix is a transition matrix, λ_1 will always be 1).

In the discussion that follows, all parameters used follow the simulation setup described in [17] unless otherwise noted. In the case that we were uncertain about a parameter needed for our calculations, we have noted our estimate of that value.

In order to compute a residual graph as shown in the EigenTrust paper, we take the matrix convergence and raise it to increasingly larger powers to calculate how much of the residual remains after each exchange of reputation information as shown in Figure 6.2. In comparing our analytical results to the simulation results of EigenTrust[17], we note that we have estimated an average coverage of 20% of the system and a second eigenvalue of 0.465. Our results are similar to those from simulation data, though the simulation shows a minutely quicker convergence than our analytical results.

The accuracy of EigenTrust is dependent on the the amount of reputation a peer can obtain times the amount it can deviate its opinion from the truth. Hence, the goal of an attack (particularly a collusive attack with $k - n$ attackers as based on Figure 6.1) against an EigenTrust system is to maximize the average value of Region II from Figure 6.1 (which describes an EigenTrust matrix with n peers and k benign peers) in order to maximize the effect of Regions III and IV (which are easily controllable by an attacker). Likewise, the reputation value will converge largely based on the amount of interaction in the system along with the convergence rate of the eigenvector, as noted in [43]. If each peer in the system has interacted with every other peer in the system, then the system is fully connected and $E = N \times N - 1$, resulting in the only factor of convergence being the second eigenvalue of the reputation matrix. As a result, we see that the trust effect from EigenTrust would degrade when applied to sparsely interacting systems.

We can analytically produce results that approximately match the simulations done by [17]. Since the attack of a naive individual is not an interesting case in trust

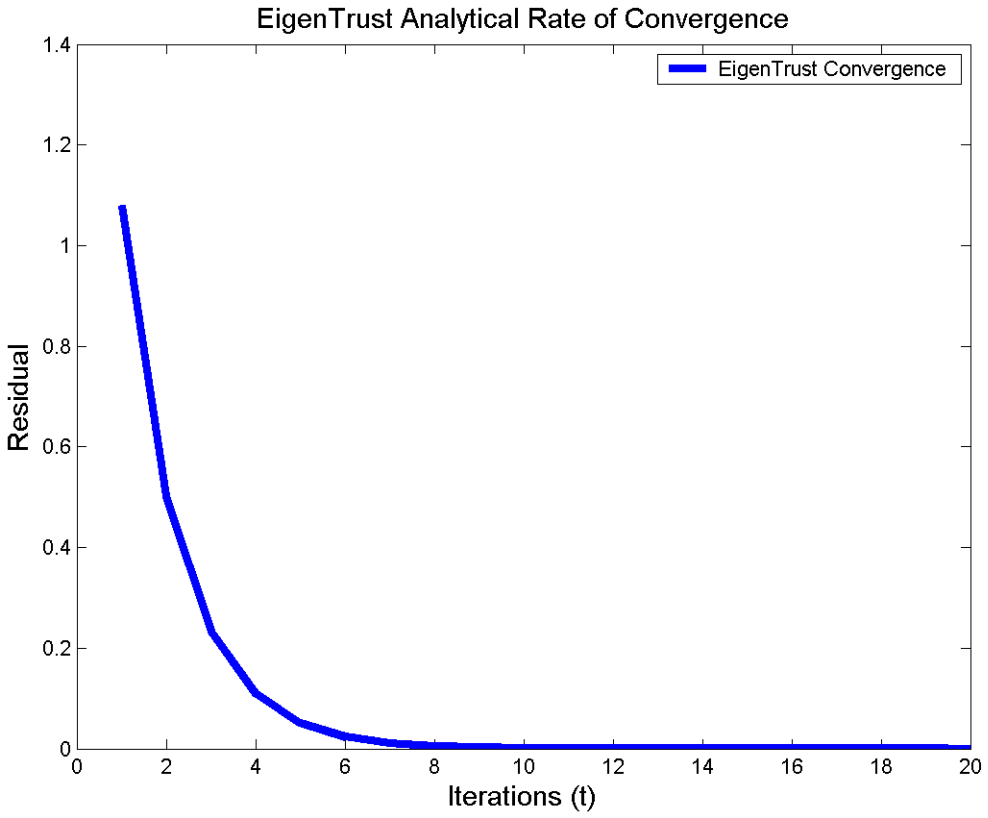

Figure 6.2. EigenTrust Remaining Residual.

mechanisms, we will focus our discussion on collusive attack strategies. In all cases, Region I will always be populated (before normalization) with the average value of $Participation(t) \times (2 \times Quality - 1)$ where $Participation$ is the average number of requests by benign peers to benign peers in the system up through time t and $Quality$ is the average quality of benign peer uploads (Kamvar, et al.[17] set this value to 0.95 to account for the fact that benign peers sometimes make mistakes). The reason for the term $2 \times (Quality - 1)$ is that for every negative transaction, the peer will have its score reduced, rather than just not increased. This function produces the expected rating of a benign peer by another benign peer at a given time. In the collusive

attacker models, attackers set benign peer ratings to 0, so Region III will be all be 0, and attackers set each others' reputation to 1.

The most interesting region for studying collusive attacks is Region II. This is the region that defines to what extent benign peers trust malicious peers. The first collusive attack simulated in EigenTrust involves all malicious peers always attacking. As a result Region II will become 0 and effectively render the attackers useless when the eigenvector is calculated and as a result we can analytically produce approximately the same results as the EigenTrust simulations. The second attack involves attacking at a rate of $f\%$. Analytical evaluation of this attack becomes slightly more difficult because of the probabilistic component; however, we can sum the expected reputation of the average peer in Region II over the course of the system run where the expected reputation at time t is defined by Equation 6.8 where i is the current reputation of a peer, f is the rate at which the malcious peer attacks, and r is the rate at which malicious peers interact.

$$ExpectedTrust(t) = \sum i \times Pr(i,t) \tag{6.8}$$

$$\begin{aligned} Pr(i,t) = {} & Pr(i-1,t-1) \times (1-f) \times r \\ & + Pr(i,t-1) \times (1-r) \\ & + Pr(i+1,t-1) \times f \times r \end{aligned} \tag{6.9}$$

In Figure 6.3 we model the most effective attack against EigenTrust, in which peers divide the labor of obtaining good reputations and attack. To do this we identify two sets of peers, D and B. Peers of type D obtain high reputations and report that the attackers, type B, are highly reputable. In order to validate the model, we recreated similar results by solving for the eigenvector of the reputation matrix in order to provide a steady state analysis (which is what EigenTrust simulations appear to try to do by dropping the results from the first 15 query cycles of each simulation).

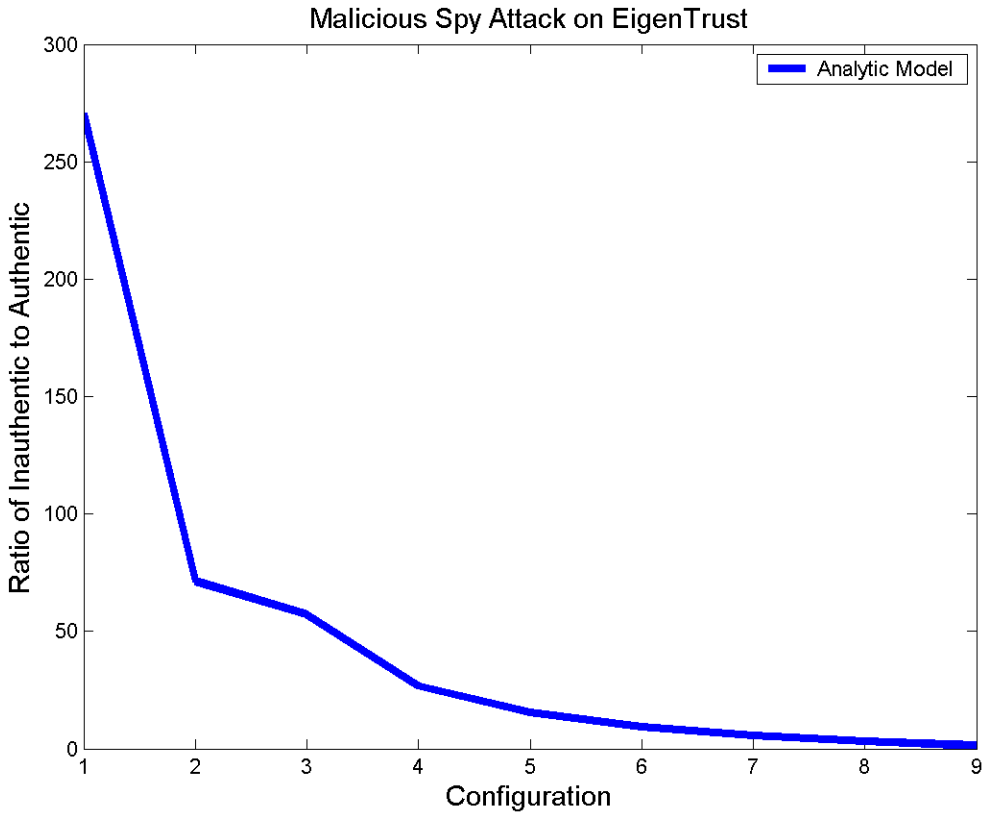

Figure 6.3. Malicious Spy Attack.

In Figure 6.4 we show the effect of increasing the number of peers in an EigenTrust system while holding constant the average rate of connectivity between peers. For this figure we start with a λ_2 value of 0.65, a fully connected system ($E = N \times (N-1) = 9900$), and a perfectly accurate reputation report ($TrustAcc = 1$). As a result, it is obvious that EigenTrust is effective in reasonably well-connected environments; however, it quickly degrades in quality with an increase in the number of information exchanges needed for the global reputation values to converge.

It is noted that the point of this exercise is to show that a reputation mechanism can be broken down analytically. As the entire details of every simulation result in

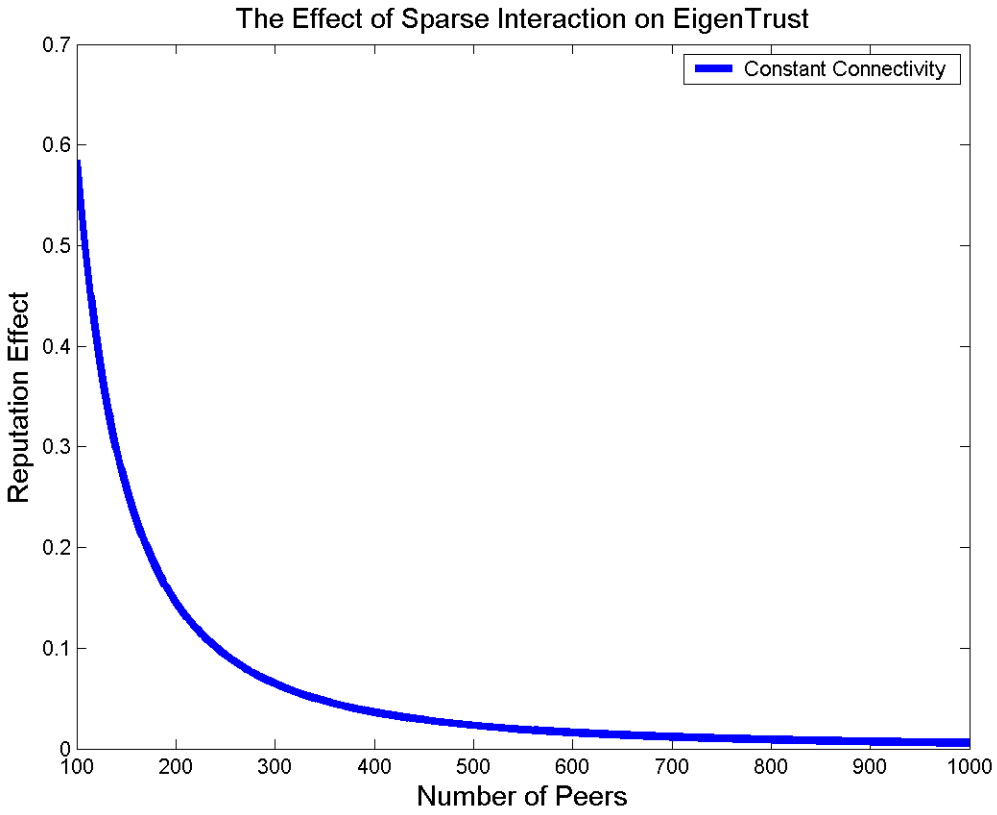

Figure 6.4. The Effects of Sparse Connectivity on EigenTrust.

[17] were not available, we made some assumptions to compute results. In general the pattern of the plots in Figures 6.2, 6.3, and 6.4 are similar to those in [17]. Through the proposed model, developers and researchers can compare and evaluate their reputation mechanisms, provided the appropriate analytical equations and algorithmic descriptions are available.

6.5 Summary

This chapter presented metrics for quantitative analysis of trust mechanisms which allows researchers to compare trust mechanisms and predict performance and

vulnerabilities. These metrics are created to better understand the implications design choices and enable developers to determine best trust mechanism for their applications.

CHAPTER 7

SOCIAL AUGMENTATION OF TRUST

7.1 Introduction

Pervasive computing systems are invisible systems, oriented around the user. As a result, many future pervasive systems are likely to include a social aspect to the system. The social communtities that are developed in these systems can augment existing trust mechanisms with information about pre-trusted entities or entities to initially consider when beginning to establish trust. Social network information can provide a starting point for AREX to produce improved results.

For example, the Collaborative Virtual Observation (CoVO) system must accomplish virtual observation efficiently whilst protecting the data from corruption from unknown remote nodes. When an event of interest occurs, the given infrastructure (bus cameras, etc.) may not sufficiently cover the necessary information (be it in space, time, or sensor type). To enhance observation of the event, infrastructure is augmented with information from sensors in the environment that the infrastructure does not control. These sensors may be unreliable, uncooperative, or even malicious. Additionally, to execute queries in soft real-time, processing must be distributed to available systems in the environment. Social Trust (SoTru) can be used to satisfy these requirements.

This chapter focuses on utilizing social network information to augment trust-establishment techniques. The system uses social behavior of nodes to predict a subset that it wants to query for information. In this context, social behavior may include behavior such as transit patterns and schedules (which can be used to determine if

a queried node is likely to be reliable – that they actually know the area well that they are in) or known relationships, such as a social network, that can be used to determine networks of nodes that may also be able to assist in retrieving information. Neither implicit nor explicit relationships necessarily imply that the user trusts an entity, but rather will provide a starting place for establishing trust. This chapter proposes a general framework for utilizing social network information to assist in trust establishment and apply that framework to the problem of utilizing sensors controlled by third parties to assist in sensing events.

Increases in network connectivity and the increasing pervasiveness of computing resources have led to more social applications of computing. Many applications focused on explicitly enhancing this social aspect in the form of social network websites [44, 45, 46, 47] have become popular in recent years. Much research has been conducted on analyzing and establishing trust in social networking sites [48, 49, 50], but little has been done on utilizing information in social networks to establish trust in other domains. Some trust models [2] include a social context in the model for establishing trust, but no system has focused on using social information to assist in the trust-establishment problem.

In this context, a social network is defined as the set of relations connecting multiple entities in a system. These relations may be either explicit or implicit. Explicit relations are stated as in a social networking websites such as Facebook [44] where people explicitly and mutually agree that they are friends. Implicit relations are implied, often by context information or recurring mutual patterns, such as people that spend overlapping time in the same coffee shop on a regular basis. Relations in a social network do not imply that trust exists. For example, in social network sites, people often become friends with people that they do not know or with whom they are only acquaintances. Likewise, regular patterns that are discovered as implicit

social relations in systems do not imply trust. SoTru provides this information as a starting point to bootstrap the trust establishment process.

The purpose of establishing trust using social information is to enhance the reliability and coverage of a collaborative virtual observation system. The goal of collaborative virtual observation is to provide anywhere, anytime access to events occurring in a dynamic environment. Collaborative virtual observation combines dynamic service composition [37] with virtual observation [51] to utilize existing infrastructure with third party services available in the environment. Since these services are not under the control of the system, they may be unreliable or malicious.

There are many ways that trust is established. Some approaches use digitally-signed certificates to guarantee that the owner of the certificate can be trusted for some task or that the owner fills some role in the system. This approach requires an authority that can issue and guarantee the validity of these certificates. Furthermore, it requires users to obtain the certificates. Obtaining a certificate for each service that is necessary in a highly dynamic system would consume the user's attention and be counter to invisible operation. Another common way of establishing trust is to use reputation. Reputation is more flexible and scales better in dynamic environments. AREX, described in Chapter 4 scales well to highly dynamic systems. AREX uses a game theoretic approach to motivate attackers to attack less often. It also does not suffer the same vulnerabilities as reputation mechanisms, such as vulnerability to intermittent connectivity, startup/traitor attacks, and being highly connected to malicious entities. The downside is that in large systems, such as performing collaborative virtual observation in a major urban area, AREX adapts to the system slowly. SoTru incorporates social network information into AREX and thereby overcome performance issues caused by large systems.

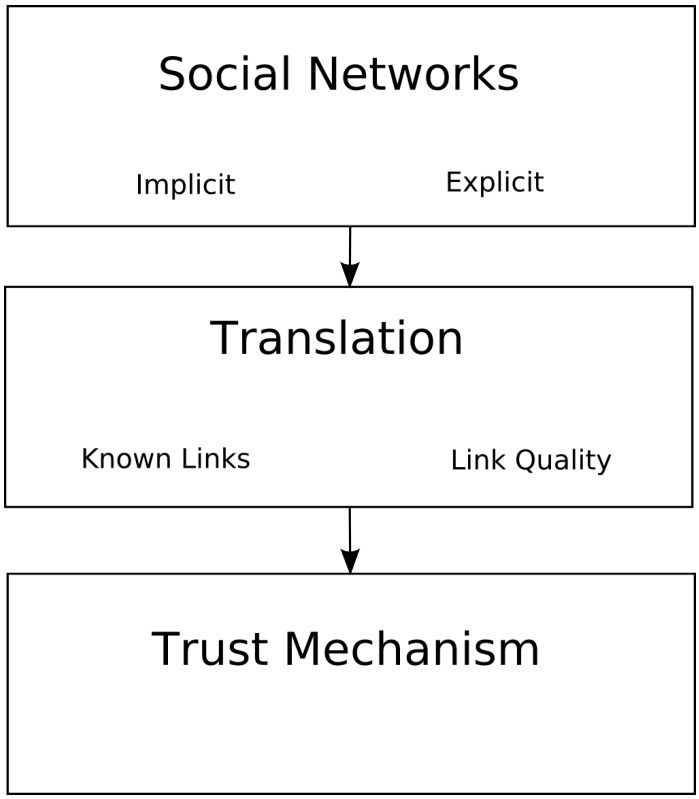

Figure 7.1. Social Augmentation Framework.

7.2 Design

This section introduces the Social Augmentation Framework for utilizing relations in social networks to bootstrap trust algorithms. It also describes the application of this framework to collaborative virtual observation.

7.2.1 Social Augmentation Framework

This section describes the Social Augmentation Framework as shown in Figure 7.1 and the interactions of each component. The Social Augmentation Framework consists of three components: Social Networks, Translations, and Trust Mechanisms.

7.2.1.1 Social Networks

Social Networks are a set of data sets of social connectivity information that are available to the system. These may take the form of web-based social networking sites, contact lists in a cell phone, or databases of traffic information that can be mined for links.

7.2.1.2 Translation

The Translation component is responsible for translating information from the Social Network into a usable form for the Trust Mechanism. This may be as simple as identifying connected nodes or as complicated as analyzing connections to provide initial values for nodes to the trust mechanism. The Translation component also is responsible for resolving nodes in the social network into addressable nodes in the system. Section 7.3 describes a translation component that assigns an initial starting experience value for AREX based on membership in the social network.

7.2.1.3 Trust Mechanism

The Trust Mechanism requests initial information to assist in bootstrapping the request for trust computations. Additionally the Trust Mechanism can make requests to the Translation component to augment current calculations.

7.2.2 Social Augmentation of CoVO

Collaborative Virtual Observation (see Figure 7.2) involves utilizing resources from pre-existing infrastructure in addition to opportunistic access from services and resources available in the environment such as cell phone cameras and processors. Observations made by these resources are then stitched together using available ser-

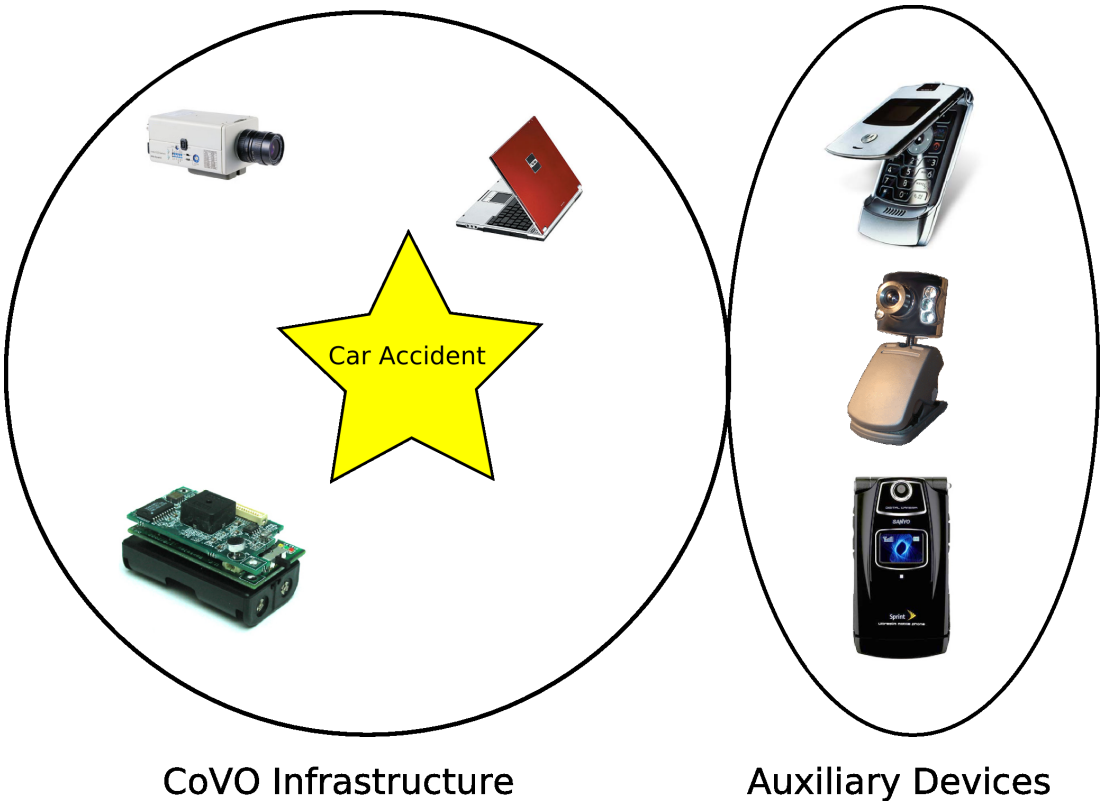

Figure 7.2. CoVO Example.

vices based on variables such as time and space. The result is a virtual observation of events that can be accessed in soft real-time. To accomplish this though, the resources and services from the environment must provide valid services; otherwise, the resulting virtual observation may be worse than would be the case with just the available infrastructure. Existing trust mechanisms do not achieve the soft real-time requirements in the dynamic environments in which collaborative virtual observation is performed; however, by augmenting trust mechanisms with social information, the performance of the trust mechanism can be enhanced to meet the requirements in many cases.

SoTru uses social augmentation to assist with the trust computations to achieve soft real-time performance for CoVO. SoTru utilizes primarily implicitly formed social

networks based on patterns derived from the observation of bluetooth signals by mobile nodes. The physical presence of an entity provides an implied social network since the entity is connected in terms of physical presence in a place in the society. An assumption contributing to this decision is that if a node consistently appears in a particular area for a long period of time that it is more likely to stay in the area while it is needed. For example, if a node is consistently in the Computer Science Department building for about 6 hours a day from Monday through Friday, there is a strong chance that it will remain in the area to provide its service during that regular pattern. An argument can be made that forming a consistent physical pattern is of high cost to an attacker and may be less likely, but just because a social network is used, it does not imply that there is trust between the nodes, just that the trust mechanism will augment its approach with information from the social network. Therefore, an actively malicious node would still have to undergo the same calculations from the trust mechanism as it would without the social network, just at a higher cost to itself.

SoTru uses the available nodes that are expected to remain in the area based on previous traffic patterns mined from the traffic databases to seed the trust calculation process. The seeding is made possible by the translation component which selects a predetermined number of social nodes to use in the initial trust computation of the trust mechanism. SoTru uses adaptive trust mechanisms for resources access and service composition based on AREX [42]. These trust mechanisms work best in smaller systems, so operating on a city-wide basis can slow the adaptation of the mechanism. The initial seeding from the social network provides the trust mechanism with the ability to reliably operate on a much smaller scale and provide similar reliability in less time.

7.3 Evaluation

The system was evaluated via simulation. The purpose of this simulation was to show how social network information can be utilized to assist in the establishment of trust between entities, even when the social network includes entities that are not trustworthy. The simulations were primarily used to test the question of how reliable the social network must be to be beneficial to the system. The simulation was run with the same simulator and setup as AREX in Chapter 4. Preference ratings are accomplished by converting social network membership into initial experience values for nodes in the system. Unless otherwise noted, the size of the social network used was 20 nodes.

Table 7.1. Preference Values

Preference	Initial Experience Value
No Preference	1
Slight Preference	10
Strong Preference	100
Very Strong Preference	1000

The first experiments were run to show that social network information can be used to improve the response time of adaptive trust mechanisms. Figure 7.3 shows the average utility produced at a given time step averaged over 100 simulations. In this simulation, the social network was 95% trustworthy. The figure shows that the more preferred the social network is, the quicker the utility converges to its final value. Figure 7.4 shows the average utility produced per step by a given time step averaged over 100 simulations. The figure shows similar results to Figure 7.3, but provides the cumulative viewpoint rather than the per-step results.

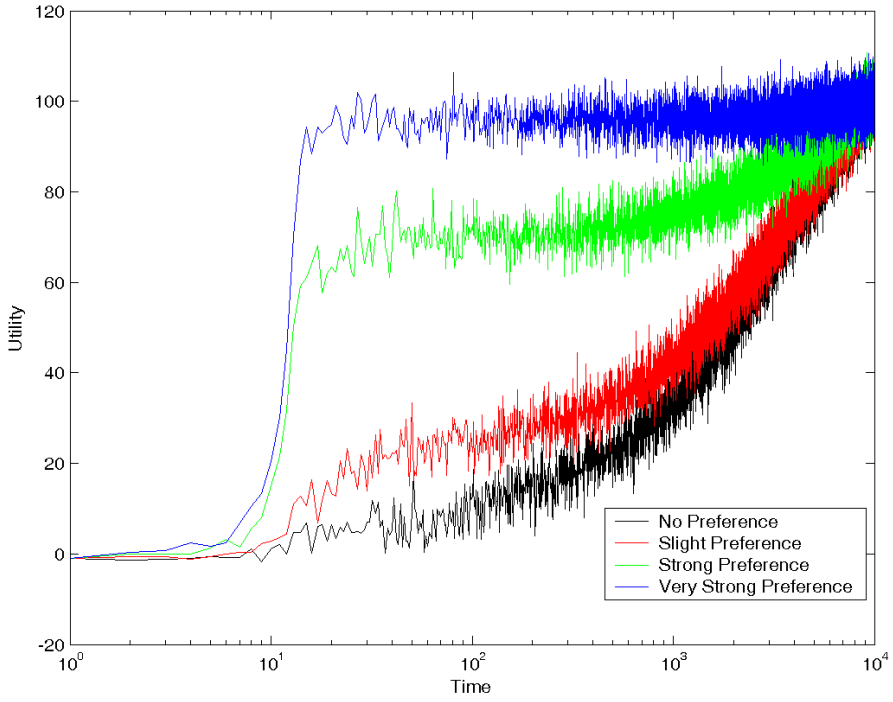

Figure 7.3. Effect of Social Network Preferences on AREX (Per Time Step).

Since it is possible that a social network could be compromised, the next experiments measure the effectiveness of using social networks to augment adaptive trust mechanisms when the social network itself consists of untrustworthy nodes. Figures 7.5, 7.6, and 7.7 show how AREX responds to trust augmentation of a faulty social network. The values were taken after 1000 time steps. Two conclusions can be drawn from this experiment. First, social networks should be strongly preferred when there are large numbers of attackers in the system. When the system has a large portion of attackers, giving strong preference to the social network will prevent exploration of nodes that are likely to attack. Second, social networks should be strongly preferred when the social network is reliable. In general, the assumption can be made that

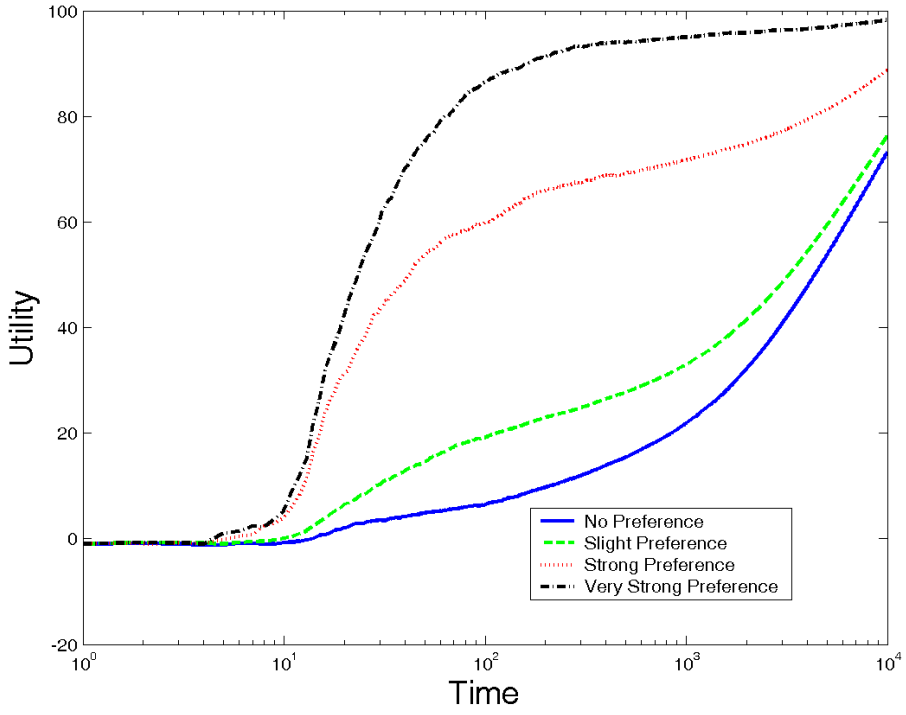

Figure 7.4. Effect of Social Network Preferences on AREX (Cumulative).

social networks should be preferred when the social network is more reliable than the overall system.

Figure 7.8 shows the effect on utility for a variety of system sizes with half of the system as malicious. Once the system size grew above that of the social network, the performance grew along with the reliability of the social network. While the smaller systems had higher utilities after 1000 time steps, the larger system had closer performance than when the social network was not used.

The next experiment tested the hypothesis that the cost of malicious nodes in the social network could be greatly diminished by using exponential backoff of experience values rather than incremental backoff (as used in the AREX and previous SoTru simulations). In this case, for each attack that was detected, the node lost half

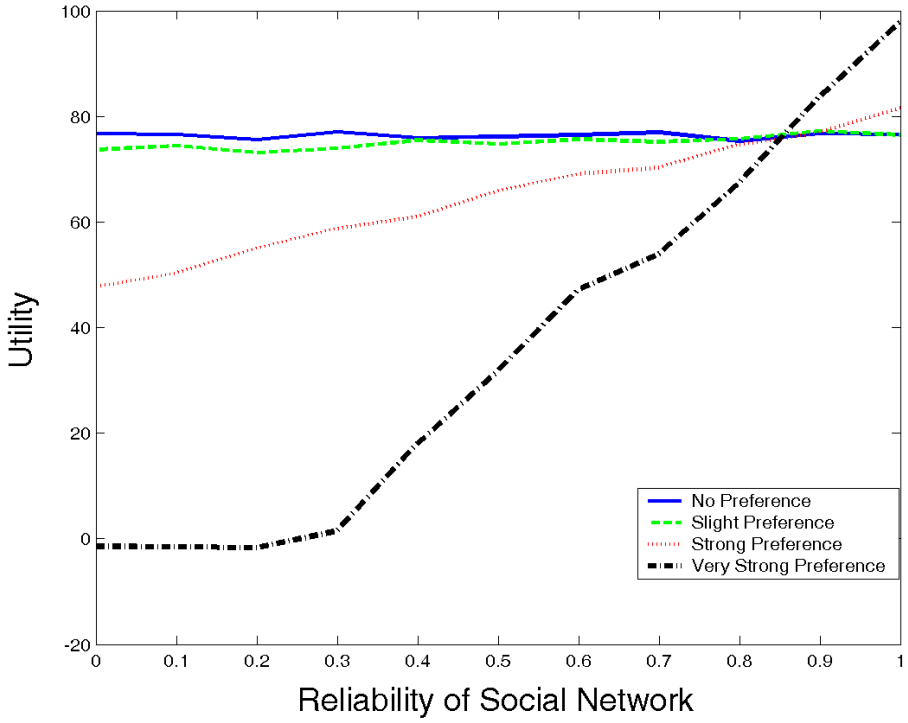

Figure 7.5. 20% of the System is Attackers.

of its experience value. The simulation provided evidence that this approach is an effective method for dealing with untrustworthy social networks. Figure 7.9 shows that positive utility was achieved with exponential backoff regardless of how reliable the social network was (as opposed to the incremental backoff, which only achieved positive utility when nearly half or more of the social network was reliable).

7.4 Summary

Social networks are becoming increasingly available alongside computing resources. SoTru is a mechanism that allows trust mechanisms to enhance their performance through utilization of social information. This chapter has shown how SoTru can be used to increase the adaptation rate of of AREX and improve its performance.

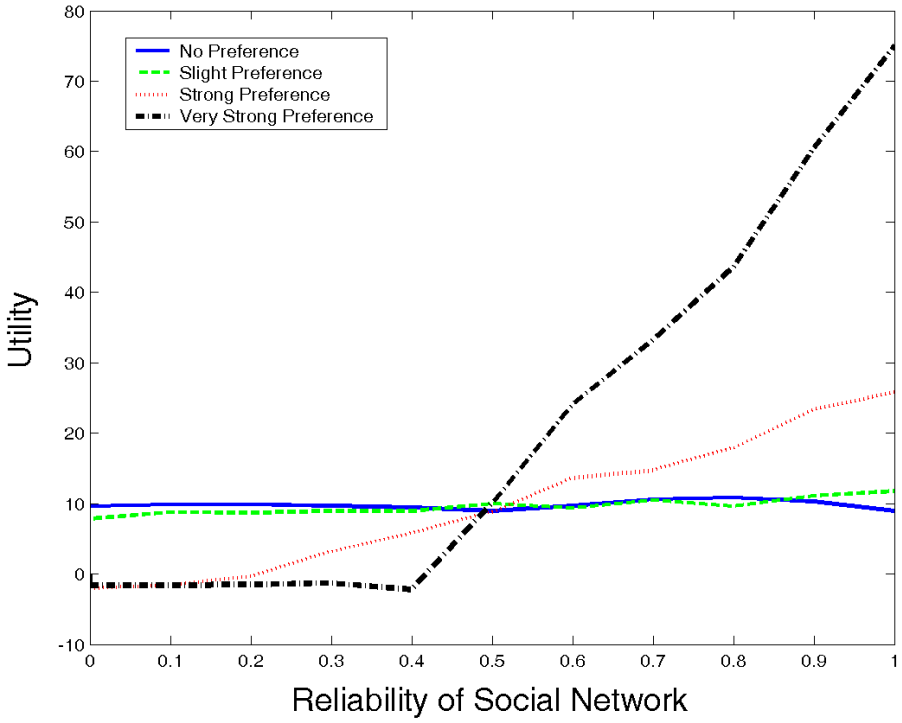

Figure 7.6. 50% of the System is Attackers.

This chapter introduced SoTru, a framework for augmenting security mechanisms with social network information. SoTru enhances security mechanisms by providing a starting point for them to use to begin establishing trust. The chapter also applies social network information to collaborative virtual observation and performs an evaluation of the SoTru by examining its ability to handle the challenges of applying social network information to trust. In the simulation study, SoTru provided significant performance increases when the reliability of the social network was greater than that of the system as a whole.

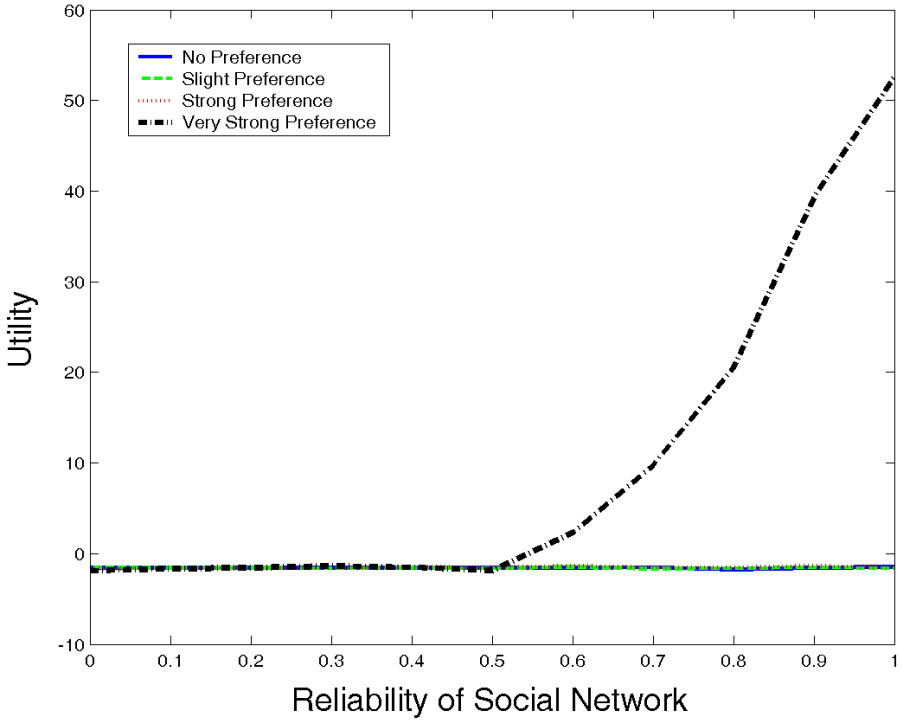

Figure 7.7. 80% of the System is Attackers.

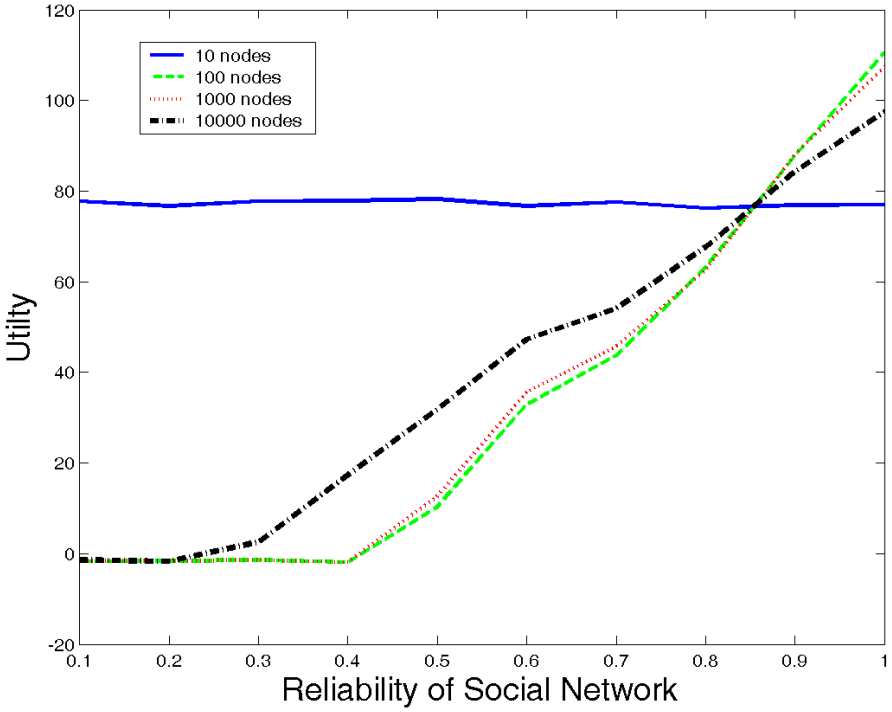

Figure 7.8. Effect of Malcious Nodes in the Social Network.

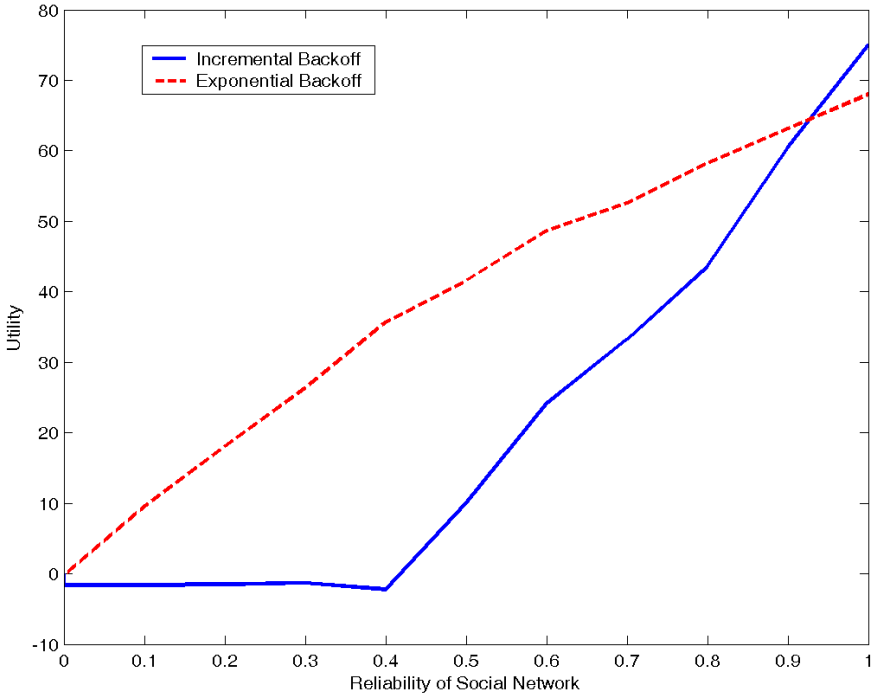

Figure 7.9. Effect of Backoff Techniques when Attacked.

CHAPTER 8

CONCLUSION

Advances in technology are increasing the widespread use of mobile computing devices and the distributed systems that are being developed on them. These systems must be able to operate seamlessly despite factors such as mobility, heterogeneity, openness, and device constraints. In addition to these challenges, dynamic systems must operate despite malicious users and unreliable service that could ruin the user's experience and be very costly to the user. To address these challenges, trust mechanisms must be developed that can operate in dynamic systems. Despite many different definitions of trust, trust mechanisms generally work by identifying nodes, services, and resources that operate as the user expects. There are many different mechanisms for establishing trust, but current work is either inflexible or focuses on collaborative efforts that involve the use of a third party to determine trust.

8.1 Summary Of Contributions

This dissertation developed and evaluated mechanisms for enhancing the usability and security provided by trust mechanisms. The technical contributions of the dissertation are encapsulated in four systems: the DTT, AREX, ReSCo, and SoTru. These systems focus on easing the development, deployment, and use of trust mechanisms while improving security and reducing costs in dynamic systems.

The DTT is a framework and system for the development and deployment of trust mechanisms. The DTT enhances the modularity and extensibility of trust mechanisms by abstracting them into three components: Presentation, Computation, and

Communication Protocol. Known as trust blocks, these mechanisms run on a DTT daemon which is designed to provide platform independence of trust mechanisms. The daemon handles the interface between the application and trust block, storage of data, and protocols of the network interfaces. DTT allows for incremental deployment since trust blocks are interoperable with pre-existing trust mechanisms.

AREX is a system for adaptive resource exploration. AREX uses a game theoretic strategy to test peers in the system using exploratory requests. By testing with exploratory requests, AREX incurs no additional cost when under attack and is able to determine which peers are acting maliciously. AREX approximates a Nash equilibrium strategy (in terms of the probability of sending an exploratory request instead of a real request) against strategic attackers. If non-strategic malicious peers or benign peers exist in the system, then AREX will adapt to utilize the most reliable of those peers. Thus, AREX approximately achieves a worst-case performance as the result of the Nash Equilibrium and can adapt to perform better whenever possible. AREX accomplishes this without relying on collaborative information such as reputation; it avoids the vulnerabilities associated with relying on third parties in trust establishment.

ReSCo is a system for reliable service composition. Like AREX, it is designed to operate without requiring third party information to provide security; however, it can incorporate external information to accelerate the adaptation process. ReSCo utilizes a stochastic algorithm for selecting a path of composition (when there are multiple methods for composing the same service) and the individual nodes on the path (when there are multiple nodes providing the same services). The selection is based on previous experience with the nodes in the system. ReSCo uses a stochastic selection mechanism because, in dynamic systems, it is likely that the available paths and nodes are frequently changing. Furthermore, it lowers the likelihood of a single

point of failure in the system (compared to only deterministically selecting the path and nodes that are perceived as best).

SoTru is a framework for utilizing information from social networks to help establish trust in other systems. Adaptive systems such as AREX or ReSCo require time to adapt to achieve peak performance in a system and other systems such as reputation mechanisms often rely on pre-trusted peers to assist in the trust establishment process. SoTru converts relationships in both implicit and explicit social networks to be used by trust mechanisms.

8.1.1 Applications

The application domain for the research discussed in this dissertation continues to grow as the technology used is shown to be more reliable. While AREX, the DTT, and ReSCo were developed as underlying mechanisms and are currently application-dependent, they can be extended for use in a variety of applications. AREX has been shown to be effective in both distributed computation and file sharing applications. The DTT was originally developed to assist in establishing trust for an Adaptive Media System [52]. Both ReSCo and AREX have applications in Collaborative Virtual Observation, which extends to broad applications in emergency, military, and physical security domains.

8.2 Future Direction

Many areas of future research can stem from this dissertation. This work raises many questions about the limits of security that can be provided in dynamic systems without utilizing a third party. While AREX is able to motivate attackers to attack less often through a game theoretic approach, there may be other individualistic mechanisms to protect users from malicious nodes. AREX and ReSCo both have the

ability to incorporate third party data into their calculations, but this dissertation does not explore the question of when it is better to act as an individual and when it is better to collaborate with other nodes in the system to determine whom to trust.

The DTT opens many potential research directions in trust information sharing and trust group formation. This dissertation suggests that trust information can be reused, even if it was not generated by the same trust mechanism, but no formal mechanism exists for doing so. Trust groups are assumed to be a pre-existing trusted relationship, such as devices forming a PAN, but risks and rewards of forming and maintaining trust groups from nodes that are not trusted a priori have not been explored.

AREX, ReSCo, DTT, and SoTru are designed for dynamic systems, but they have applicability in other types of systems. Because of their modularity and low overhead costs, these systems are useful to a wide variety of systems such as sensor systems and cyber-physical systems. Further research will explore appropriate applications and adaptations of these systems in additional system classes.

Successful research in this area will enhance the security and usability of dynamic systems, allowing them to fade into the background of the user's life. Users will experience the benefits of applications built on dynamic systems without having to deal with the hassles and vulnerabilities that are seemingly inherent in dynamic systems.

REFERENCES

[1] S. P. Marsh, "Formalising trust as a computational concept," Ph.D. dissertation, University of Stirling, Apr. 1994.

[2] L. Xiong and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, 2004.

[3] D. H. McKnight, L. L. Cummings, and N. L. Chervany, "Trust formation in new organizational relationships," University of Minnesota, MIS Research Center Working Paper WP 96-01, 1996.

[4] B. Barber, *The Logic and Limits of Trust*. Rutgers University Press, NJ, USA, 1983.

[5] L. Kagal, J. Undercoffer, F. Perich, A. Joshi, and T. Finin, "A security architecture based on trust management for pervasive computing systems," in *Grace Hopper Celebration of Women in Computing*, Oct. 2002.

[6] G. D. M. Serugendo, "Trust as an interaction mechanism for self-organising systems," in *ICCS*, 2004.

[7] G. Sampemane, P. Naldurg, and R. H. Campbell, "Access control for active spaces," in *ACSAC*. IEEE Computer Society, 2002, pp. 343–352.

[8] L. Xiong and L. Liu, "Building trust in decentralized peer-to-peer electronic communitties," in *International Conference on Electronic Commerce Research*, 2002.

[9] K. Walsh and E. G. Sirer, "Experience with an object reputation system for peer-to-peer filesharing," in *NSDI*. USENIX, 2006.

[10] B. Raman, S. Agarwal, Y. Chen, M. Caesar, W. Cui, P. Johansson, K. Lai, T. Lavian, S. Machiraju, Z. M. Mao, G. Porter, T. Roscoe, M. Seshadri, J. S. Shih, K. Sklower, L. Subramanian, T. Suzuki, S. Zhuang, A. D. Joseph, R. H. Katz, and I. Stoica, "The SAHARA model for service composition across multiple providers," in *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*. London, UK: Springer-Verlag, 2002, pp. 1–14.

[11] D. Chakraborty, F. Perich, A. Joshi, T. W. Finin, and Y. Yesha, "A reactive service composition architecture for pervasive computing environments," in *PWC '02: Proceedings of the IFIP TC6/WG6.8 Working Conference on Personal Wireless Communications*. Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 2002, pp. 53–62.

[12] J. Robinson, I. Wakeman, and T. Owen, "Scooby: Middleware for service composition in pervasive computing," in *MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*. New York, NY, USA: ACM, 2004, pp. 161–166.

[13] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, 1999.

[14] S. Marti and H. Garcia-Molina, "Taxonomy of trust: Categorizing p2p reputation systems," *Computer Networks*, vol. 50, no. 4, pp. 472–484, March 2006. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2005.07.011

[15] D. Z. Kevin Hoffman and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," Purdue University, Tech. Rep. CSD TR 07-013, 2007.

[16] A. Josang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decis. Support Syst.*, vol. 43, no. 2, pp. 618–644, March 2007. [Online]. Available: http://dx.doi.org/10.1016/j.dss.2005.05.019

[17] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in P2P networks," in *WWW*, 2003, pp. 640–651.

[18] R. Housley, W. Ford, T. Polk, and D. Solo, "Internet x.509 public key infrastructure, certificate and CRL profile," IETF, RFC 2459, Jan. 1999.

[19] A. Samuel, A. Ghafoor, and E. Bertino, "Context-aware adaptation of access-control policies," *IEEE Internet Computing*, vol. 12, no. 1, pp. 51–54, 2008.

[20] D. Kulkarni and A. Tripathi, "Context-aware role-based access control in pervasive computing systems," in *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2008, pp. 113–122.

[21] D. Ferraiolo and R. Kuhn, "Role-based access control," in *In 15th NIST-NCSC National Computer Security Conference*, 1992, pp. 554–563.

[22] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[23] Ebay, "Ebay," July 2009, `http://www.ebay.com`.

[24] P. Michiardi and R. Molva, "Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," 2001, pp. 107–121.

[25] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *IEEE Symposium on Security and Privacy*, May 1996, pp. 164–173.

[26] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, "The KeyNote trust-management system version 2," IETF, RFC 2704, Sept. 1999.

[27] J. Buford, R. Kumar, and G. Perkins, "Composition trust bindings in pervasive computing service composition," in *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*. IEEE Computer Society, 2006.

[28] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward, "QoS-assured service composition in managed service overlay networks," in *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2003, p. 194.

[29] S. Jiang, Y. Xue, and D. Schmidt, "Minimum disruption service composition and recovery in mobile ad hoc networks," in *Computer Network Journal, Special Issue on Autonomic and Self-Organizing Systems*, 2008. [Online]. Available: http://www.truststc.org/pubs/442.html

[30] Glansdorff and Prigogine, *Thermodynamic Study of Structure Stability and Fluctuations*. Wiley, 1971.

[31] M. Bartoletti, P. Degano, and G. L. Ferrari, "Enforcing secure service composition," in *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 211–223.

[32] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, December 1959.

[33] G. Theodorakopoulos and J. S. Baras, "Trust evaluation in ad-hoc networks," in *Third ACM workshop on Wireless Security*, 2004.

[34] K. Krukow, M. Nielsen, and V. Sassone, "A framework for concrete reputation-systems with applications to history-based access control," in *ACM Conference on Computer and Communications Security*, V. Atluri, C. Meadows, and A. Juels, Eds. ACM, 2005, pp. 260–269.

[35] Kazaa, "Kazaa," July 2009, `http://www.kazaa.com`.

[36] K. Walsh and E. G. Sirer, "Fighting peer-to-peer spam and decoys with object reputation," in *SIGCOMM Workshop on Economics of Peer-to-Peer Systems*, 2005.

[37] S. Kalasapur, M. Kumar, and B. Shirazi, "Dynamic service composition in pervasive computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 907–917, 2007.

[38] B. Lagesse and M. Kumar, "Ubca: Utility-based clustering architecture for peer-to-peer systems." in *ICDCS Workshops, Mobile and Distributed Computing*. IEEE Computer Society, 2007.

[39] J. Nash, "Equilibrium points in n-person games," in *Proceedings of the National Academy of Sciences of the United States of America*, 1950.

[40] ——, "Non-cooperative games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, September 1951. [Online]. Available: http://dx.doi.org/10.2307/1969529

[41] X. Gu, K. Nahrstedt, and B. Yu, "Spidernet: An integrated peer-to-peer service composition framework," in *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 110–119.

[42] B. Lagesse, M. Kumar, and M. Wright, "AREX: An adaptive system for secure resource access in mobile P2P systems," in *Eighth International Conference on Peer-to-Peer Computing*. IEEE Computer Society, 2008, pp. 43–52.

[43] T. Haveliwala and S. Kamvar, "The second eigenvalue of the google matrix," Stanford University, Tech. Rep. 20, 2003. [Online]. Available: http://www.stanford.edu/~taherh/papers/secondeigenvalue.pdf

[44] Facebook, "Facebook," July 2009, `http://www.facebook.com`.

[45] Twitter, "Twitter," July 2009, `http://www.twitter.com`.

[46] Linkedin, "Linkedin," July 2009, `http://www.linkedin.com`.

[47] Blogger, "Blogger," July 2009, `http://www.blogger.com`.

[48] C. Dwyer, S. R. Hiltz, and K. Passerini, "Trust and privacy concern within social networking sites: A comparison of Facebook and MySpace," in *Proceedings of the Thirteenth Americas Conference on Information Systems*, August 2007.

[49] J. Caverlee, L. Liu, and S. Webb, "Socialtrust: tamper-resilient trust establishment in online communities." in *ACM IEEE Joint Conference on Digital Libraries*, R. L. Larsen, A. Paepcke, J. L. Borbinha, and M. Naaman, Eds. ACM, 2008, pp. 104–114.

[50] J. Fogel and E. Nehmad, "Internet social network communities: Risk taking, trust, and privacy concerns," *Computers in Human Behavior*, vol. 25, no. 1, pp. 153–160, January 2009.

[51] S. Greenhill and S. Venkatesh, "Virtual observers in a mobile surveillance system," in *ACM Multimedia 2006, 23-27 October, Santa Barbara, USA*, 2006.

[52] J. M. Paluska, H. Pham, U. Saif, G. Chau, C. Terman, and S. Ward, "Structured decomposition of adaptive applications," *Pervasive and Mobile Computing*, vol. 4, no. 6, pp. 791–806, 2008.

BIOGRAPHICAL STATEMENT

Brent graduated from Illinois Institute of Technology with a BS in Computer Engineering in 2004 and from University of Texas at Arlington with an MS and PhD in Computer Science in 2006 and 2009, respectively. During his graduate study, he worked with Lockheed Martin Missiles & Fire Control and Lawrence Livermore National Laboratory. He was the recipient of the National Physical Science Consortium fellowship for graduate study. Following graduation, he will begin as a Research Scientist at Oak Ridge National Laboratory in the Cyberspace Sciences and Information Intelligence Research group.