DTT: A distributed trust toolkit for pervasive systems

DTT: A Distributed Trust Toolkit for Pervasive Systems

Brent Lagesse,* Mohan Kumar,* Justin Mazzola Paluska,† and Matthew Wright*
* Department of Computer Science Engineering, University of Texas at Arlington, Arlington, TX
{Brent.Lagesse,mkumar,mwright}@uta.edu
† MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA
jmp@mit.edu

***Abstract*—Effective security mechanisms are essential to the widespread deployment of pervasive systems. Much of the research 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. We present the design and implementation of a novel framework called 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. In addition to the adaptability and extensibility provided by this design, we demonstrate through simulation that use of DTT improves utilization of resources and enhances performance of existing trust mechanisms in pervasive systems. We are currently developing an implementation of the DTT that can be easily deployed in pervasive environments.**

I. Introduction

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], [10], [15], [7], [3]. 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.

In this paper, we present the DTT, an infrastructure for managing trust mechanisms in pervasive computing environments. We have accomplished three main objectives with the DTT: (1) facilitating sharing of trust information in pervasive systems, (2) encouraging sharing of trust algorithms and code, and (3) making trust information systems easier to use and deploy in pervasive environments. The DTT enables the sharing of trust information through the formation of Trust Groups, provides a ready-made infrastructure for developers to create new and reuse existing trust mechanisms, and eases the integration of trust into pervasive systems through interoperability and incremental deployment.

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

A. Motivating Scenario

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

978-1-4244-3304-9/09/$25.00 ©2009 IEEE

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 Credence [13], 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.

B. Trust Groups

An important novel component of the DTT is the ability to create *Trust Groups*, in which a subset of the nodes with pre-established trust relationships share trust data. Trust Groups utilize the heterogeneity of pervasive computing environments to enhance the performance of existing trust mechanism. The members of a Trust Group, such as the various devices in a single user's personal area network (PAN) or a cluster of friends, trust each other's judgments about other nodes in the system. In our example, a group of friends may share reputation information about sources, so only one peer needs to engage in the full Credence protocol. DTT enables nodes within a trust group to share trust information with each other, thus substantially reducing the time and energy spent on evaluating the trustworthiness of others.

C. Paper Roadmap

We will first discuss the background and work that is related to DTT in Section II. In Section III, we outline the design of the DTT, including the implementation of Trust Groups. We use simulation to evaluate the benefits of our design in Section IV. We discuss the features and versatility of the DTT in Section V. Finally, we conclude in Section VI.

II. Background

Pervasive environments are replete with disparate hardware and software entities that are geared to meet the individual needs of users. Pervasive computing 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. Providing security and reliability of services is particularly challenging in these environments.

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.

A. Related Work

Distributed trust management has recently been a topic of much research interest in pervasive computing. However, the focus of these systems has largely been on the infrastructure and implementation of individual trust mechanisms. In this section, we describe a few of these systems and note how their approaches differ from that of DTT.

PolicyMaker [3] is a distributed trust-management engine designed to handle authorization via compliance checking. A compliance checking algorithm in the engine handles requests, credentials and policies in order to determine if the request is allowed. Keynote [2] is designed with similar goals as PolicyMaker and maintains many similarities; however, it has two additional goals: standardization and ease of integration with applications. We share these goals with Keynote, but take a much different approach. Whereas Keynote places more responsibility on the compliance checker and standardizes the policy assertion language, DTT accomplishes standardization and ease of integration through an abstracting trust mechanisms into modular Trust Blocks which can be easily integrated into both new and existing systems through the DTT Daemon.

Vigil [5] is a distributed trust-management architecture that utilizes Public Key Infrastructure (PKI) and Role-based access control (RBAC), but extend RBAC by using ontologies in XML to represent properties and constraints which allows for dynamic and policy-based delegation and revocation of rights.

The most similar work to our research from an architectural standpoint is that of the Gaia Authentication Service [10]. Gaia Pluggable Authentication Modules (GPAM) are used to implement a security system that decouples authentication from both the individual hardware and the security framework itself. GPAMs are divided into two types, Authentication Mechanism Modules (AMM) and Authentication Device Modules (ADM). AMMs are used to define the authentication mechanism in a device-independent manner and ADMs allow new devices to be incorporated into the system for use by AMMs. Our approach differs from that of Gaia in that we do not focus solely on authentication, but on distributed trust in general. Our approach is designed not only to allow for pluggable modules (our Trust Blocks), but also to ease implementation through extension and reuse of Trust Blocks.

1) Limitations: The systems described above primarily focus on the distribution and checking of certificates against policies to establish trust. We find that 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 pervasive computing such as reputation-based [15], [7], [12], [8], [14] or recommendation-based [11], 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. DTT addresses both of these issues by providing a framework for highly modular and reusable components, allowing different trust mechanisms to be applied to many types of environments and applications. DTT goes beyond just extending the scope of a trust management toolkit, it also utilizes the resources in pervasive systems to improve the performance of existing trust mechanisms.

III. DTT System Design

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 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 IV-B for an example of transition Credence from an internet-based implementation to one more suitable for a MANET by only changing the Protocol component).

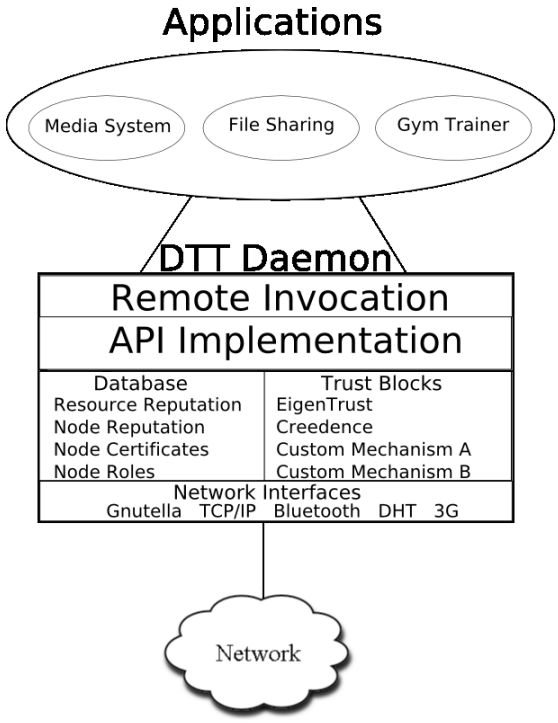

Fig. 1. DTT Architecture

A. Design Goals

In Section I, we introduced three goals for 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 [1], 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 discussed in Section I-A. 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

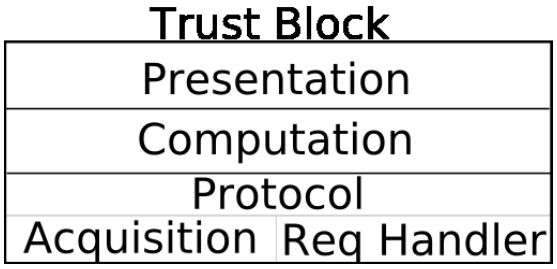

Fig. 2. The components of a Trust Block

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.

B. Modular Trust Blocks

Each Trust Block consists of three layered components: the Protocol, the Computation, and the Presentation. Figure 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 intiate 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 III-B2.

1) Trust Block 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 [4] 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 III-C.

2) Trust Block 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 I-A, 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) Trust Block 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 conists 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

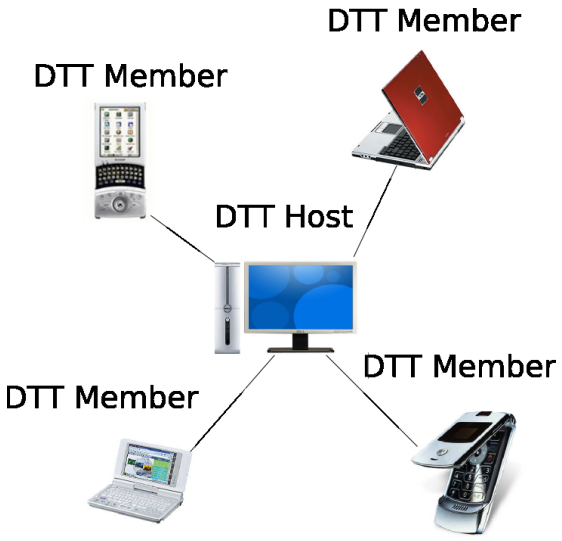

Fig. 3. Devices in a personal area network using a Trust Group.

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.

C. 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 Host may be running several different applications that utilize different Trust Blocks, 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. In Section IV, 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. 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) [6] in dynamic environments or the Utility Based Clustering Architecture (UBCA) [9] for decentralized clustering.

D. Platform 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.

E. 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.

IV. Evaluation

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 [13]. 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.

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 [14] and as a simulated system [13], 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.

A. Simulation Setup

The simulation portrays an adaptive media system similar to that described in Section I-A. 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 [13], we focus primarily on the reduced cost to achieve these results that is accomplished by using Trust Groups. For convenience, Table I 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.

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%

TABLE I
DEFAULT SIMULATION PARAMETERS

B. Simulation Results

Our first simulations were designed to demonstrate the effect of increasing the number of peers in the system. Figure 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 peer eventually

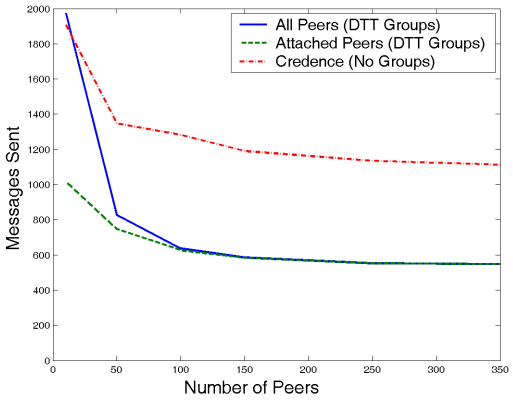

Fig. 4. The Effect of the Number of Peers on Trust Messages per Peer

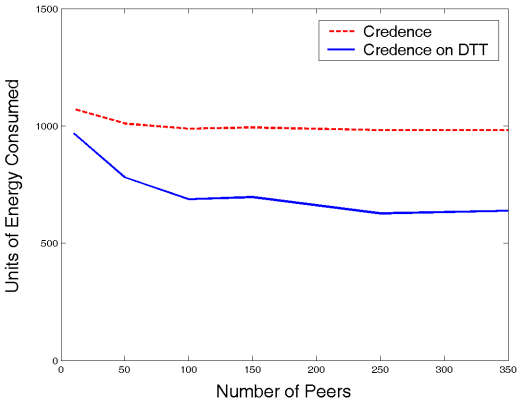

Fig. 6. Energy Consumption per Peer

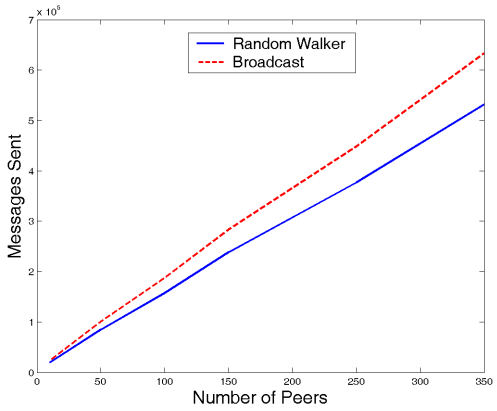

Fig. 5. Comparison of Acquisition Components

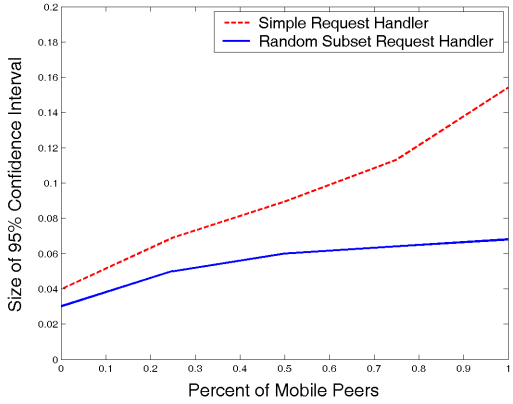

Fig. 7. Effect of Exchanging Protocol Components on Confidence Intervals

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 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.

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 6. The dominating portion of the energy consumption in this simulation setup comes from the messages sent. In scenarios where 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 parameters in Table I). To obtain the confidence results in Figure 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.

V. Discussion

A. Trust Programming

The primary benefit of the Trust Block abstraction is the ability to tune, extend, and adapt trust mechanisms simply by changing Trust Block components. Suppose the users in our streaming media example start relying more on their smartphones, which have less energy available than their laptops. Alice investigates and determines that Credence network broadcasts are using most of the power. She first tunes the system by modifying her CredenceReputationTrustBlock so that it uses a RandomWalkerAcquisition component instead of a BroadcastAcquisition component.

Alternatively, Alice may work on a lower level. She decides that she wants to change the overlay discovery network that she is using from a Gnutella-based overlay to a supernode-based overlay, the decision has no effect on the Trust Block she is using, since each network implements a generic network interface that the Trust Block uses. The new acquisition component allows Alice to reduce the amount of energy spent searching for trust information and provide more time for her mobile users to access media. In this scenario, the DTT easily allows Alice to customize the Trust Block she is using to fit her system's needs without having to rebuild her system beyond the modified components. Additionally, if she does decide to make changes to her system's architecture, the Trust Blocks that she uses do not have to make any changes to continue functioning on the new system.

VI. Conclusion

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.

In our implementation, we have exploited the modular feature of Trust Blocks by mixing and matching components from existing Trust Blocks. Moreover, in simulation, we found that using the DTT with Trust Groups reduced the number of trust messages sent by an individual host by 58%, leading to power and computation savings on mobile hosts.

VII. Acknowledgments

The work presented in this paper was partially supported under US National Science Foundation Grants CSR 0834493 and ECCS 0824120.

References

[1] http://www.kazaa.com/us/help/faq/supernodes.htm. 2008.
[2] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system version 2. RFC 2704, IETF, September 1999.
[3] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, May 1996.
[4] Russell Housley, Warwick Ford, Tim Polk, and David Solo. Internet x.509 public key infrastructure, certificate and CRL profile. RFC 2459, IETF, January 1999.
[5] Lalana Kagal, Jeffrey Undercoffer, Filip Perich, Anupam Joshi, and Tim Finin. A security architecture based on trust management for pervasive computing systems. In *Grace Hopper Celebration of Women in Computing*, October 2002.
[6] Swaroop Kalasapur, Mohan Kumar, and Behrooz Shirazi. Dynamic service composition in pervasive computing. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):907–917, 2007.
[7] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW*, pages 640–651, 2003.
[8] Karl Krukow, Mogens Nielsen, and Vladimiro Sassone. A framework for concrete reputation-systems with applications to history-based access control. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *ACM Conference on Computer and Communications Security*, pages 260–269. ACM, 2005.
[9] Brent Lagesse and Mohan Kumar. Ubca: Utility-based clustering architecture for peer-to-peer systems. In *ICDCS Workshops, Mobile and Distributed Computing*. IEEE Computer Society, 2007.
[10] Geetanjali Sampemane, Prasad Naldurg, and Roy H. Campbell. Access control for active spaces. In *ACSAC*, pages 343–352. IEEE Computer Society, 2002.
[11] Giovanna Di Marzo Serugendo. Trust as an interaction mechanism for self-organising systems. In *ICCS*, 2004.
[12] George Theodorakopoulos and John S. Baras. Trust evaluation in ad-hoc networks. In *Third ACM workshop on Wireless Security*, 2004.
[13] Kevin Walsh and Emin Gün Sirer. Fighting peer-to-peer spam and decoys with object reputation. In *SIGCOMM Workshop on Economics of Peer-to-Peer Systems*, 2005.
[14] Kevin Walsh and Emin Gün Sirer. Experience with an object reputation system for peer-to-peer filesharing (awarded best paper). In *NSDI*. USENIX, 2006.
[15] Li Xiong and Ling Liu. Building trust in decentralized peer-to-peer electronic communitties. In *International Conference on Electronic Commerce Research*, 2002.