Policy Based Framework for Trust Management and Evolution of Peer to Peer Groups

Third Annual Progress Seminar Report

Submitted by

Madhumita Chatterjee
Roll No: 04429802

under the supervision of

Prof. G. Siva Kumar and Prof. Menezes Bernard

Computer Science and Engineering Department
Indian Institute of Technology, Mumbai

January 16, 2008
Acknowledgments

I take this opportunity to thank Professor G. Siva Kumar and Professor M. Bernard for their invaluable guidance, encouragement and support. Acknowledgements are also due to my RPC committee members, Professor Anirudha Sahoo and Professor Om Damani for their valuable suggestions and comments during my earlier presentations.

Madhumita Chatterjee
Abstract

Peer to peer collaborative groups are becoming increasingly popular for collaborative applications like video/audio conferencing, IP telephony, file sharing, collaborative work spaces, and multi-user games. The decentralized nature of these groups gives rise to the need of a secure group layer which integrates authentication, admission control, authorization, access control and key management. With the evolution of dynamic, self organizing groups like wikipedia, f/oss and other business community groups, it motivates the need for dynamic multi-level access control. In an open decentralized system attacks from malicious peers pose a significant threat to the integrity of the system. Reputation based trust management systems are able to provide some countermeasures.

We integrate role based, policy based and trust based access control for decentralized groups and propose a dynamic multi-level policy mechanism with a context based trust metric for our previously proposed Integrated Framework for Authentication and access control in peer to peer groups. We have successfully modelled our protocols in a peer to peer simulator Peersim. The performance of the group can be tested for different policies in the presence of malicious peers.
# Contents

1 Introduction ............................................. 1

2 Proposed Integrated Framework .................. 3
   2.1 Model Overview .................................. 3
   2.2 Proposed Architecture ......................... 4

3 Policy based framework for trust management .... 7
   3.1 Group composition and Policies ............... 7
       3.1.1 Dynamic Access Control Policies ............ 8
   3.2 Policy Language Specification ................. 9
       3.2.1 X-GTRBAC ........................................ 9
   3.3 Trust Management ................................ 12
       3.3.1 Decentralized Trust Management .......... 12
       3.3.2 Game Theoretic Analysis of Trust Management .. 13
   3.4 Threats for Peer to Peer Groups ............... 14
       3.4.1 Adversaries ..................................... 14
       3.4.2 Trust Model in presence of malicious peers .. 15

4 Trust evaluation for multiple context dimensions in Incentive based system .... 16
   4.1 Context Sensitive Trust Model ................. 16
   4.2 Modified Framework ............................. 18
   4.3 Unsolved Issues ................................. 18

5 Simulation and Analysis ............................. 21
   5.1 Introduction ..................................... 21
   5.2 Peersim Simulator ................................ 21
       5.2.1 PeerSim Architecture ....................... 22
       5.2.2 PeerSim Simulation Life Cycle ............... 23
   5.3 Code Overview ................................... 24
   5.4 Simulation: Graphs and Analysis ............. 27

6 Conclusion .............................................. 30
   6.1 Future Scope ...................................... 30
List of Figures

2.1 Integrated Framework ........................................ 5
4.1 Modified Framework ........................................ 19
5.1 General Functioning of PeerSim ............................ 22
5.2 Configuration File ........................................ 23
5.3 Graph1 .................................................. 27
5.4 Graph2 .................................................. 28
5.5 Graph3 .................................................. 28
5.6 Graph4 .................................................. 29
Chapter 1

Introduction

Web based collaborative groups are becoming increasingly popular as peers with common interests form a network among themselves and automatically tend to create interest groups among each other, called communities. The group is governed by a set of rules that describe the conditions required to be part of the group. Security in such dynamic collaborative groups is governed by membership control, authentication, access control and key management. Membership control and authentication is required to allow only authorized members to join the group. In the dynamic scenario where peers are constantly leaving and joining this becomes a difficult task.

The evolution of dynamic, self organizing groups like wikipedia, f/oss and other business community groups motivates new security requirements for peer to peer communication. Consider an example of a developer’s community for some critical security related open-source software. The communication forum could be either chat or e-mail. New users would be allowed into this forum to put in their suggestions, only if they are introduced by an existing member. Depending on the sensitivity level of the code being developed, there could be hierarchial levels for the members of this forum. New members and existing members could be periodically rated or evaluated by existing members based upon the quality of their contribution. A member receiving a high rating value could be elevated to a higher level in the forum. A peer or a member in any level need not always be evaluating other peers, i.e a member can have different roles in a group. For critical software like security applications, authentication of developing members would be important as well as deciding which member is allowed to play which role and participate in which level.

In the dynamic P2P communities where peers are unknown to each other and uncertain about each other’s reputation it is necessary to develop strategies for establishing trust among peers. Peers in self-organising groups with multiple levels, should have the right to collaboratively modify the access control policies governing their levels and the levels below them. Current solutions do not allow dynamic change of access levels or policies, with incremental building of trust during the communication. Nor do they allow dynamic authentication of participants who were previously part of the group and later wish to rejoin. In view of the varied security requirements for diverse peer to peer applications, we had proposed an integrated framework, which has a flexible, dynamic and totally decentralized access control mechanism, integrated with authentication and dynamic rekeying.

We now build upon our previous framework and introduce the concept of unequal roles for peers in a group. There can be different levels within a group and peers at each level can have different functionalities. Peers in one level can dynamically elevate their levels in a group. Group composition determines join priorities. In applications such as workflow
management, access to information may be allowed on a need to acquire basis. For e.g the sensitive resources of an enterprise may only be exposed during relevant production stage. Access rights to individuals may therefore need to be dynamically adjusted depending on the context. A single task may require the co-operation of multiple policy domains. We thus model a new framework for dynamic multi-level policies consisting of global meta policies and context based policies. The context based policies take inputs from a dynamic trust engine. We propose different metrics for trust evaluation in multiple context dimensions.

In order to analyze the pros and cons of different trust settings and different policies and study their effect on the evolution of P2P groups we chose a java based peer to peer simulator peersim. We have successfully modelled our protocols for join, leave and updation in this simulator and we analyze the effect of different policies in the presence of malicious peers in our framework.
Chapter 2

Proposed Integrated Framework

We have modelled an integrated framework for decentralized groups, which allows us to implement secure communication between different peers using admission and trust based multi-level access control along with key management. The architecture is suitable for applications that involve sharing of high value resources, e-commerce applications, as well as general collaborations where security is not a major issue. Protocols have been proposed for new peers to join, level updation of existing peers and dynamic leaving of peers. Idea of per-session authentication is also proposed which depends on the incremental trust factor and eliminates the need of the voting process for a old member of the group who wishes to re-join. Our dynamic access control policy framework is flexible and permits peers to collaboratively frame, and modify policies at various levels. The policy also permits peers to update their access rights within a group. Join priorities can also be decided by the group policy and depend upon current group composition.

The different cases supported by our framework are:

1. A peer can wish to join in an existing group but it should have a X.509 certificate signed by trusted CA or can have self-signed certificate with proof of possession of private key.
2. An existing peer can ask to update his level or role.
3. A peer who was previously part of the group can wish to rejoin in a new session.
4. Dynamic leaving of peers from the group.

2.1 Model Overview

We consider a peer based model where every peer $P_i$ has a unique user identity $UUID_i$ associated with it. A group is a named set of peers or a subset of a community that is governed under a set of rules that describe the conditions required to be part of the group, and is formed based on a particular interest criterion. The dynamic membership of the group implies that no peer will be highly available. The challenges are rapid admission of new members with appropriate access rights, dynamic elevation of access levels of existing members and dynamic revocation of access rights of leaving or defaulting members.

Basic Assumptions The following are the assumptions made in our model.
• No centralized control: Traditional access control models such as ACL or RBAC \cite{3} rely on central servers for authorization. In our model a peer has a high level of autonomy and can frame and manage his own policies.

• Peers in the P2P system are loosely coupled and interacting partners are unknown.

• Peers in a group can belong to different levels within the group and can perform several roles.

• A role of a peer is independent of the level which the peer is in.

• Peers are individually capable of performing tasks of authentication, voting, access control, key management etc.

• Peers can compute their own public-private key pair and provide self signed certificates, binding their identity with their public key.

Each group has a group policy which describes the roles permitted in the group and the rules associated with each role. The functions that a peer is capable of performing in a group are storing and verifying certificates, authentication, voting, updation of levels, key management etc. The extent to which a peer performs these tasks is decided by the role the peer wants to play in the group. We identify the following roles:

1. **Member peer** A peer in this role participates in the activities of the group like gaming, conferencing, file transfer etc, but does not contribute to the admission of new peers, nor updation of levels of existing peers. Thus this peer need not store certificates of every other peer and need not invoke voting. either.

2. **Admission Control Peer** A peer with this role would participate in the voting algorithm, and would therefore store public keys of every other peer. However it need not store the updated ratings.

3. **Maximal role peer** A peer in this role would have all the functional components and would participate in updation levels of existing peers also.

The access policy would decide the role a peer assumes, based on the request of the peer and his credentials as well as rating calculated by the other peers.

### 2.2 Proposed Architecture

We present here a flexible architecture framework for secure group communication in decentralized peer groups, based on our proposed access control framework. The architecture is suitable for applications that involve sharing of high value resources, e-commerce applications, as well as general collaborations where security is not a major issue. The access policy framework is flexible and allows peers to change their levels as well as change the rules for membership within an existing level. Peers are rated based on their performance in a group along with contributory recommendations by other peers. A peer can then invoke the rekeying algorithm to elevate his level. Policy rules for elevating members from one level to another as well as for introducing new levels are dynamic and peers are able to frame and manage such policies.
A peer in a maximal role would be expected to have the following functional components:

**Attribute Manager**: Responsible for storing the attributes and certificates of the peers. The Attribute Manager is also responsible for updating the certificates of the peers with the new rating.

**Authentication Manager**: Verifies the validity of the identity and invokes the voting algorithm.

**Trust Engine**: Responsible for calculating the trust values of each peer, based on responses from other peers. It thus has a calculation engine and a feedback engine associated with it.

**Access Policy Manager**: Frames rules or policies based on attributes, behavior of peers and inputs from trust engine. It also checks the rules to allow members to join. It is called upon either by the Authentication Manager in case of a new member join, or the Update Manager in case of an existing member level updation.

**Update Manager**: Responsible for checking the rating levels of peers after they have collected feedback from other peers and gives the input to the Access policy manager. It is also responsible for granting or revoking access rights to peers based on the dynamic policy.

**Key Management System**: Invoked whenever a new peer joins or an existing peer’s access level is changed depending on the dynamic policies, and is responsible for computing the group key. These components interact as shown in the figure 2.1.

Our initial membership admission protocol is similar to the protocol proposed by Kim et al. [1][2] and allows all peers to collaboratively decide whether a new user should be allowed entry into the group and if so decide the hierarchy level or role in which he can join the group. First time authentication is based on user identity, either an X.509 credential, or a self generated pseudo certificate and other attributes presented during initial admission request phase. This is handled by the Authentication Manager and Attribute Manager. If permitted to join, a peer is given a signed membership certificate which serves as an authentication token. For all practical purposes a new peer will be allowed to join a group at the lowest level initially. The membership certificate contains a field which has the trust level or rating of the member.

![Figure 2.1: Integrated Framework](image-url)
Subsequent per session authentication could be based on an authentication token received in prior session and recognized by all peers. This would require inputs from the Access Policy Manager and the Trust engine.

Once a member of the group, the access control policy could be dynamically changed and the user could be switched from a low hierarchy level to a higher hierarchy level depending on his performance in the group as well as the rating given to him by other peers in the group. A rating of a peer can either be periodically updated by all existing peers in the group or a peer can request for rating certificates from other peers in the group with whom he has had transactions.

Rekeying algorithm would be invoked by the peer whose rating increases, so that he could switch to a higher hierarchy. If a member who was previously part of the group and has left, wishes to rejoin the group, then based on the rating token and credentials that he possesses, he could easily rejoin without having to be subjected to the voting process, or if he has a high rating i.e. a high trust level, then he may be permitted to join as a trusted member even though the votes of remaining members are less than the threshold i.e. 50 percent i.e. the admission policy changes if the member is an old member.

The credentials provided by the user could have various levels of trust, that could be incremental.

The complete architecture and protocols proposed have been published in [20].
Chapter 3

Policy based framework for trust management

The model proposed earlier did not take into account malicious behavior of peers. The trust model was a simple model based on Eigen\cite{9} trust model with some variations for e-commerce transactions. We now focus on policy language issues, trust management issues and the behavior of our framework in presence of malicious peers.

3.1 Group composition and Policies

Our model focuses on the genetic evolution of groups based on group composition. Multiple policies can apply to a group simultaneously, or the system can switch between policies over time, influenced by the system’s state or environment. Some policies can have higher priorities than others. Policies can apply to different levels of the system and can be global or apply to only parts of the system. Currently we define two levels of policies viz:

Global meta-level policy

This would be the policy initiated by the creator of the group. The peers can contribute to policy rules by dynamically adding new rules. The meta policy would contain rules to specify the threshold of peers required and conditions and constraints to be satisfied. As an example the meta policy could have rules like:

- For a new policy to be framed at least 75% of existing peers with trust value > some threshold x must collaborate.
- For a new level to be introduced at least y no. of peers must be present in the current highest level and all 100% should collaborate.
- To update level a peer must have an accumulated trust value > x and should be an authenticated member at the lower level.

Domain specific policy

This set of policies would inherit the properties of the global policy and would contain application specific policies. So policy rules for an e-commerce scenario, online gaming sce-
nario or online software development would be specified here depending on the application. The performance of the group would depend upon policy rules like:

- Number of member level peers to be permitted to complete a specific task.
- Number of maximal role peers required.
- The group policy can also dynamically prioritize requests for join.
- Low level requests for join could be postponed in a group which already has a large number of members performing the lower roles.
- It is also possible for peers at the highest level to introduce a new level into the group if the situation so demands.

3.1.1 Dynamic Access Control Policies

Our hybrid access control model is based on an integration of collaborative roles and trust. A role represents a set of privileges and rights. It involves meta-level policies in regard to admission of users to the role. Roles are globally defined. Users can exist in different hierarchial levels in the group and can be assigned roles in these levels. Users assigned to a role are dynamically assigned privileges and tasks based on the context. The privileges assigned to a user in a role may change with time due to the behavior of peers in the group. Thus permissions need to be assigned to roles dynamically based on the context. Integration of the roles, tasks, privileges and security policies needs a proper framework for policy specification.

Requirements for secure collaboration

The security and access control requirements that should be supported in our hybrid model are outlined below.

Role Admission constraints

Conditions that need to be satisfied when a user requests to join a group in a particular role.

- List of users who can be allowed to join.
- Previous qualification requiring that the requesting user is currently admitted in some lower role.
- Role membership cardinality
- Events that must happen before a user is admitted in the role.

For e.g a maximal role peer in a software development workflow context could have the following admission constrains:

#members(Maximal_role)> 2 & #members(Maximal_role) < 5 &
member(thisUser,Moderator) &!member(thisUser,developer)
& #members(admission_control_peer)>0
Co-ordination requirements Co-ordination between peers in different roles within an activity is referred to as *Intra-role coordination*. For example the inter-role co-ordination requirement in a software development project can be that a verifier can test a software only after the developer has submitted the final product. When multiple users are there in a role it might be necessary to have some co-ordination between such as jointly doing certain tasks, known as *Intra-role coordination*.

3.2 Policy Language Specification

To specify the dynamic policy language for our framework we propose to use the Trust-enhanced X-GTRBAC model [22]. This section begins with an introduction to the GTRBAC model. It then describes the mechanism to configure X-GTRBAC to provide context-aware trust-based access control.

GTRBAC Model

The GTRBAC [23] model incorporates a set of language constructs for the specification of various temporal constraints on roles, including constraints on their activations as well as on their enabling times, user-to-role assignments, and permission-to role assignments. In particular, GTRBAC makes a clear distinction between role enabling and role activation. An enabled role indicates that a user can activate it, whereas an activated role indicates that at least one subject has activated a role in a session. The notion of separate activation conditions is particularly helpful in large enterprises, with several hundred users belonging to the same role, to selectively manage role activations at the individual user level.

3.2.1 X-GTRBAC

The X-GTRBAC framework is based on Generalized Temporal Role Based Access Control (GTRBAC) model. X-GTRBAC augments GTRBAC with XML to allow for supporting the policy enforcement in an heterogeneous, distributed environment. GTRBAC extends the widely accepted Role Based Access Control (RBAC) model proposed in the NIST RBAC standard. RBAC uses the concept of roles to embody a collection of permissions within an organizational setup. Permissions are associated with roles through a permission-to-role assignment, and the users are granted access to resources through a user-to-role assignment [3]. GTRBAC uses the core RBAC model as per the NIST standard, together with support for separation of duty and role hierarchies. It provides a generalized mechanism to express a diverse set of fine-grained temporal constraints on user-to-role and permission-to-role assignments in order to meet the dynamic access control requirements of an enterprise. The RBAC model consists of the following components:

- Sets Users, Roles, Permissions and Sessions representing the set of users, roles, permissions, and sessions, respectively;
- UA: Users X Roles, the user assignment relation, that assigns users to roles;
- assigned users(r: Roles) —> 2^Users, the mapping of role r onto a set of users. Formally: assigned users(r) = u ∈ Users — (u, r) ∈ UA
- PA: Roles X Permissions, the permission assignment relation, that assigns permissions to roles;
• assigned permissions(r: Roles) → \(2^{\text{Permissions}}\), the mapping of role \(r\) onto a set of permissions. Formally: assigned permissions(r) = \(p \in \text{Permissions} \parallel (p, r) \in \text{PA}\)

• user: Sessions → Users, which maps each session to a single user;

• role: Sessions → \(2^{\text{Roles}}\) that maps each session to a set of roles; . \(\text{RH} \subseteq \text{Roles} \times \text{Roles}\), a partially ordered role hierarchy (written \(\geq\)).

Features of X-GRTABC

X-GRTBAC allows the specification of all the elements of GRTBAC model through a context free grammar called X-Grammar which supports the tagging notation of XML. Some of the salient features of the model are given below:

• XML User Sheet(XUS) Declares the users and their authorization credentials.

• XML Role Sheet Declares the roles, their attributes, role hierarchy.

• XML Permission sheet (XPS) Declares the available permissions.

• XML User-to-Role-Assignment Sheet(XURAS) Declares rules for assignment of users to roles

• XML Permission-to-Role Assignment Sheet(XPRAS) Defines rules for assignment of permission to roles

Context aware access control

This section defines the set of specifications needed to configure X-GTRBAC for context aware access control. The set of specifications are based on a tuple language that can be readily mapped onto the existing XML-based framework. The formal model relies on the components defined below:

• Parameter Name Set: A set PN to denote the possible names of context parameters

• Parameter Type Set: A set PT to denote the possible types of context parameters

• Context Parameter: A context parameter is represented by a data structure \(p\), having the following fields: name \(\in\) PN, type \(\in\) PT, and a function getValue().

• Roles Set: \(\text{RR} = \{r_{r1}, \ldots, r_{rk}\}\), where \(r_{ri}\), \(i = 1, \ldots, k\), is a regular role in GTRBAC

• Operations Set: \(\text{RO} = \{r_{o1}, \ldots, r_{ok}\}\), where \(r_{ok}\), \(i = 1, \ldots, k\) is a regular operation in GTRBAC

• Service: A service is an abstraction of the operations provided by the system on its resources. Formally, a service is a subset of the operation set \(\text{RO}\), and is designated by the service name \(\text{srv}\).

Context set. A context set \(C\) consists of \(n\) context parameters \(\{p_1, \ldots, p_n\}\), \(n \geq 0\), s.t. for any \(p_i, p_j\), with \(i \neq j\) and \(1 \leq i, j \leq n\), we have that \(p_i.\text{name} \neq p_j.\text{name}\) (i.e. the parameter names must be distinct) For example, the set PN may be defined as: \(\text{PN} = \{\text{time of day, location, duration, system load}\}\), with the corresponding set PT defined as: \(\text{PT} = \{\text{Time, String, Long, Integer}\}\).
Modelling our framework

We define the following components in our language framework:

- **Role Sets (RS):** This comprises of the different roles permitted.

- **Group Hierarchy (GH):** These are the different levels possible in the group and would depend upon the attributes of the peers and the composition of the group.

- **Service Set (SRVC):** This is the possible set of operations permitted within the group.

**Service access request.** A JOIN_REQ or an UPDATE_REQ is defined as a quadruple <\(role, srv, contextset\)> where \(role \in RS\), \(srv \in SRVC\) and context set comprises of context parameters captured dynamically at the time of access request. Different rules would be specified for different levels where \(level \in GH\). Based on the join or update request, the system determines the applicable access policy for the requested service. This policy will be based on a set of constraints on the role and service name, and evaluated in conjunction with the available contextual information to enforce fine grained access control. An access constraint in our framework is a collection of access conditions. The access policy for a role service pair is a set of clauses consisting of multiple access conditions.

Some sample rules for a multi-chat application are:

**Rule1:**
If request = JOIN_REQ then
if role = member \& level = 1 \& srv = mesg_receive then
if rating > 1 \& voting minbound > 50 then permission = ”grant”

**Rule2:**
If request = JOIN_REQ then
if role = member \& level = 2 \& srv = mesgreceive, msg_send then
if rating < 4 then permission = ”deny”

**Rule3:**
If request = UPDATE_REQ then
if role = admission_peer \& level = 2 \& srv = mesg_receive then
if rating > 6 then permission = ”grant”

**Rule4:**
If request = UPDATE_REQ then
if role = maximal_peer \& level = 3 \& srv = mesg_receive then
if rating > 8 then permission = ”grant”

We use attribute-based credential specification. There is no compulsory reliance on X.509 identity-based certificates to encode user authentication information. The authentication credential comprises of self signed user attributes which are used for role assignment. The credential specification in our framework facilitates a combination of rule-based role assignment and role-based authorization. Roles in our model are of two types: **Attribute-Roles** and **Group Roles**. Attribute Roles correspond to the role the peer is assigned in the group based on his attributes and rating, thus in case of an University it could be the role of a Professor or a student or a Director. Whereas group-role corresponds to the functionality of the peer in the group, which the peer could choose at the time of joining and could modify later while in the group.
3.3 Trust Management

Trust is the firm belief in the competence of an entity to act independently [10] [18], securely and reliably within a specified context. It is conditionally transitive and can be multi-dimensional and depends on the context. Reputation is an expectation about an individual’s behavior based on information about or observations of its past behavior [17]. Reputation-based trust management allows each peer to manage its own trust, hence there is no global centralized map of trust relationships, and it isn’t convenient to manage. Based upon the approach adopted to establish and evaluate trust relationships between peers, trust management has been classified into three categories:

- Credential and policy-based trust management where peers use credential verification to establish a trust relationship with other peers. e.g Policy maker, Keynote.

- Reputation-based trust management where peers establish trust relationship with other peers and assign trust values to those relationships. Trust value assigned to a trust relationship is typically a function of the combination of the peer’s global reputation and the evaluating peer’s perception of that peer. e.g Eigen[9] trust model

- Social network-based trust management where social relationships between peers are taken into account when computing trust relationships. e.g Regret, NodeRanking.

3.3.1 Decentralized Trust Management

In a Distributed trust model [5], trust relationship is always exactly between two entities and is conditionally transitive. It is either a Direct trust relationship where one peer trusts another based on direct interactions or a recommender trust relationship where a peer trusts other peers to give recommendations about another peer’s trustworthiness. The main challenges in building a reputation based trust system in decentralized groups are

- How to cope with potential malicious peers
- How to cope with indirect trust given input of direct trust valuations
- How to continuously revise trustworthiness of a peer at runtime
- How can trust evaluations be done when peers have unequal roles in the group

Different models have been proposed.

Complaint based model This model relies on negative feedback or complaints to convey reputation information. Peers do not store information about successful interactions, only negative experiences are stored. Complaints are then forwarded to other peers. When a peer wants to evaluate the trustworthiness of a target peer, it first searches its own history to locate complaints registered by itself. It then queries other peers. This scheme is adopted by P-Grid data structure.

Voting based Reputation Model XREP [24] is a distributed protocol that allows these reputation values to be maintained and shared among the servents. It consists of the phases of resource searching, resource selection and vote polling, vote evaluation, best servant check, and resource downloading.
eBay Reputation Management System: This does not include a trust computation mechanism [10]. It presents the reputation information about a user and leaves trust determination and decision making to its users.

Credence uses a voter correlation scheme that weighs peers opinions [14]. NICE platforms includes trust algorithms that enable good peers to form robust co-operative groups with other good peers[13]. Marti and Garcia propose a limited reputation sharing scheme to reduce the no. of failed interactions between peers.

PeerTrust is a reputation based trust model for e-commerce communities [12]. It focuses on incorporating the credibility of recommenders in the evaluation of trust. It also includes the number of transactions and transaction community context to make trust evaluation more accurate.

3.3.2 Game Theoretic Analysis of Trust Management

Game theory has been successfully used to model and analyze different aspects of P2P networks, especially the problems related to trust. Game theoretic analysis in P2P mainly deals with pure strategic games where a player is fully certain of the action he/she should take (with probability 1) and mixed strategies are not considered sufficiently. However a natural question that arises is how to model uncertainty and belief among peers. Mixed strategy games could be used to cope with uncertainty of peers’ decision making process.

Non-Incentive based system

In a complaint based distributed system peers report about the past performance of other peers in a distributed manner. However reporting could be costly in terms of time consumed or bandwidth used and hence a peer may not want to report, but would prefer that some other peer does it. A peer would gain some level of satisfaction if a malicious peer is reported. Assume that the peer attaches a value s to this. Let the cost associated with reporting be c. Assume that s > c > 0.

Model:
Players : n reporting peers
Actions : Peer can decide whether to report or not to report.
Payoff : Given the assumptions mentioned above, s is 0 if a malicious peer is not reported, (s-c) if that peer himself is the reporting peer and s if at least one other peer (not himself) reports.

Theoretical Analysis.

- P = probability that each peer reports about malicious peer.
- Probability that no one out of the remaining (n-1) peers report = \((1 - p)^n\).
- Probability that at least one peer out of remaining peers report = 1- \((1 - p)^{n-1}\)

Nash Equilibrium

In game theory Nash Equilibrium is a solution concept of a game involving two or more players in which no player has anything to gain by changing his or her strategy, i.e A and B are in Nash Equilibrium if A is making the best decision he can, taking into account B’s decision and vice versa. Many players are in Nash Equilibrium if each one is making the best decision they can taking into account the decision of others.

Considering Nash Equilibrium state here, the expected payoff of reporting each player is equal to the expected payoff for not reporting.
Thus \( s - c = 0 + s (1 - (1 - p)^{n-1}) \)

\[ p = 1 - (c/s)^{1/n-1} \text{ for } 0 < c < s \text{ i.e. } 0 < c/s < 1 \text{ which is a probability.} \]

Thus the probability that each peer reports about malicious peers decreases as the reporting peers increase. In our model we can conclude that in a non-incentive based system, more the number of maximal role peers, less likely that the malicious peer is reported. Thus more the number of decision makers, less reliable is the outcome.

### 3.4 Threats for Peer to Peer Groups

The two primary types of adversaries in peer-to-peer networks are selfish peers and malicious peers. They are distinguished primarily by their goals in the system. Selfish peers wish to use system services while contributing minimal or no resources themselves. A well-known example of selfish peers are freeriders in file-sharing networks, such as Kazaa and Gnutella. To minimize their cost in bandwidth and CPU utilization freeriders refuse to share files in the network. The goal of malicious peers, on the other hand, is to cause harm to either specific targeted members of the network or the system as a whole. To accomplish this goal, they are willing to spend any amount of resources Examples include distributing corrupted audio files on music-sharing networks to discourage piracy or disseminating virus-infected files for notoriety. Reputation system designers usually target a certain type of adversary. For instance, incentive schemes that encourage cooperation may work well against selfish peers but be ineffective against malicious peers. The number or fraction of peers that are adversaries also impacts design.

#### 3.4.1 Adversaries

Anonymous nature of peer-to-peer (P2P) systems exposes them to malicious activity. Establishing trust among peers can mitigate attacks from malicious peers. Peers trust each other to perform operations such as routing file search queries and downloading/uploading files. However, a malicious peer can use the trust of others to gain advantage and can harm the operation of a system. Detecting malicious behavior is difficult without collaboration. However, feedbacks from peers might be deceptive, and thus, identifying a malicious peer with high confidence becomes a challenge. Some of the adversaries for P2P networks are:

**Traitors:** Some malicious peers may behave properly for a period of time in order to build up a strongly positive reputation, then begin defecting. This technique is effective when increased reputation gives a peer additional privileges, thus allowing malicious peers to do extra damage to the system when they defect. An example of traitors are eBay [10] merchants that participate in many small transactions in order to build up a high positive reputation, and then defraud one or more buyers on a high-priced item. Traitors may also be the computers of well-behaved users that have been compromised through a virus or trojan horse. These machines will act to further the goals of the malicious user that subverted them.

**Collusion:** In many situations multiple malicious peers acting together can cause more damage than each acting independently. This is especially true in peer-to-peer reputation systems, where covert affiliations are untraceable and the opinions of unknown peers impacts one’s decisions. Most research devoted to defeating collusion assume that if a group of peers collude they act as a single unit, each peer being fully aware of the information and intent of every other colluding peer.

**Front peers:** Also referred to as moles, these malicious colluding peers always cooperate
with others in order to increase their reputation. They then provide misinformation to promote actively malicious peers. This form of attack is particularly difficult to prevent in an environment where there are no pre-existing trust relationships and peers have only the word and actions of others in guiding their interactions [19].

**Whitewashers**: Peers that purposefully leave and rejoin the system with a new identity in an attempt to shed any bad reputation they have accumulated under their previous identity [15].

### 3.4.2 Trust Model in presence of malicious peers

The literature discussed in section 3.3.1 shows that malicious peers can be detected in a Complaint based system. However a cost would obviously be associated with rating other peers. Hence a peer may refuse to rate other peers. The solution to this is an **Incentive based** reputation system [15] where a level of incentives could be provided for reporting malicious behavior of peers or for rating other peers. The reward system could consist of credit points earned by peers when they rate other peers. This could lead to a situation where a peer gains a very high trust rating just by rating other peers. Thus some weightage could be given to credit points earned by rating.
Chapter 4

Trust evaluation for multiple context dimensions in Incentive based system

In an incentive based reputation system different context categories can be incorporated into the basic trust metric with a weight function. For example for an e-commerce transaction, peers can submit feedback about each other with different parameters such as quality of product, delivery time, payment reliability, etc. The transaction cost could also be taken into account. The final trust would then be a weighted aggregation of feedback along these different parameters multiplied by the transaction cost for high-value transactions. Similarly for a software development community peers would submit feedback along dimensions such as quality of software, Lines of Code, completion time and reliability.

We propose a context based trust model that determines the weight function for such different parameters and integrates the different extensions into our previous trust metric.

4.1 Context Sensitive Trust Model

We identify different factors for evaluating trust of a peer in different application scenarios

1. Feedback in terms of reputation of a peer as computed by peers with whom it had direct interactions.

2. Reputation computed on the basis of indirect recommendations given by unknown peers.

3. Total number of transactions and cost of transaction in e-commerce scenario.

4. Credibility of peer giving the feedback.

5. Context specific attributes associated with the transactions and current trust value of peer u as calculated by peer v based on the attributes

6. Trust value of peer u based on past history

7. Weightage to be given to recent interactions.
Weights and attributes  We assign weights with different attributes depending on the type of application. Let \( x \) be the set consisting of the different attributes. \( x = \{x_1,x_2,\ldots,x_n\} \)

Relative importance assigned to each attribute can be modelled as weight \( w_{x_i} \) such that \( \sum w_{x_i} = 1. \)

Let \( I_u \) denote the total number of transactions performed by peer \( u \) and \( P \) denote the set of peers.

\( RI_u \) is the global rating of peer \( u \).

\( RI_v \) is the rating of peer \( v \).

\( T_{uv} \) is the trust value of \( u \) as computed by \( v \)

\( C(u,i) \) is the cost associated with the \( i \)th transaction of peer \( u \)

Basic Trust Metric

In the basic trust metric we compute peer \( u \)'s trust as evaluated by every peer \( v \) with whom \( u \) has had direct interactions as well as all peer \( k \) who recommend peer \( u \) based on indirect interactions.

\[
RI_u = \sum_{v \in P, v \neq u} \frac{T_{uv} \ast RI_v}{I_u} + \sum T_{uk} \ast RI_k
\]

Peer \( v \) evaluates peer \( u \) against attributes \( x_i \) and the rating thus computed is multiplied by \( w_{x_i} \) which is the weight associated with attribute \( x_i \).

Thus \( \sum T_{uv} = \sum R_{x_i} \ast w_{x_i} \)

For e-commerce domain

In this domain we identify the attribute set \( x = \{P, D, Q, S, R, T\} \)

where

1. \( P \) \( \rightarrow \) price of product.
2. \( D \) \( \rightarrow \) Delivery time
3. \( R \) \( \rightarrow \) Reliability of payment
4. \( S \) \( \rightarrow \) prompt shipment
5. \( Q \) \( \rightarrow \) Quality of product
6. \( T \) \( \rightarrow \) quantum of trust in unknown

To account for high value transactions we add the cost of the transaction and fine tune its weightage with a constant.

\[
RI_u = \sum_{v \in P, v \neq u} \frac{T_{uv} \ast RI_v \ast \alpha \ast C(u,i)}{I_u} + \beta \ast \sum T_{uk} \ast RI_k
\]
For F/oss domain

The attribute set for this domain would consist of parameters like...

1. \( P \rightarrow \) price of software.
2. \( D \rightarrow \) Delivery time
3. \( R \rightarrow \) Lines of Code
4. \( Q \rightarrow \) Quality

Adding incentives for rating

\[
RI_u = \sum_{v \in P, v \neq u} T_{uv} \cdot R_{Iv} \cdot \alpha \cdot C(u, i) + \beta \cdot \sum T_{uk} \cdot RI_k + \gamma \cdot \frac{F_u}{I_u}
\]

where \( F_u \) is the total number of feedback ratings given by peer \( u \). \( \alpha \) and \( \gamma \) are the fine tuning constants to control the amount of weightage to be given to transaction cost and amount of reputation gained by rating others respectively. \( \beta \) is the fine tuning constant to control the quantum of trust to be placed in indirect recommendations.

4.2 Modified Framework

As shown in figure 4.1 our framework now includes two levels of policy viz global meta policy and context based policy. The context based policy gets inputs from a trust engine which is again context based.

The inputs to the trust engine are self-trust of peers, trust computed by known peers, recommendations from unknown peers and incentives gained by rating other peers. The direct trust computation has context based parameters as attributes which have weights associated with it. The weightage given to each input of the trust computation block can also be fine tuned.

4.3 Unsolved Issues

We have proposed a trust metric for multiple context dimensions. The limitations of above proposal are discussed below.

Behavioral changes

The above metric evaluates a peer based on its consistent behavior and cannot react to temporal behavioral changes. For example a peer may initially perform well until it gains a high reputation and trust value and then start misbehaving.

Overcoming Whitewashers

In order to prevent peers with a bad reputation from re-entering the group with a fresh identity, a simple solution is to assign a default low trust value to a new peer. Once part of the group the new peer must upgrade its trust value to remain in the system or else will be voted out. The drawback of this is that it might result in fresh peers from being voted out. It needs to be seen whether a dynamically calculated initial trust value can be the solution to this.
Colluding malicious peers

It is necessary to provide a balance factor in the trust metric to overcome the negative feedback provided by malicious peers. Our analysis shows that if the number of malicious peers is less then they would not have an effect on the system but if there is a large number of colluding malicious peers then the system would not be able to differentiate them from normal peers and hence the trust metric would collapse.

Overcoming Sybil attack

An attacker could subvert the reputation system by creating a large number of pseudonymous entities and use them to gain a disproportionately large influence. Currently the trust metric proposed is vulnerable to such Sybil attacks.
Reducing storage overhead and Communication Cost

To reduce storage overhead, one can determine the optimum number of control peers storing certificates. The issues that could arise here would be for how much time should a peer’s certificate be stored, if the peer is no longer an active peer in the group.
Chapter 5

Simulation and Analysis

5.1 Introduction

We would like to use our framework to study the evolution of P2P groups in the dynamic scenario where policies as well as trust is adaptive and in presence of malicious peers. As a first step towards this we modelled our protocols using a simulator in order to test the performance. The protocols for join, leave, updation of peers have been successfully implemented in the Java based simulator for peer to peer groups viz. peersim. However current simulations did not include actual interactions between the peers. Experimental runs were done with different policy rules and by varying the no of peers in the group and the composition of the group. Malicious peers have also been simulated and the graphs show that job success rate changes with change in policies as well as with change in the number of malicious peers in the group.

5.2 Peersim Simulator

PeerSim [21] is a Peer-to-Peer simulator designed to be both dynamic and scalable. The engines consist of components which may be ‘plugged in’ and use a simple ASCII file based configuration, mechanism which helps reduce the overhead. PeerSim is a library written in the Java language which consists of different components or classes which help in constructing and initializing the underlying network, which can handle different protocols, can control and modify the network. Every component can be dynamically loaded through the configuration file and almost every component can be replaced with alternative implementations.

It works in two different modes cycle-based and event driven mode. In cycle-based mode, nodes communicate with each other directly, and the nodes are given the control periodically in some sequential order, when they can perform arbitrary actions such as call methods of other objects and perform some computations. While in event-driven mode, controls have to be scheduled explicitly since there are no cycles. In this mode we can send events (messages) to protocols as per our need. Event driven mode is more realistic because generally the system will be driven completely or partially by external events such as queries by users. We have used the event-driven mode for simulation in our case.
5.2.1 PeerSim Architecture

PeerSim Architecture mainly consists of a Simulation Engine and a Network. The network of the PeerSim Simulator can be divided in 2 parts:

- **Protocols:** They are used to define the behavior of the different peers. They can be of different uses, for example handling and simulating the overlay network, or implementing an algorithm.

- **Nodes:** They represent the peer themselves in the P2P network. Every node has a stack of protocols which will define their behavior and network can be seen as the stack of nodes.

The Simulation Engine of a PeerSim Simulator consists of 2 components:

- **Initializers:** It initializes all the nodes at the start of the simulation.

- **Controls:** As name implies, controls can control the simulation, either at regular intervals or during the initialization of the simulation.
  - They can be simple observers which will gather statistics and print them.
  - They can also be dynamics which can modify the simulation itself to change its behavior.

The simulation engine of PeerSim is based on 2 modes:

1. **Cycle-based:** The cycle-based engine is based on cycles. At each cycle, the simulator goes through each node of the network and executes every protocol associated to this node. Controls are also executed periodically to control the simulation. It is based on the class CDSimulator from the peersim.cdsim package.

2. **Event-based:** The event based engine has a different way of scheduling events. Instead of scheduling the execution of the different protocols with cycles, they are scheduled through events. Events (or messages) are sent to the different protocols (for example by the control components, or by the protocols themselves), and the protocols can handle these messages and respond to them accordingly. It is based on the class EDSimulator from the peersim.edsim package. Due to the fact that it relies on messages, the event-driven simulator can emulate a transport layer, thus adding more realism to the simulation.
To give the controls to our simulation engine we have designed a configuration file which is a simple ASCII file which defines that which components will be used in the simulation and how they will interact with each other. Every comment is prefixed with a # at the beginning of the line. There is no order in the configuration file for the instructions as shown in Fig 3.3.

5.2.2 PeerSim Simulation Life Cycle

PeerSim was designed to encourage modular programming based on objects (building blocks). Every block is easily replaceable by another component implementing the same interface (i.e., the same functionality). The general idea of the simulation model is:

- Choose a network size (number of nodes).
- Choose one or more protocols to experiment with and initialize them
- Choose one or more Control objects to monitor the properties you are interested in and to modify some parameters during the simulation (e.g., the size of the network, the internal state of the protocols, etc).
- Run your simulation invoking the Simulator class with a configuration file that contains the above information.

The objects created during the simulation are instances of classes that implement one or more interfaces. The life-cycle of an event-based simulation that we are going to use in our simulation is as follows:
• The first step is to read the configuration file, given as a command-line argument. The configuration contains all the simulation parameters concerning all the objects involved in the experiment.

• Then simulator sets up the network initializing the nodes in the network, and the protocols in them accordingly as stated in configuration file. Each node has the same kinds of protocols so basically initializations set up the initial states of each protocol. In the configuration file, the initialization components are stated by prefix `init`.

• Initialization will also include the wiring of the nodes at starting which will tell us that what the network is or how the nodes are connected to each other.

• After initialization, there should be some traffic or events generated at starting so that event driven engine calls those events from the event queue and perform them. This can be done through `control` objects. The simulation ends when there will be no events left in the queue. Messages or events can also be passed to each other by nodes while performing the simulation. The control components can be stated by prefix `control` in configuration file.

5.3 Code Overview

To simulate our framework in PeerSim we designed some protocols, controls and Initializers to give to the simulator.

Protocols

The protocols that are used or made are:

1. **IdleProtocol** This is the predefined class of PeerSim library which is used to link all the nodes initially.

2. **JoinProtocol** This is the main class which implements the EDProtocol used for event driven simulation. This class contains all the functionalities of a peer used for joining, updation, leaving etc. It contains a standard method called `processEvent` which is used to process incoming messages to a peer. The main functions used to process these messages are:

   • **Authentication** : This is the function works as an authentication manager which takes the messageEvent (comes through `processEvent`) as an input. This function is called by `processEvent` only when a user requests for a join. It verifies the validity of the identity of a peer and invokes the voting algorithm or in other words sends messages to all other neighbor peers with the request of voting.

   • **Voting** : This function is called when an incoming message with a voting request comes to `processEvent`. Voting function then makes a signed vote and sends that vote through a message to the requested peer.

   • **AccessControl** : This works as an Access Manager which is used to join or update levels. In our framework it is called upon by Authentication manager at the time of joining and Updation manager at the time of updation of levels.
so basically functions **Authentication** and **UpdateManager** sends message to the respective peers with respective requests, which goes to the **processEvent** and then **processEvent** at the request of joining in a new session, vote result and update calls **AccessControl**.

- **UpdateManager**: As name suggests, this is used for updation of levels. It is directly called by peers if they want to update their levels. Peer just sends a message including UpdateREQ and his certificate to the requesting peer. Then it goes to requesting peer’s **processEvent** which then call this function for a request of update. This function then extracts rating certificates from the certificate sent in the message. It then computes the average rating from all the certificates and finally this average rating score along with the UpdateREQ will be sent to the access manager for verification from the policy manager.

- **KeyManagement**: This function is called by **AccessControl** when the re-keying of levels is required. If the requested peer is of the same level as of the requesting peer then it will directly call this function for re-keying but if the level is not same then **AccessControl** will just sends a message with the request of re-keying to the peer having the same level as of requested peer then this is called by **processEvent** of that peer when requests is of type rekey.

### Initializers

Initializers are classes which are called initially at the start of the simulation to initialize all the nodes or peers. We have made two Initializers as follows

1. **Initializer**: This class is used to initialize some nodes or peers, groups and some group policies in the network. Basically before starting the simulation we need some peers and groups already in the network and later on new peers can join using protocols that been made.

2. **WireInetTopology**: This class is used to wire or link the peers in the network. All the peers in a group should be linked to every other peer in that group. Class **WireInetTopology** extends the predefined class **WireGraph** of peerSim library and make links between peers of groups. As in our framework we are assuming that there is no link between two groups, all group works independently on their own in a decentralized manner, so there is no link between peers of different groups.

### Controls

Controls are used to control the simulation in between. Now as simulation is not user friendly, the requests of updation, joining or leaving can be given through controls only in between the simulation. So we made a control named **TrafficGenerator**.

It generates the traffic or messages of requests of joining, updation or leaving and adds these requests periodically to the **EventQueue** of EDSimulator which is taken by **Simulator** one by one in the order of the time at which they added. We can decide the order of requests or number of requests from the configuration file because it read the inputs from the configuration file at the start of simulation and assigns values to the variables accordingly.

The structure of the message can be user defined. So we made a structure of message as a class **MessageEvent**. This class consists of mainly **String requestType**, **senderNode**
(who is requesting) and a vector of messages to send with one message of Request. Request consists of three things type, role and level. These are all stored in a class named Request. This message is added in the EventQueue of the EDSimulator in the form of "requestedNode, MessageEvent". Simulator sends the MessageEvent to the requestedNode as it extracts the event from the EventQueue. The different events or requests generated by the TrafficGenerator are as follows:

- **LeaveRequest**: This request is made by a peer who wants to leave the group. We are maintaining a vector named leavingNodes containing the nodes who left, in the TrafficGenerator only. So as a peer leaves, we add it in that vector so that they can join further in a new session later.

- **JoinRequest**: This request can be made by a fresh peer or a peer who left the group previously. For joining as a fresh peer we make a new node and then add it to the network later if he is allowed to join and for joining in a new session, we randomly take a peer from the vector 'leavingNodes' and make a request of join for it.

- **RatingFeedback Request**: Feedback is given randomly by a peer to a random peer in his group. We are using the Tit For Tat strategy for that. So first it checks for the rating given to him by the peer to whom he now wants to give feedback and give feedback to him accordingly.

- **UpdateRequest**: For this we choose randomly a peer from the network and checks if it contains sufficient amount of rating certificates or not. If it not contains sufficient amount of certificates, it first makes sufficient amount of RatingFeedback requests to have sufficient number of rating certificates and then make a UpdateRequest containing the messages as we have discussed in the framework part.

**Execution**

The command that is used to run the simulation is:

```
java peersim.Simulator config-file.txt
```

The config-file that is used in our simulation is the same that we have shown in Figure 3.3. peersim.Simulator class is the predefined class of the peerSim library which reads all inputs like protocols, Initializers and controls from the config file and loads them in starting. It reads that the simulator we are using is the EDSimulator from the protocol JoinProtocol so it schedule all the events accordingly as follows:

```java
for i := 1 to simulation.experiments do
    initialize EventQueue events
    create Network

    create prototype Node:
        for i:= 1 to #protocols do
            create protocol instance
        for j := 1 to network.size do
            clone prototype Node into Network
        create (controls,initializers)
```
execute initializers

time = 0
while (time < simulation.endcycle) do
    (node, e) = EventQueue.getMin();
    node.getProtocol().processEvent(node, event)
    if (event is a control that returned true) then
        break

5.4 Simulation: Graphs and Analysis

Some simulations have been carried out for analyzing the behavior of the peers in a group. Different graphs have been taken to analyze the performance of the framework.

Graphs

Graph 1 shows that the overhead of admission of new peer or updation of level increases linearly in a group as the number of peers increases in a group.

![Image of Graph1]

Figure 5.3: Graph1

Graph 2 shows that how job success rate changes with the increase in malicious peers in the network. This graph contains two graphs with different policies. In Policy 1 the minimum voting threshold for a new peer join is higher than Policy 2. Similarly for level updation the minimum rating bound is higher for Policy 1. So in case of policy 1, the job success rate decreases continuously with increase of malicious peers because malicious peers tries to lower the ratings as well as tries not to admit the new peers. But in case of policy 2, the job success rate remains constant initially with increase in number of malicious peers and decreases very slowly later on.
This graph shows that when the no of malicious peers is low their presence does not affect the performance of the group since their feedback does not contribute highly to the group as their ratings are low. However as the no of malicious peers increases and they collude with each other, the system is unable to differentiate them from good peers and the performance of the group degrades.

This graph shows the change in behavior of the group for different meta policies. Here we maintained the percentage of malicious peers same for all the four
policies.

Figure 5.6: Graph4
Chapter 6

Conclusion

We have modelled an integrated framework for decentralized groups, which will allow us to implement secure communication between different peers with admission control and trust based access control along with key management. We propose a dynamic multi-level access control policy which is integrated with trust based access control and permits peers to collaboratively frame, and modify policies at various levels as well as update levels or introduce new levels as required. The policy framework consists of global meta level policies and context specific policies which inherit the properties of the global policy. The policies are modified based on inputs from a trust engine which is context based. We have defined trust metrics for different contexts which makes it easier to measure the trustworthiness of peers. To motivate the peers to give ratings we have given a metric which includes incentives for rating. We have successfully simulated our protocols in a peer to peer java based simulator known as Peersim and modelled malicious peers in our framework. The job success rate in terms of joining, leaving and updating levels within the group was monitored for different policies in the presence of malicious peers.

6.1 Future Scope

We would like to focus on the composition of groups in the dynamic scenario, and functionality of individual peers within the group. Future work would involve exploring the pros and cons of different trust settings and different policies. Further the anonymous nature of P2P systems exposes them to malicious activity and thus the trust decisions need to be adaptive to changes in context. The trust level of a peer may thus need to be enhanced or reduced and the authorizations appropriately adapted. A self-organizing adaptive trust mechanism capable of handling temporal and behavioral changes of peers, in which a peer can develop trust relations without using any prior information is in the scope for future work.

The evolution of P2P groups in the context of dynamic policies and an adaptive trust mechanism needs further research. We would like to see how the group role dynamics affects performance and reliability and finally provide a flexible framework for expressing policies suitable for different applications. We would like to evaluate different meta policies and see how they help us to capture the performance of the group in different context settings.
Bibliography


[23] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor: *A generalized temporal role based access control model (GTRBAC)*, IEEE Transaction on Knowledge and Data Engineering 17, 1 Jan 2005