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

Sixth Annual Progress Seminar Report

Submitted by

Madhumita Chatterjee
Roll No: 04429802

under the supervision of

Prof. G. Siva Kumar and Prof. Bernard Menezes

Kanwal Rekhi Institute of Information Technology
Indian Institute of Technology

January 17, 2011
Acknowledgement

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

Madhumita Chatterjee
Abstract

Peers with common interest often tend to create collaborative interest groups among each other. Being dynamic and self-organizing in nature in nature, these collaborative groups have constantly changing application requirements and special security requirements not addressed by typical access control mechanisms. They require a secure and reliable group communication system to ensure that they evolve and adapt according to interactions within the group. In the interests of the group there is a need for peers to collaboratively modify access control policies dynamically. The dynamic nature of peers and the changing topology of the network necessitates an environment for different functional roles of peers under the same overlay network. Another major challenge for collaborative P2P systems is the ability to manage risks involved in interacting and collaborating with previously unknown and possibly malicious peers.

To address above mentioned issues we propose an integrated framework for dynamic collaborative groups with authentication, admission control and access control. The framework supports a dynamic policy driven trust based access control model.

The framework has the flexibility to dynamically change access control policies based on the behavior and trust level of peers in the group and the current group composition. Peers can self regulate their behavior based on an adaptive trust mechanism. Our trust metric is a tunable metric which supports multiple attributes. Our model provides incentives to good peers while malicious peers are gradually isolated and eventually ejected from the group. Functionality of members in a group is also dynamic and the group can dynamically prioritize requests for join.

We simulated the framework and the dynamic policies using simulator Peersim integrated with Prolog. Our experiments and graphs show that such collaborative groups with varying compositions perform more efficiently in the presence of dynamic policies with adaptive trust, and the effect of malicious peers is mitigated.
## Contents

1 Introduction ....................................................... 1  
  1.1 Related Work ............................................... 1  
  1.2 Challenges and issues of P2P groups ....................... 2  
  1.3 Motivating Applications ................................... 4  
     1.3.1 F/OSS ............................................... 4  
     1.3.2 Multi Player Online Gaming ........................... 4  

2 Generic Model for Collaborative Peer Groups ............... 6  
  2.1 Peer Characteristics ........................................ 7  
  2.2 Group Charter .............................................. 7  
     2.2.1 Group Activities .................................... 8  
     2.2.2 Task Allocation Policies ............................ 9  
     2.2.3 Group Roles and System Roles ..................... 10  
  2.3 Group Operations and Maintenance ....................... 11  
     2.3.1 Group Initialization and Advertisement ............. 11  
     2.3.2 New Peer Join ....................................... 11  
     2.3.3 Peer evict .......................................... 13  
     2.3.4 Member Level Update ................................ 13  
     2.3.5 Member Freeze ....................................... 13  
     2.3.6 Group Removal ....................................... 13  
  2.4 Group Goals ................................................ 13  
     2.4.1 Metrics for Group Efficiency ....................... 14  

3 Dynamic Policy Driven Trust Based Access Control (DPDTBAC) ........................................ 15  
  3.1 Global policy .............................................. 15  
  3.2 Domain specific policy .................................... 16  
  3.3 Modeling Policies using Prolog ......................... 16  
     3.3.1 Multi-Level Security Policy Framework ............ 17  
     3.3.2 Modeling Rules for Join ............................ 18  
     3.3.3 Modeling Rules for Level Update .................. 18  
     3.3.4 Modeling Dynamic Policies ....................... 19  

4 Threat Model in P2P groups .................................. 21  
  4.1 Trust Model .............................................. 22  
     4.1.1 Reputation .......................................... 23  
     4.1.2 Direct Trust ........................................ 23  
     4.1.3 Indirect Trust ....................................... 23  
     4.1.4 Modeling Malicious Peers ......................... 24
5 Modeling F/OSS
5.1 Peer Roles ......................................................... 25
5.2 Task Allocation Policies ........................................... 26
5.3 Trust parameters and attributes ................................. 26
5.4 Peer Behavior ..................................................... 27
5.5 Modeling Domain Specific Policies ............................... 27

6 PeerSim Simulator .................................................... 29
6.1 PeerSim Architecture ............................................... 29
6.2 Config File ......................................................... 30
6.3 PeerSim Simulation Life Cycle ..................................... 32
   6.3.1 Protocols ..................................................... 33
   6.3.2 Initializers ..................................................... 33
   6.3.3 Controls ....................................................... 34
   6.3.4 Application Framework F/OSS ............................... 34
   6.3.5 Control ......................................................... 35
   6.3.6 How to Run ................................................... 35

7 Implementation and Results ......................................... 37
   7.0.7 Overall working of framework with prolog ............... 37
7.1 Implementation Details ............................................ 38
7.2 Simulations and Analysis ........................................ 38
7.3 Conclusion ......................................................... 44
List of Figures

2.1 Generic Model ....................................................... 7
3.1 Policy based framework ........................................... 20
6.1 General Functioning ................................................. 30
6.2 Cycle-based ......................................................... 30
7.1 Overall Framework .................................................. 38
7.2 Static vs dynamic join ............................................... 39
7.3 Static vs dynamic join ............................................... 40
7.4 Varying job allocation with easy jobs ............................ 41
7.5 Varying job allocation with difficult jobs ....................... 41
7.6 Static vs dynamic job allocation ................................. 41
7.7 30 percent malicious peers with varying job allocations .... 42
7.8 10 percent malicious peers static vs dynamic .................. 42
7.9 30 percent malicious peers static vs dynamic .................. 43
7.10 Aggregate trust values of peers with 10 percent malicious peers ........ 43
7.11 Average trust ratings of malicious peers ....................... 44
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. Some examples of intrinsic communities being formed, are Yahoo Groups [12] and Google Groups [13]. Applications like F/OSS, file-sharing, online gaming, video/audio conferencing, collaborative work-space, virtual meetings, distance learning environments, discussion forums and board rooms are examples of applications that are organized as peer groups [15] [16] [17]. Such communities or groups need a communication model in which all the peers must collaborate in order to provide the basic services such as content or messages. 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. Another major challenge is access control since decisions need to be made based on collaborations from all peers and feedback mechanisms based on trust.

1.1 Related Work

Role based access control models [3] [4] have been popular in group communication systems. Distributed establishments typically involve peers that do not know each other and have never met before. This brings in the concept of risk when peers perform transactions without knowing the reputation of those whom they are interacting with. This has precipitated work on trust and reputation mechanisms in peer-to-peer networks [5] [7] [8]. Trust is a measure of how much a peer is willing to transact with another peer. It can be defined as a peer’s belief in attributes such as reliability, honesty and competence of the trusted peer either based on the peer’s own experiences or based on recommendations by other peers. Reputation [28] is one specific way of establishing trust. It defines an expectation about a peer’s behaviour, based on recommendations received from other peer’s or information about the peer’s past behaviour within a specific context at a given time.

A number of authorization systems have been developed to provide access control to shared resources in distributed environments e.g PRIMA [20], CAS [35] and Akenti [21]. These systems do not however address purely decentralized collaborations. Several efforts have been made towards securing group communication systems, for e.g Secure Spread [23], Antigone project [22] and Secure group layer SGL [19]. Secure Spread [23] is a
secure group communication system that uses a fully distributed group key generation protocol, but it does not provide any authentication or group access control mechanisms and focusses primarily on LAN and interconnected LAN environments.

Antigone project [22] includes a flexible framework for secure group communication and utilizes a centralized member admission approach which is controlled by a session leader SL who interacts with an on-line trusted third party (TTP) in order to admit a new member. The TTP shares a symmetric key both with the SL and every potential new member. However Antigone is not designed for peer-to-peer networks.

Secure group layer SGL [19] is a secure group communication system aimed at WAN environments. It bundles a reliable group communication system, a group authorization and access control mechanism and a group key agreement protocol to provide a comprehensive and practical secure group communication platform. However the access control mechanism is not dynamic or scalable.

Some frameworks are focussed on peer to peer applications. Kim et al [1] [2] proposed an admission control framework which revolves around two basic elements, viz: a group charter which has well defined admission policies and a group authority which is an entity that can certify group admission. However their scheme lacks the attributes of peers and cannot simplify authorization in collaborative environments. Further access control has not been considered, all members have equal access rights, and neither does it integrate admission control with group key agreement.

1.2 Challenges and issues of P2P groups

Collaborative peer groups can be defined as peer to peer overlay networks with controlled membership. Such group networks share the same properties as other peer networks, such as decentralization and dynamism. Peers join these groups subject to acceptance, which maybe group specific authentication and some common interests and functionalities.

A group is a set of peers that are governed under a set of rules that describe minimal conditions to be part of the group and is formed based on a particular interest criterion. These initial or minimal set of rules or access policies form the group charter. The peers in such P2P groups can be categorized as resource nodes and user nodes. Resource nodes provide services utilizing their resources such as shared files, memory and processors for users. The roles of resources and users are dynamic as a node can either be a resource or a node or both simultaneously. For a group to survive it must adapt to the environment and to the behavior of the peers.

One main challenge of P2P groups is Admission control. A group membership control mechanism must guarantee that the group members are approved to join the group by satisfying the admission criteria. Admission criteria are group specific and form an integral part of the group security policy. A prospective member must learn these rules, hence they need to be specified in a readily available document, referred to as the Group Charter. In addition to group charter a well defined procedure to admit a new member to a group is needed. In a static group this can be achieved by a pre-defined access control list ACL. However in dynamic groups this would not work.

One way is to appoint a trusted Group Authority GAuth to handle admission procedures. This however goes against the principles of decentralization and limits scalability. Kim et al [1] [2] proposed an admission control framework which revolves around two basic elements, viz: a group charter which has well defined admission policies and a group authority which is an entity that can certify group admission. However their scheme lacks
the attributes of peers and cannot simplify authorization in collaborative environments.

A second approach is to let group members handle the admission procedures themselves. So any subset of at least $t$ members can jointly decide to admit a new member, via a voting protocol which is based on $(t,n)$ threshold signature scheme where $n \geq t$ is the current group size. In [] a distributed CA model is used where the CA private key is distributed among a specific subset of peers without complete replication. Each peer knows only part of the key and collaboration is required to retrieve it. The distribution is done using Shamir’s secret sharing scheme. In a $(t,n)$ threshold scheme, the secret is distributed to $n$ peers allowing to compute the secret with the data of any $t$ peers, but obtaining no information about the secret in case $t-1$ collude. The challenges are determining the size of $t$ in a dynamic environment. Besides, traditional secret sharing has its disadvantages as the secret shares which have been distributed remain unchanged, posing a threat to backward secrecy. Further malicious secret share holders may collude.

Another major challenge for collaborative P2P systems is the ability to manage risks involved in interacting and collaborating with previously unknown and possibly malicious peers. In order to enable practical information sharing in such decentralized and dynamic systems, a viable trust model needs to be incorporated that will allow peers to have varying amounts of dynamically changeable trust amongst each other. The main challenges that need to be addressed are: how to describe if a peer is trustworthy, what low-cost verification algorithm can be executed by a peer to determine the trust value of some other peer, how are trust values about peers exchanged within the system, how can dishonest peers be punished.

A third major problem is resource management and scheduling problem.

- How to efficiently motivate resources to provide satisfactory services to users
- How to guide users to select reliable resources in the presence of malicious resources

It is necessary for peers in a group to be able to dynamically modify the admission and access control policies based on evolving trust parameters. The changes must however be within the groups’ constitution charter, so that loosening of access control is not permitted to such an extent that the group performance degrades. Further malicious peers should not be allowed to change the group’s policies.

Thus in order for the peer groups to operate effectively additional security services must be provided. In unpredictable environments there is a need for self-organizing groups to collaboratively modify access control policies dynamically in the interests of the group.

Hence there is a need for an access control framework which should

1. Control admission of peers.
2. Permit peers to collaboratively modify policies based on group composition and evolving trust.
3. Motivate non-malicious peers to provide best possible service
4. Reduce the impact of malicious peers on the system
5. Provide facility for a user to access the best resource possible.

The dynamic nature of peers and the changing topology of the network further motivates the need to provide an environment for different functional roles of peers under the
same overlay network. One possible approach to handle the dynamic and unreliable behavior of peers in such dynamic groups is to provide self organizing peer services or roles. These roles would differ in the context of a peer group. Peers could be allowed to dynamically make decisions based on specific conditions and assume additional functionality to ensure that the group reaches a certain satisfaction level.

With above issues in consideration, we propose a decentralized and integrated framework for authentication, admission control and access control in peer groups. We model a Policy Driven Mechanism for access control which integrates Trust and Reputation. Our model permits deployment of access control policies flexibly and dynamically. New polices can be added dynamically by peers within the framework of the original group charter.

Our trust metric is a tunable metric based on context specific attributes. 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. Peers can collaboratively modify policies at their level based on current group composition and trust level of existing peers. Thus low level join requests could be postponed in a group which has a certain threshold of peers already at the same level, by changing the join policy dynamically. Our model provides incentives to good peers while malicious peers are gradually isolated and eventually ejected from the group. Our framework allows us to test the evolution of peer groups based on different policies. Our experiments show that dynamic polices based on the adaptive trust and changing group composition lead to better group efficiency as compared to static access control policies. We further show that the effect of malicious peers is mitigated and such peers are gradually ejected from the group.

1.3 Motivating Applications

1.3.1 F/OSS

One motivating example is the Free and Open Source Software (F/OSS) self-organizing P2P group F/OSS[45] which represents an approach for communities of like-minded participants to create, distribute, acquire and use software and software-based services that are intended to be shared freely, rather than offered as closed commercial products. Participants within these communities often participate in different roles like core developer, module owner, code contributor, code repository administrator, reviewer or end-user. They contribute software content like programs, artifacts, and code reviews to Web sites within each community, and communicate information about their content updates via online discussion forums, threaded email messages, and newsgroup postings.

Active developers in such communities join and propose patches to existing code. The system gives opportunities to developers to make contributions and enable them to adjust their actions based on evolving code. Projects are evaluated by moderators on the basis of technical merit and elegance of contribution. When a developer with a high reputation develops a new application or a thread of development, others are attracted resulting in more threads. Such systems and communities co-evolve with the contribution of a large number of participants [44].

1.3.2 Multi Player Online Gaming

Massively Multiplayer Online Games MMOGs are computer games capable of supporting thousands of players simultaneously and are typically played in a giant persistent world. In massively multiplayer online role-playing games (MMOs), main aspects of game-play
involve grouping with fellow players to defeat monsters and complete quests. During the process, group members get experience points to advance their character levels and build their fighting skills. Game groups are similar to those in the real world in terms of members with similar and, in some cases, complementary expertise collaborating with one another to achieve some collective outcome. Further, the fast changing game environment makes the groups in online games dynamic and short-lived - a feature increasingly observed in the real world. Combat groups are usually formed to complete a set of tasks and get specific rewards. For example, several players may form a group to kill an extremely strong monster or complete a difficult quest together. The group will typically disband after finishing the task. Without pre-determined boundaries, the groups form, change, and disband quickly according to the nature of tasks, ability of player characters, and the changing environment.

Thus the notion of a group is well defined as an in-game feature where players have the option to invite others to join them to form a group. These groups can then perform various in-game tasks together and the game software ensures that any rewards and experience points gained during this period are shared by all the group members. The tasks can range from simple to elaborate quests, players’ guild operations, and pretty much anything under the umbrella of shared personal goals.

These type of decentralized and dynamic peer groups require a secure group admission policy and an adaptive access control mechanism where peers can collaboratively frame and revise access control decisions, based on behavioral attributes, current group composition as well as trust evolved in the group.
Chapter 2

Generic Model for Collaborative Peer Groups

In an electronically connected world, network-addressable computing elements such as a desktop personal computer, a laptop computer, a personal digital assistant, and so on can be termed as peers. Peers have comparable roles and responsibilities and are used by their owners to communicate information, share or consume services and resources with other peers whom they know. While a group is a physical collection of objects, a community is a set of active members, who are involved in sharing, communicating and promoting a common interest. Communities are formed implicitly, i.e. they are self organizing. Peers are implicitly grouped into communities based on the common interests they share.

We propose a security framework for collaborative peer groups where self organizing peers aggregate in a specific manner and use new communication primitives to achieve their specific tasks. Previous basis for peer to peer group formation has been ”symmetry” where peers are always capable of performing equivalent tasks. Our model proposes a framework for peers with different functional capabilities to form special interest peer groups. We achieve peer authentication, controlled admission and dynamic access control. Our system model consists of a set of peers, a set of activities performed by peers, a set of events (external and internal) and a set of processes. The peers in our model are governed by a set of rules defined in the Group Charter, that describe the minimal conditions to be part of the group. The collaborative groups are formed by sets of peers with some common interest and controlled membership. The framework has the flexibility to change rules and make it adaptive. Peers can self regulate their behavior based on learning experience in the group.

Assumptions

We make the following assumptions for our model:

- The system is asynchronous i.e there is no global clock.
- Domain specific rules and policies reside in the local memory of a peer.
- Reliable message delivery is assumed.
- No peer symmetry, i.e peers do not have equivalent functionalities.
- Peers can be queried about their group specific activities/capabilities and other information.
• No distinction is made between node failure, link failure and unannounced exit of a peer.
• Every group must have a well defined Group Charter at the time of creation, which is updated periodically.

2.1 Peer Characteristics

A peer can either be a service provider or a rater. Some specific characteristics of peers in our model are:

• Peers have an initial self proclaimed rating which is the initial trust value
• Peers are assigned different roles based on their functionality.
• Peers in a group can belong to different levels based on capability. A Group Role assigned to 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.

![Generic Model for collaborative groups.](image)

Figure 2.1: Generic Model

2.2 Group Charter

Each group has a group Charter with information about:

• Group Description
• Group Activities
• Meta Rules for membership
• Group Composition

We define two types of policies, Global and Domain Specific. The global meta policy describes the roles permitted in the group and the rules associated with each role. Domain specific policies could change with the application domain. They could decide on the optimum number of peers required at each level. A well formed group charter could have some initial rules as follows:

• Well defined rules for adding new peers and discarding malicious peers.
• Rules for sharing resources that exist among different peers based on the trust value associated with the peers.
• Framing group policies like minimum trust levels to be maintained.
• Rules for upgrading trust level of a peer or degrading trust level.
• Rules for creating and destroying a group.

2.2.1 Group Activities

We broadly classify the set of activities performed by peers as:

• Group Management Activities
• Task Oriented Activities
• Social Tasks
• Malicious or Destructive Activities

Group Management Activities

These can be listed as follows:

• Admission Control
• Access Control
• Publishing of Group Existence/Charter
• Creation of Groups
• Removal of Groups
• Resource Management
• Scheduling Tasks
2.2.2 Task Allocation Policies

The group has different tasks or modules at different difficulty levels. These tasks could be totally independent, sequential or concurrent or there could be some dependency among concurrent tasks. Say a peer could be allocated job C provided he has earlier met some pre-requisites. Task scheduling is done amongst peers based on peer capability, difficulty level of tasks and completion requirements. Consider a scenario where there are m peers with available resources needed by x peers for job completion. Each time K resources are generated by the system and each resource node can process only one job per time. If k \( \geq m \) then a job allocation policy could be

- Allocate jobs to resources depending on their trust values/roles without considering the complexity of jobs.
- Allocate jobs depending on the complexity level.
- Allocate jobs depending on deadline to finish.
- Allocate jobs depending on security policy, i.e whether it needs encryption or not.

Task Oriented Activities

These can be classified as:

- Procedural (function/task of Groups i.e Domain Specific)
- Information/Resource request.
- Opinion/Reputation request (Trust)
- Opinion/Reputation response
- Record Maintenance (of Group activities)
- Maintenance of certificates, keys, ratings.
- Voting
- Evaluator/critic (measures group activities against some Group Performance Index GPI)

Social Activities

Those activities which contribute towards improving the overall performance of the group fall in this domain.

- Reward actions like providing incentives for correct ratings.
- Punish actions for providing poor service or giving inconsistent ratings.
- Observer actions i.e maintaining records of those group activities that help in providing feedback.
Malicious activities

One of the major challenges of collaborative P2P groups is the ability to manage risks involved in interacting with malicious peers. We model malicious activities as:

- Aggressor behavior i.e peers who lower the rating of other peers.
- Recognition seeker i.e colluding peers who call attention to self by providing very good behavior for some time and increasing each others ratings.

2.2.3 Group Roles and System Roles

A peer can either a service provider or a rater. Quality of a peer as a service provider is independent of the quality of the peer as a rater. The additional functions that a peer is capable of performing in a group are storing and verifying certificates, authentication, voting, updating 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 define two types of Roles viz Group Roles and Domain Specific Roles.

Group Roles

A peer can take on 3 different Group roles.

- *Member peer* A peer in this role is a minimal functionality peer who participates in the normal group activities but does not contribute to the admission of new peers, nor updating of levels of existing peers. Thus this peer is only a service provider.

- *Admission Peers* Those which are allowed to register new group members.

- *Control peer* The control peers are the super nodes of the framework. Every group must have at least one control peer. These peers are responsible for broadcasting essential messages like
  1. Joining of a new peer
  2. Policy change(if it happens)
  3. Updated ratings of peers.

Thus a peer in this role would have all the functional components and would participate in periodically updating levels of existing peers and also permitting change of role. These peers are also responsible for monitoring group activity and keeping a track of group performance index GPI. If GPI falls below a certain threshold then a CP can call for a consensus of AP’s or MP’s to decide on a policy change. Framing of new policies like adding a new level to the group can also be done by a CP.

Domain Specific Role

These are system roles at hierarchial levels, depending on the application domain, access rights/capability and trust value. So, for example for a e-learning application the roles could be teacher, student, T.A or Grader, Examiner, Candidate whereas for F/OSS application the roles could be Developer, Reviewer, Moderator.

We define Users/Peers(Pi), Roles(Ri), Permissions assigned(PA), Permissions(P).

\[ R = DR \cup GR \text{ where } DR = \text{domain role and } GR = \text{group role and } PA \in PxR \]

10
Change of Roles

At any time a member peer may want to take up the role of an Admission Peer or control peer. Since the model focusses on the self-organizing and collaborative nature of peers, in order to achieve group efficiency, any peer which is part of the group may apply for a role change or level change. As peers belonging to CP or AP perform more operations, nodes with sufficient bandwidth or computing power may apply. Similarly if a CP decides to change its role to MP for some reason, such a peer would lose group membership unless it can transfer control to some other peer. If a peer is a single CP then it cannot be allowed to change its role. Thus change of role is also part of the global group policy.

Set of Events/Processes

Our framework models both external and internal events. External events are Join, Leave, while Internal events are Update, Task Allocation and Scheduling, Policy change. The schedule of the policy change can be either event driven or periodic. We assume an event driven schedule.

2.3 Group Operations and Maintenance

2.3.1 Group Initialization and Advertisement

A new group can be created by a peer willing to function as a control peer. This peer then specifies the initial group charter and advertises the existence of the group along with documents that specify the parameters of the group, such as group type, admission policies, group name, group members etc. Once additional peers join the group, a member list is maintained either by the creator or by other control peers and this list is published. The group advertisement should be periodically broadcast by the peers of the group.

2.3.2 New Peer Join

A peer Pi may become a member of group G upon invoking the join protocol. In our framework, the admission peers are responsible for join operations. Peers are allowed to join as per an access policy. The policy specifies whether the new per has access to communications before he/she joined. It also specifies the level of cryptographic security needed for the new peer to access existing resources. Once a join is done, information about the new node is broadcast to the overlay. The peer joining the group is provided with the current membership view known as the local\textunderscore view.

Protocol for New Member Join

The joining of a new member would have the phases as detailed below:

Admission Request

A new peer wishing to join a group would first search for an advertisement of a group that matches his requirements. He would then submit a signed request to any peer which would include his credentials that he obtains from a CA say X.509 certificates or self-generated certificates, along with a request for the role that he wishes to join in.

For a peer to authenticate itself to a peer group, we state that just submitting his identity is not sufficient. Given that each peer has his own certificate which could be self signed or
signed by a CA, a peer credential could be created by hashing the concatenation of unique user ID UUID and public key fields and then signing this hash with the private key of the user and using this digital signature as the identity of the peer. This identity could be used as the peer’s credential in the messages.

\[ P_{\text{new}} \rightarrow P_i : \{\text{JoinREQ}\}^{SK_{\text{new}}}, \text{Cert}_{\text{new}} \]

where

\[ \text{Cert}_{\text{new}} = \text{UUID}_{\text{new}}, PK_{\text{new}}, \text{Rating}_{\text{new}}, \{H[\text{UUID}||PK_{\text{new}}]\}^{SK_{\text{new}}} \]

For a new user his Rating$_{\text{new}}$ field will contain a single value signed by himself.

**Peer rating certificate RC**

A rating certificate can be used as a means of recommendation. This certificate contains the original trust value of the peer along with the recommended rating value given by each peer. The recommendation is signed by the recommending peer thus preventing a peer from being able to modify the recommendations/ratings given to him by other peers. The rating certificate also contains an expiry date to prevent the recommended peer from recycling good rating certificates beyond the specified period. When a certificate is about to expire, the peer can contact the recommending peer and get the validity of the certificate extended. Thus for each peer it has interacted with, the peer needs to keep a copy of the certificates it issued to that peer for validation purpose and for trust and contribution score updating. In addition a peer could maintain a revocation list of peers who are not to be trusted and periodically this information could be exchanged amongst peers. The fields in the rating certificate are Recommending peer’s identity, Recommended peer’s identity, Original trust value, Issuing date and time, Expiry date and time, Contribution score and Signature of recommending peer.

**Authentication**

The peer receiving this signed request will obtain the identity of the requesting peer from the Certificate. If it is a signed certificate then the verification is easy. If however the Public Key pair is pseudo generated then the receiving peer will compute the hash of UUID and PK$_i$ and tally this with received message. If the two hashes match then the user is authenticated and will be granted access permissions based on the access control policy. If the user’s credentials contain a rating field with a single entry he will be treated as a fresh user and the voting process will be invoked by broadcasting this join request to the other peers. The certificate of the new peer is co-signed by the broadcasting peer, so that further verification of ID is not required by other peers.

\[ P_i \rightarrow \text{All} P_h : \{\text{JoinREQ}\}^{SK_{\text{new}}}, \{\text{Cert}_{\text{new}}\}^{SK_h}, \text{VoteReq} \]

**Voting and Authorization**

The peers all verify the authenticity of the new peer from his credentials. The peers return the results of voting to the peer who had initiated it.

The initiating peer verifies the votes and checks against the access policy for join. If the votes are $> t\%$ and the other access criteria are met, then the later then gives a signed membership token to the new peer and allows this peer to join. The threshold $t$ could be modified collaboratively by peers depending upon group performance and efficiency.

The entry of the new peer is broadcast to all peers of the group.
2.3.3 Peer evict

A peer pi may be evicted from a group G upon invoking the leave protocol. The leaving peer normally broadcasts its information to the overlay and retires from all maintenance functionalities.

2.3.4 Member Level Update

Once in the group peers are periodically rated by other peers in the group and signed rating certificates are saved by peers in their local rating history table. Expert peers store the ratings of all peers in their level and can update the ratings of peers in their level. Each peer has a maximum intrinsic potential MIP and can rise only upto that potential. Since the rating certificates are signed by the recommending peers they cannot be modified. Depending upon an access policy a peer can be elevated from his current level to a higher level, or can be demoted to a lower level. The access policy itself can be collaboratively modified by the control peers, if need arises. For e.g in a multi-player gaming scenario, if the current policy to update from level 1 to level 2 requires say 50 game points and there are too many participants who are achieving it then the peers may decide to make the update criteria more stringent.

2.3.5 Member Freeze

If a peer has not been participating in group activities for a long time, the system does not send it any further key updates.

2.3.6 Group Removal

This is done by shutting down the group overlay, so that no nodes remain. A control peer is needed for this task. Group and membership information is removed from the overlay.

2.4 Group Goals

For a group to survive it must adapt to the environment and to the behavior of the peers. One of the major challenges for collaborative P2P systems is the ability to optimize performance while collaborating with previously unknown and maybe malicious peers. The goals of collaborative peer groups can be summarized as

- Admission control and access control
- Secure Group operations
- Resource Management and Scheduling
- Optimum performance in presence of malicious peers

For self-organizing groups it is necessary for peers in a group to be able to self regulate and dynamically modify the admission and access control policies based on evolving trust. The changes however must be within the group’s constitution charter, so that loosening of access control is not permitted to such an extent that group performance degrades. Further malicious peers should not be able to change the group’s policies.
2.4.1 Metrics for Group Efficiency

We define some of the metrics for evaluating group efficiency as:

- Evolution of the group
- Current Aggregate trust
- Current Group composition
- Past history
Chapter 3

Dynamic Policy Driven Trust Based Access Control (DPDTBAC)

Authorization deals with issues like who can access which resources/services under which conditions. Many mechanisms e.g Role based, Rule Based, Identity Based access controls exist. However these authorization mechanisms alone cannot satisfy the access requirements of distributed services as access depends on many factors like privacy requirements of the requestor, authentication requirements of the service, trust relationship with the requestor, authorization and management policies among participating parties etc.

Authorization in a distributed environment should be determined as a result of evaluating the request of an authenticated user against various policies like privacy policy, trust policy, authorization policy and so on.

Our policy driven model allows us to deploy access control policies flexibly and dynamically. Based on the behavior and trust level of peers in the group and the current group composition, it is possible for peers to collaboratively modify policies governing their level. The group policy can also dynamically prioritize requests for join. Join priorities would depend on current group composition. Thus low level requests for join could be postponed in a group which already has a large number of members performing the lower roles. Peers in one level are also allowed to be dynamically updated to a higher level. Update policy also depends on current group composition and behavior. A peer could also be collaboratively ejected from a higher level to a lower level or even out of the group if his behavior in the group degrades. It is also possible for peers in the highest level to introduce a new level in the group if the situation so demands.

Multiple policies can apply to a group, and 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.

We define two levels of policies viz:Global Policies and Domain Specific Policies.

3.1 Global policy

This is the policy initiated by the creator of the group. It includes rules for admission, updating and leave as well as rules for deciding the no of control peers needed to collaborate to frame new rules, or change existing rules. Some sample rules are:
• A peer can join as a member peer at level 1 of the group if he has received votes greater than 50% and if his current rating is greater than some threshold x.

• To update level a peer must have an accumulated trust value > x and should be an authenticated member at the lower level.

• A new level could be introduced if at least y no. of peers are present in the current highest level and all 100 % of them collaborate.

• For a new policy to be framed at least 75% of existing peers with trust value > some threshold x must collaborate. This ensures that malicious peers cannot frame new policies.

The initial policies are framed by the creator of the group and a hash of the global policy is also stored in the Group Charter.

3.2 Domain specific policy

This set of policies inherits the properties of the global policy and contains application specific policies. Say for example if we had a group for some multi-project software ecosystem and publish/subscribe etc, where peers could join as developers, reviewers, or moderators depending on their capability level. Further depending upon additional functionality each peer wishes to perform in the group a peer has different roles. A peer is allowed to elevate his level in the group based on some policy. Domain specific policies decide the optimum number of actors required at each level. Maybe some policy could prune down no of peers in a particular level/role to optimize group performance. Another domain specific policy could be the framing of adaptive trust policies, i.e the group would work with trust framework TF1 until it reaches a particular state and then adapt TF2 and so on.

3.3 Modeling Policies using Prolog

In policy driven TBAC, the policy description language besides being expressive and easy to use must be flexible enough to allow extension of a policy by adding or modifying its constraints easily, without affecting existing policies. Cassandra[35] is an authorization language that defines the actions of activating a role and deactivating a role. Users can thus write state-dependent and implicitly state manipulating policies, but this rather ad-hoc approach is inflexible. Some languages such as Ponder[36] support obligation policies. An obligation is a task to be executed after evaluating and enforcing an access request. However it does not provide a precise semantics for the state changes. Some work has been done on analyzing security properties in dynamic role-based systems, in the context of the role-based authorization language RT[37] and Administrative RBAC (ARBAC)[38], where members of administrative roles can modify the role membership and privilege assignments. In [41], policies written in Datalog can refer to facts in the authorization state, as in our model. Hezberg et al. propose in [?], a prolog-based trust management language, but do not focus on dynamically changing policies with the state of the environment.

We use a logic programming system to realize our access control policies. We capture the policies using Prolog rules. A prolog rule is an expression of the form

\[ Ro(uo):-R1(u1) \ldots \ldots Rn(un) \]
where $R_i$ are predicates and $u_i$ are (possibly empty) tuples of variables and constants. The head of the rule is $R_0(u_0)$ and the sequence of formulae on the R.H.S is the body of the rule. If a knowledge base contains a rule head :- body, and Prolog knows that body follows from the information in the knowledge base, then Prolog can infer head. This fundamental deduction step is what logicians call modus ponens. There are only three basic constructs in Prolog: facts, rules, and queries. A collection of facts and rules is called a knowledge base (or a database) and Prolog programming is all about writing knowledge bases. That is, Prolog programs simply are knowledge bases, collections of facts and rules which describe some collection of relationships that we find interesting. One uses a Prolog program by posing queries. That is, by asking questions about the information stored in the knowledge base.

A policy is a set of Prolog rules. We chose Prolog as our policy description language because of the following features.

- It is declarative. A rule in PROLOG defines a relationship between several objects.
- It is based on a subset of First Order Logic, thus properties such as soundness and completeness can be guaranteed.
- It supports back tracking and can express non-deterministic constraints.
- It is a unification based language which allows writing policy templates.
- It is a productive modeling language supporting incremental policy writing and refinement.
- It is possible to reason from a set of Prolog rules, supports meta-level reasoning, thus making policy conflict detection possible.
- Dynamic rules can be modelled using the assert and retract clauses.

The Prolog inference engine provides a mechanism to derive consistent access control decisions at runtime. It may also be used to analyze the correctness and consistency of access control and other rules. Access control rules, defined in Prolog, can be more expressive than the traditional (subject, object, action) tuple. The access control policy is machine readable and directly under the control of the administrator. It also possesses dynamic updating capabilities. Dynamic rules can be modelled using the assert and retract clauses. New facts from independent policy sources can be added to the policy base before decisions are made, ensuring dynamic decisions at runtime. A policy interacts with its dynamic environment by consulting facts in the environment and constraining certain actions in the environment. In order to ensure that totally new policies outside the original group charter cannot be framed dynamically, we maintain a hash value of the original group charter with every peer. Thus a peer can at any time verify whether the policies being applied are as per the constitution framework.

### 3.3.1 Multi-Level Security Policy Framework

We give here a Prolog based model to represent various policies like security polices for access control. Broadly speaking, a security policy must address issues like Confidentiality, Integrity and Availability. We give some notations and definitions associated with our policy framework.
A group can consist of different levels of hierarchy within it. A level sets a logical boundary for a subset of peers who can communicate with each other according to an individual security policy defined for that level. Each level thus has its own individual security policy. There could be communication between peers of different levels, which is handled by an inter-level multi policy framework. So for e.g an interaction between a peer p1 accessing another peer p2 through access operation op(e.g read and write) can be defined with a policy P(p1, p2, op) which is of type grant or reject.

Level_p is used to denote the domain belonging to P, which consists of all entities i.e peers that are submitted to P. For any access (p1, p2, op), a policy P will contain an access rule if and only if p1, p2 ∈ Level_p.

We use a closed security policy, where only allowed operations are specified or only authorized entities are allowed access. Operations to be denied need not be explicitly specified since Prolog’s negation-by-failure mechanism enforces a default denial on messages other than those explicitly specified in the knowledge base and inference rules.

We define Prolog Facts as entities, levels, roles and allowed operations. Entities are basically peers (can be humans or nodes).

Entity_Set = \{UUID, PK\}
Role_Set = \{GRUSR\}
GR = \{MP, AP, CP\}
SR = \{faculty, staff, student, TA\}
Operation_Set = \{Read, Write\}

3.3.2 Modeling Rules for Join

An example of rules for peer with unique identity to join as a member peer in a group at level1 of a groups is

\[\text{verify}(Npeer, Request, Rl, Level, Rate, SPR) :\]
\[\text{join}(Request), \text{member}(Rl), \text{Level} =:= 1, \text{Rate} >= 40, \text{assert}(\text{belongs}(Npeer, Level)).\]
\[\text{verify}(Npeer, Request, Rl, Level, Rate, SPR) :\]
\[\text{join}(Request), \text{member}(Rl), \text{Level} =:= 2, \text{Rate} >= 3, \text{assert}(\text{belongs}(Npeer, Level)).\]

Verify predicate takes 6 parameters as Peer ID, type of Request, Role, Level, Overall Rating and Self-proclaimed rating. If the RHS is true for all the given cases then the predicate returns true. A more stringent join policy which ensures that the percentage of expert peers is greater than 20 for a total group size upto 200 is

\[\text{verify}(Request, Rl, Level, Rate, SPR, Expert, Tot) :-\]
\[\text{join}(Request), \text{admission}(Rl), \text{Level} =:= 2, \text{SPR} > 50, \text{Expert} > 20, \text{Tot} < 200.\]

3.3.3 Modeling Rules for Level Update

When a member peer requests for level update, the policy verifies the current level of the peer and checks his trust rating. Two sample rules are
verify(Npeer, Request, RL, Level, Rate, SPR) :-
belongs(Npeer, Level - 1), update(Request), admission(RL), Level =:= 2, Rate >= 5,
retract(belongs(Npeer, Level - 1)), assert(belongs(Npeer, Level)).

verify(Npeer, Request, RL, Level, Rate, SPR) :-
belongs(Npeer, Level - 1), update(Request), maximal(RL), Level =:= 3, Rate >= 8,
retract(belongs(Npeer, Level - 1)), assert(belongs(Npeer, Level)).

3.3.4 Modeling Dynamic Policies

We are able to express dynamic rules at runtime, as well as change existing policies depending on the status of the group at runtime by exploiting the assert and retract clauses of Prolog. A rule is asserted in prolog when all the clauses on the L.H.S of assert are true. One sample rule to prioritize join requests if a peer requests to join the group as a member peer after the group has evolved for some time and the number of member peers in the group is greater than 50% of current group size then the previous join rule is deleted from the database and a new join rule is asserted, thereby postponing low-level requests for join. Similarly other rules for checking no of control peers.

\[
\text{update} \text{engine}(\text{Levels, MPs, APs, CPs}) :- \text{CPs} > 20, \\
\text{retract}((\text{verify}(\text{Npeer, Request, RL, Level, Rate, SPR}) :- \\
\text{update}(\text{Request}), \text{maximal}(\text{RL}), \text{Level} =:= 3, \text{Rate} \geq 7)), \\
\text{assert}((\text{verify}(\text{Npeer, Request, RL, Level, Rate, SPR}) :- \\
\text{update}(\text{Request}), \text{maximal}(\text{RL}), \text{Level} =:= 3, \text{Rate} \geq 9)).
\]

\[
\text{update} \text{engine}(\text{Levels, MPs, APs, CPs}) :- \text{CPs} < 20, \\
\text{retract}((\text{verify}(\text{Npeer, Request, RL, Level, Rate, SPR}) :- \\
\text{update}(\text{Request}), \text{maximal}(\text{RL}), \text{Level} =:= 3, \text{Rate} \geq 9)), \\
\text{assert}((\text{verify}(\text{Npeer, Request, RL, Level, Rate, SPR}) :- \text{update}(\text{Request}), \text{maximal}(\text{RL}), \text{Level} =:= 3, \text{Rate} \geq 7)).
\]

Here update_engine function can be used to update the new rule by retracting the old rule and then assert a new rule. update_engine takes 4 inputs as Levels(no of levels), MPs(% of member peers), APs(% of admission peers), CPs(% of control peers).

To address privacy issues the domain plays an important role. Domain or context specific policies refer to the set of rules/regulations/requirements of a domain to which an entity must conform to in order to be in that domain. To implement privacy based access, a variable can be attached to the information to indicate the privacy level of information. So if full privacy is chosen then the information will be sent with encryption and signature. Domain specific policies are part of a dynamic policy database and could be modified by peers, depending on peer behavior, group composition and evolving trust of peers in the group.

Thus we have a policy based framework which interacts with the evolving trust parameters of peers in the group. Figure3.1 depicts the interaction between global policies, application specific meta policies and the trust engine for peers having multiple roles in multiple levels. All peers maintain their local view of the policies and any policy change is broadcast by the control peers.

Good peers rebuke the misbehaving peers by degrading their trust value and including them either in the doubted list or in the black list. Transaction is avoided with the peer
Figure 3.1: Policy based framework

in the black list. Thus, malicious peers are slowly isolated from the network. Incentive is provided to the good peers who give good service by increasing their trust value.
Chapter 4

Threat Model in P2P groups

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. 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. Some of the adversaries are

1. Honest Peers. These peers are not malicious. However they try to achieve monopoly earning a high trust by giving best service always. Could prove to be a threat if it later becomes malicious
2. Naive peers These peers try to gain trust by promising good services but always provide bad services
3. Hypocritical peers 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. They could be random or structured. Random peers act maliciously with some probability say 20%. Structured peers follow a fixed pattern, say provide one malicious service after four good ones.
4. Colluding peers Peers could collude and form a special group. They act differently depending on whether requesting peers are from within the group or outside it, and normally provide malicious service to an outsider.
5. **Front peers** 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 [30].

6. **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 [26].

Our hybrid access control model is based on an integration of collaborative roles and trust [3, 4]. The trust parameters in our model build upon the PeerTrust[39] model. We specify context based attributes for specifying trust and assign specific weights to these attributes. We further define a tunable trust metric to combine all trust parameters viz: direct trust, indirect trust based on recommendations and credibility of the peer. As peers perform in the group, trust of each peer evolves based on the interactions. A role represents a set of privileges and rights. It involves meta-level policies in regard to admission of users to the role. Integration of the roles, tasks, privileges and security policies needs a proper framework for policy specification. We are able to express dynamic rules at runtime, as well as change existing policies depending on the status of the group at runtime through Prolog.

### 4.1 Trust Model

Trust is a peers belief in attributes such as reliability, honesty and competence of the trusted peer. Trust evolves with time and experiences. Positive experiences can lead to increase of trust while negative experiences can lead to decrease of trust. Trust can also vary over time. Reputation is an expectation about an individual’s behavior based on information about or observations of its past behavior [9]. Traditional trust models only consider the reputation accumulated by peers’ long-term behaviors, and are not adaptable to dynamics of the behaviors.

Reputation-based systems like ebay rely upon aggregated feedback to evaluate participant’s trustworthiness. However it does not consider various aspects like the merchant’s service quality such as price, delivery time etc. P2P Trust model proposed in [?] calculates trustworthiness of a peer using direct reputation, and indirect recommendation. Peertrust[39] is an adaptive reputation based trust model which includes different parameters like credibility, transaction factor and cost into the trust metric.

We propose a layered model with a hierarchy of privileges which builds upon the Peertrust model. We incorporate various attributes for feedback, into the basic trust metric with a weight function. We include a decay factor as trustworthiness is not always the same and may change with time. Our trust model along with our dynamic trust policy helps us to slowly remove malicious peers whose behavior is dependent upon a specific pattern. We compare the results of our model with the P2P Trust model[40] and show that our model handles malicious peers more efficiently.

The Basic trust metrics in our model are:
4.1.1 Reputation

This is a metric measuring overall quality of previous transactions between peer x and y. We represent \( f_E(x, y) \) as the feedback given by peer y for peer x for a transaction. Let \( a_i = a_1, a_2, a_3, \ldots, a_n \) be the set of attributes used for computing the feedback. Then

\[
\begin{align*}
    f_E(x, y) &= (f_{a_1}(x, y), f_{a_2}(x, y), f_{a_3}(x, y), \\
    &\quad \ldots, f_n(x, y))
\end{align*}
\]

where \( f_{a_i}(x, y) \epsilon [0, 1] \) is the feedback score given by peer y about peer x for attribute \( a_i \). Relative importance assigned to each attribute can be modelled as weight \( w_{a_i} \) such that \( \sum w_{a_i} = 1 \). Assume that peer y stores upto n feedback ratings of previous transactions with peer x. Then reputation of peer x is calculated by peer y as

\[
R(x, y) = \frac{\sum_{i=1}^{n} \alpha^i \cdot f_{Ei}(x, y)}{\sum_{i=1}^{n} \alpha^i} \tag{4.1}
\]

where \( f_{Ei}(x, y) \) denotes the ith feedback given by peer y to x and \( \alpha \epsilon [0, 1] \) is a decay factor which indicates how important the most recent interaction is to the user. Each user may set a different value of \( \alpha \) for itself.

4.1.2 Direct Trust

This is derived from a peer’s all Direct transaction experience, their credibility and the number of times the peer has been interacted with them.

\[
T(D) = \frac{\sum I(x, y) \cdot R(x, y) \cdot Cr(y)}{\sum I(x, y) \cdot Cr(y)} \tag{4.2}
\]

where \( R(x,y) \) is the reputation which can be calculated using equation 1, \( I(x,y) \) is the number of times peer x interacted with peer y, \( Cr(y) \) denotes the credibility of peer y.

4.1.3 Indirect Trust

If a peer does not have sufficient number of transactions with a resource it can ask the opinion of other nodes to gain an overall evaluation \( T_{ID} \). Assume that each user x receives job ratings for peer y from n references \( k = (1, 2, \ldots, n) \) and all nodes use the same decay factor \( \alpha \). Then user x can compute the reputation \( f_E(k_i, y) \) of each indirect peer \( k_i \) to resource y.

\[
T_{ID}(x, y) = \frac{\sum_{i=1}^{n} I(x, y) \cdot R(k, y) \cdot Cr(k)}{\sum_{i=1}^{n} I(x, y) \cdot Cr(k)} \tag{4.3}
\]

Context Factor

To add incentives for rating we add the factor

\[
F_x = \frac{F_x}{I(x, y)}
\]

where \( F_x \) is the total number of feedback ratings given by peer x. It is a positive factor for correct ratings and a negative factor for incorrect ratings. Thus we increase the credibility factor of peers who give proper recommendations and decrease the credibility of peers giving incorrect recommendations.
Credibility of evaluating peer

The evaluator’s credibility is important for a peer to decide whether to accept the reputation value or not. We assume that the current trust value of a peer is his credibility. If an evaluator has credibility beneath a certain threshold, his evaluation will be ignored. The group dynamically decides on the selection threshold for the credibility to protect against malicious peers.

Global Trust value

Let $T$ be the global trust value of a peer $x$. Then $T$ is an aggregation of the direct experiences of every peer about peer $x$ as well as the recommendations received about peer $x$, and the context factor.

$$T = \alpha * T_D(x, y) + \beta * T_{ID}(x, y) + \gamma * Context\text{factor}$$

where $\alpha$ is the weight associated with direct experience, $\beta$ with indirect reputation and $\gamma$ is a fine tuning constant to control the amount of reputation gained by rating others.

Each peer (MP, AP or CP) has its own self proclaimed rating $SPR$ and collects trust ratings from other peers in the group with whom it has Direct Interactions. The ratings are signed with the private key of the recommending peer. Final trust value of a peer is computed by a CP when a peer requests for updating of level.

4.1.4 Modeling Malicious Peers

Malicious peers are not only peers that provide incorrect/damaging information but also are peers that use unfair methods to lower the trust values of their neighbors. We consider 3 types of malicious peer behavior

- **Simple malicious peer.** We model these peers as those who under-perform in the task allocated to them in the group, and provide poor ratings to others.

- **Hypocritical peers.** These peers behave properly for a period of time in order to build up a strongly positive reputation, then begin defecting. They could be random or structured. Random peers act maliciously with some probability say 20%. Structured peers follow a fixed pattern, say provide one malicious service after four good ones.

- **Colluding Peers.** Malicious peers who co-operate with each other and increase each others rating while decreasing the ratings of other peers.

To model malicious peer behavior each peer builds a knowledge base represented by three list structures viz: Doubt List, Black List and Trusted List. The doubted List contains unique user ID (UUID) and public key of peers who have suddenly given deviated performance. If the difference in trust value of some peer X as calculated by peer Y in earlier transactions is greater than some threshold $\epsilon$ then a peer is included in the doubted list. If peers in the doubted list continue giving bad performance then they are moved to Black list and the trust policy does not permit transactions with peers in this list. The Black List is purged according to an ageing policy. This strategy helps us to slowly remove peers whose misbehavior is dependent on some pattern. A peer in the doubted list is given a chance to improve his performance by allowing transactions and is moved to Trusted list if the trust value crosses the desired threshold. The control peers have a view of these lists and can broadcast the information about malicious peers after consensus.
Chapter 5

Modeling F/OSS

To model our P2P collaborative group we chose the self organizing group F/OSS[45]. We simulated a typical F/OSS application with core developers, reviewers, documenter, moderators etc. Active developers in such communities join and propose patches to existing code. Projects are evaluated by moderators on the basis of technical merit and elegance of contribution. Such systems and communities co-evolve with the contribution of a large number of participants [44]. For simplicity we mapped core developers of code to Level1, reviewers to Level 2 and moderators to Level 3. Peers join the group either as Developers, Reviewers or Moderators and assume roles of Member peers, Admission Peers or Control peers depending on their functional capability. A peer in Developer Level for e.g can be updated to reviewer based on his performance and reputation earned in the group. At level 1 member peers develop code, while control peers do additional administrative tasks. The group permits peers to collaboratively work on different projects, and efficiency of the group is based on completion time, quality and price of the different projects.

5.1 Peer Roles

As specified in our generic model the role assigned to a peer is a union of Group Roles and Domain Specific Roles. The Group Roles are Member Peer(MP), Admission Peer(AP) and Control Peer(CP). The domain specific roles that we have considered here are Developers, Reviewers and Moderators which we map to level 1 peers, Level 2 peers and level 3 peers respectively. Peers at every level can take on a Role which is a union of the Group Role and domain specific Role. Some examples are:

- **R1 = MP \cup** Developer. Peers in this Role are responsible for contributing new features and fixing bugs. They can contribute to the group by developing and submitting code, but have no access to view/modify any code/resources submitted by other peers.

- **R2 = AP \cup** Developer. These peers are Developers with additional functionality. They maintain certificates of peers, contribute to ratings of peers and contribute to join.

- **R3 = CP \cup** Developer. These level 1 peers, do the tasks of submitting code and additionally do the Group management tasks like updating a peer, monitoring the group and so on.
• Level 2 peers or reviewers can view/review code submitted by Level 1 users belonging to a particular sub-group which they have been assigned. They cannot modify code. The peers at this level too can take on the additional role of Admission Peer or Control Peer.

• Level 3 peers can modify code, reviews, fix bugs and take the final decision regarding code acceptance. Additionally they could take on Roles of Admission Peer and or Control Peer.

5.2 Task Allocation Policies

The group has different tasks or modules at different difficulty levels. These tasks could be totally independent, sequential or concurrent or there could be some dependency among concurrent tasks. Say a peer could be allocated job C provided he has earlier met some pre-requisites. Task scheduling is done amongst peers based on peer capability, difficulty level of tasks and completion requirements. Consider a scenario where there are m peers with available resources needed by x peers for job completion. Each time K resources are generated by the system and each resource node can process only one job per time. If k ≥ m then a job allocation policy could be

• Allocate jobs to resources depending on their trust values/roles without considering the complexity of jobs.

• Allocate jobs depending on the complexity level.

• Allocate jobs depending on deadline to finish.

• Allocate jobs depending on security policy, i.e whether it needs encryption or not.

We assume a Poisson distribution of task arrival. Tasks are modeled here as software modules of different difficulty levels easy, medium and difficult. Completion of modules and arrival of the same for review or moderation is also as per Poisson Distribution. The task allocation policy decides the task to be allocated to a peer depending on the complexity level of the task and the trust value of the peer. The Job allocation to peers is varied as shown in table 1.

<table>
<thead>
<tr>
<th></th>
<th>Easy</th>
<th>Medium</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alloc1</td>
<td>0 - 40</td>
<td>40 - 70</td>
<td>70 - 100</td>
</tr>
<tr>
<td>Alloc2</td>
<td>0 - 55</td>
<td>55 - 75</td>
<td>75 - 100</td>
</tr>
<tr>
<td>Alloc3</td>
<td>0 - 70</td>
<td>70 - 85</td>
<td>85 - 100</td>
</tr>
</tbody>
</table>

Table 5.1: Variation of job allocation

5.3 Trust parameters and attributes

The attribute set for Direct Trust for F/OSS domain consists of parameters like...

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

5.4 Peer Behavior

A peer when he joins the group has an initial self proclaimed rating SPR, which is his initial trust value. As the peer performs in the group, this trust value changes based on the quality of service he provides and feedback given by other peers. A peer can have some maximum intrinsic potential MIP beyond which his trust value cannot rise.

Multiple policy optimization would focus on optimising the behavior of a peer so that it reaches its MIP.

Groups in which every good peer reaches its MIP fast would survive.

As a rater, peer quality can be Good, Bad or Honest. A Good(Bad) peer would send absolutely correct(wrong) ratings each time based on truth. This is actually very difficult without a global view of the system, but we model it for studying extreme case of peer quality. Honest raters provide best-effort truth based on their own views.

As a service provider Peers can be Good, Average, O.K or Dynamic. We model good peers as expert peers who always provide excellent service. In the case of Average peers and OK peers the service quality gradually deteriorates. Dynamic peers are those who provide the services with one of the three qualities uniformly i.e 33% of its actual services are good.

For simplicity we assume that the Member peers behave as OK peers, Admission peers behave as average peers and Control peers behave as expert peers while providing services as well as while rating other peers. Further we assume that OK peers develop code with a certain higher percentage of bugs, at a slower rate as compared to Average and Expert peers. Similarly while rating, the Expert peers give more accurate ratings as compared to OK and Average peers.

5.5 Modeling Domain Specific Policies

Some sample domain specific policies for the F/OSS domain are listed below:

- A peer is allowed to change its role from coder to reviewer if it has coded 10 modules and out of which 7 are accepted.

- A reviewer can be updated to the moderator level if the number of reviews done by that reviewer is greater than 5 and all are accepted

Some dynamic rules can be:

- If number of reviewers in the overall group becomes greater than 40% then the policy for update to reviewer is made stricter.

- As the number of difficult jobs in the system increases change the job allocation policy

- As the number of malicious peers in the system becomes greater than 10% change the job allocation policy, so that jobs are not assigned to the lower trust value peers.
Some rules for task allocation in this domain are as follows:

In policy Alloc1, if a peer with initial rating between 0-40 enters the system then an easy job is allocated. If no easy job in queue then the next complex job is allocated and so on. Thus the policy does not permit a peer to remain idle at the cost of allocating complex jobs to less trusted peers.

In Alloc2, when a peer with rating between 0-55 enters the system, he is allocated an easy job if available, else he is not allocated a job at all. Thus peers are allocated jobs strictly according to trust values.

**Group Decisions for survival and evolution**

Group decides on the type of group composition needed for efficiency i.e the % of MP, CP or AP. Based on this decision the join/leave policy of the group changes dynamically. So say if the current join policy permits peers with SPR 40, 60 and 80 to join as MP, AP and CP respectively and if after a certain duration, the number of member peers exceeds 50% of the group size, making it difficult for the existing CPs to handle the group, thereby reducing GPI, then the group policy for join could change, postponing further low level join requests.

We formulated rules in prolog to represent these policies. We are able to express dynamic rules at runtime, as well as change existing policies depending on the status of the group at runtime by exploiting the assert and retract clauses of Prolog. Thus the system switches between policies over time depending on the job composition, group composition and current trust values of peers in the group. The framework thus gives peers the flexibility to self regulate their behavior based on learning experience in the group. Sample domain specific policies to update role from Developer to Reviewer are given below:

\[
\text{update\_level}(	ext{CodesSubmitted}, \text{PercentAccept}, \text{Level}, \text{Role}, \text{Rating}) \quad :- \text{member}(\text{Role}), \text{CodesSubmitted} > 30, \text{PercentAccept} > 50, \text{Level} =:= 1, \text{Rating} > 0.6
\]

\[
\text{update\_level}(	ext{CodesSubmitted}, \text{PercentAccept}, \text{Level}, \text{Role}) \quad :- \text{admission}(\text{Role}), \text{CodesSubmitted} > 20, \text{PercentAccept} > 50, \text{Level} =:= 1.
\]

A policy to introduce a new level is

\[
\text{introduce\_level}(	ext{Role}, \text{Level}, \text{CPs}, \text{NPeers}, \text{High\_Level}, \text{Vote}) \quad :- \text{maximal}(	ext{Role}), \text{CPs} > 40, \text{NPeers} > 40, \text{Level} = \text{High\_Level}, \text{Vote} =:= 100.
\]
Chapter 6

PeerSim Simulator

PeerSim [42] is a Peer-to-Peer simulator. It has been 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. Event driven mode is more realistic because generally the system will be driven completely or partially by external events such as queries by users.

6.1 PeerSim Architecture

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

- **Protocols:** They are used to define the behavior of the different peers, 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 can 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:** As the name implies, 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.

### 6.2 Config File

A Peersim configuration file is a standard text file and can manage the following:

- Any peersim component can be defined and configured.
- Any component can be fine scheduled.
The network size, the length of the simulation (in cycles).

The Node class type can be redefined.

Simulation details (e.g., shuffling, seed,...).

Number of distinct experiments.

Every comment is prefixed with a # at the beginning of the line. There is no order in
the configuration file for the instructions. A sample config file is given below.

```
#random.seed 1234567890
network.size 40
simulation.experiments 1
simulation.endtime 10000
simulation.logtime 1
### protocols ===========================
protocol.link IdleProtocol
protocol.p JoinProtocol
protocol.p.transport tr
protocol.p.link link
protocol.urt UniformRandomTransport
protocol.urt.mindelay MINDELAY
protocol.urt.maxdelay MAXDELAY
protocol.tr UnreliableTransport
protocol.tr.transport urt
protocol.tr.drop DROP
### initialization======================
initializer Initializer
p0.protocol p
p0.groups 1
p0.SWILocation/usr/lib/pl-5.6.55/bin/i386-linux/pl
initializer WireInetTopology
p1.protocol link
p1.peerProtocol p
### control =================-------------
control.0 TrafficGenerator
control.0.protocol p
control.0.link link
control.0.step 1
```

The following lines give a brief explanation of the sample config file.

- The lines from 1 to 5 represent the global peersim configuration, such as the seed
  for the random number generator, default network size, the number of simulation
  cycles to perform, the finishing time of the simulation and the log time.
The protocol defined is the join protocol (lines 9, 10, 11)

Lines 13-19 represents in-built protocols needed for passing message from one peer to the other in event driven mode

The first initializer (lines from 22 to 25) runs at the starting of simulation and we can initialize all the peers using this. `graphs` and `SWILocation` are the parameters used in the initializer

Lines 27-29 represents WireNetTopology which is one more control type initializer which runs at the starting of simulation

Finally, a TrafficGenerator is defined from line 32 to 35. This is used to generate events from outside in the event driven mode. `Step` is the in-built parameter used to indicate the number of steps in which an event should be generated

### 6.3 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 the simulator sets up the network initializing the nodes in the network, and the protocols 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 also includes the wiring of the nodes at the start thereby specifying the connectivity of the network.
- After initialization, there should be some traffic or events generated so that event driven engine can call those events from the event queue and perform them. This can be done through control objects. The simulation ends when there are 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.
6.3.1 Protocols

The protocols that have been used are:

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

2. **JoinProtocol**  
   This user defined protocol is the main class which implements the EDPProtocol used for event driven simulation. This class contains all the functionalities of a peer used for joining, updation, leaving etc. This class extends a class named Peer which has all the variables required for a peer like UUID and some basic functions like encryption, decryption, verification etc. JoinProtocol contains a predefined method known as `processEvent` which is used to process incoming messages to a peer. The functions designed to process incoming messages are:
   - **Authentication**: This 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.
   - **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. 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 and update calls AccessControl.
   - **UpdationManager**: 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 requested by the peer. This is called by `processEvent` of that peer when requests is of type rekey.

6.3.2 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 group policies by loading a prolog engine in the network. Before starting the simulation we need some predefined peers and groups already in the network and later on new peers can join using the defined protocols.
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.

### 6.3.3 Controls

Controls are used to control the simulation in between. `TrafficGenerator` is a user-defined control which generates the traffic of messages of requests of joining, transactions or leaving and adds these requests periodically to the `EventQueue` of EDsimulator. The order of requests and number of requests can be defined in the configuration file.

The structure of the message has been defined 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 to the `EventQueue` of the ED Simulator in the form of ”requestedNode, MessageEvent”. Simulator sends the `MessageEvent` to the `requestedNode` by extracting 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 and sends it to any one peer of the group. We are maintaining a vector (leavingNodes) which contains nodes who left till now in the `TrafficGenerator` only. If someone leaves we add it in that vector so that they can join later in a new session.

- **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 allowed joining and for joining in a new session, randomly a peer from the vector ‘leavingNodes’ make a request of join for it.

- **Feedback**: Feedback is given randomly by a peer to another random peer in his group. It sends a Rating Certificate which contains a rating signed by him with his trust value.

- **UpdateRequest**: Randomly a peer from the network sends an UpdateRequest containing the requestType and his certificate which contains all the rating certificates given to him by other peers as a feedback.

### 6.3.4 Application Framework F/OSS

Above mentioned protocols, initializers and controls are for our basic framework. To make it work for any application framework we need to define our own protocol and a control is needed for creating events for interaction between peers according to the application

**FossProtocol**

The protocol that has been made is **FossProtocol**. This class implements EDProtocol used for event driven simulation. This contain functionalities like developer can send a request to the reviewer to review a module, a reviewer can send a feedback to the developer, a developer can request the moderator to accept his module after getting enough amount
of certificates. Every incoming message will be processed by `processEvent`. The functions been made for processing incoming messages or requests are:

- **ReviewCode**: Whenever peers completes a module they sent it to the reviewers to review a module. Through control (FossControl) events are created with a message of *to review* which is sent to the corresponding reviewer peers which is then processed by the reviewer and then a reviewCode function is called to review the module. This will then create one more event with a message of *feedback* which is then send to the developer peer who was involved in creation of the module.

- **RecordFeedback**: It processes the event with a msg of *feedback*. This is used to record feedback at developer side. Whenever number of feedbacks increased by a particular limit it will then create a event with a message of *toaccept* and send to one of the moderator.

- **CheckForAcceptance**: It processes the event with a msg of *toaccept*. It calculates the average rate of the module from the reviews got by that module and checks if that is sufficient for acceptance. If that is so then it declares that module as accepted. It also gives the feedback to the all reviewers who reviewed that module on the basis of the average rate calculated and the rate given by the reviewer.

### 6.3.5 Control

The control that has been made is **FossControl**. It helps in creating starting events of sending module if finished by peers, for reviewing. It is also responsible for assigning modules to idle peers if available and checks for the project completion. Simulation ends from here when all the modules have been completed.

### 6.3.6 How to Run

The command to run the simulation is:

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

The config-file that is used in our simulation is like the same that we have shown in Figure .....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 the starting of simulation. It reads that the simulator we are using is the EDSimulator from the protocol *JoinProtocol*. It schedules 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
Chapter 7

Implementation and Results

To use Prolog and Java simultaneously we needed a library which could act as a bridge between java and prolog. Interprolog, JIProlog, CiaoProlog, PrologCafe are some bridges of which Interprolog was suitable for our system as it directly loads the prolog file from java while the simulation is running.

InterProlog is an open source library for developing Java + Prolog applications. Currently it supports XSB and SWI Prolog on Windows, Linux and Mac OS. It promotes coarse-grained integration between logic and object-oriented layers, by providing the ability to bidirectional map any class data structure to a Prolog term; integration is done either through the Java Native Interface or TCP/IP sockets. It has small part dedicated to each of the prolog systems (XSB and SWI) and each Prolog system has a specific PrologEngine subclass.

Example Java Program

Following code is the sample code which initializes the prolog engine and loads the prolog file. After loading prolog file we can pass java objects to prolog and can do verification according to our policy rules and finally can get the required results.

```java
PrologEngine engine = new
SWISubprocessEngine("/usr/lib/pl-5.6.55/bin/i386-linux/pl"); File f
= new File("verify.P"); engine.consultAbsolute(f);
// or consultRelative (to the class location), or consultFromJar(to the jar location),...

Object[] var = new Object[]{4,1}; boolean b =
engine.deterministicGoal("ipObjectSpec('java.lang.Integer',Obj1,[Peer],_)," +
"ipObjectSpec('java.lang.Integer',Obj2,[Level],_), belongs(Peer,Level)",
"[Obj1, Obj2]", var);

System.out.println(b);
```

7.0.7 Overall working of framework with prolog

Figure 7.1 shows the interaction between the events and policies in our framework. External events like join, leave, and internal events like level update, or application specific events
like submitting a review, or submitting code, in case of F/OSS might trigger some other events, like change in policy. Change in policy in turn affects the way in which events like join, leave, update occur. All events are simulated in peersim simulator [42]. Simulator triggers the policy file which is prolog rules. Database contains all the information about all the peers. Prolog rules use the database and according to the changes in database, prolog rules or policy might change or be modified.

7.1 Implementation Details

We modeled the P2P groups using a java based overlay simulator namely PeerSim[42] which is a Peer-to-Peer simulator. We chose Interprolog as our bridge between Java and Prolog as it directly loads the prolog file from java. The global and domain specific policies were implemented using SWI-Prolog which we integrated with Peersim at runtime wherein java and prolog talk to each other by means of sockets.

The simulator starts with some initial peers in the group, and then events such as join, leave, update are triggered. The events have been triggered in the event mode of peersim. At the start of the simulation we define a project which requires some certain number of modules each having its own start time, difficulty level assigned and the expected time to finish that module. A module consists of parameters like Starting time, Current time, Time of completion, Bugs, and Difficulty Level. According to the availability of peers and modules, peers are assigned modules. A module can have more than one peer working on it at a time. As soon as a module is finished, those peers will be allocated some other modules to work on, based on some job allocation policy. A priority queue is maintained based on the current time of start of a module so as to maintain the concurrency between the modules. Time is calculated on the basis of Poisson distribution where the lambda is the average time needed by a peer. Simulation runs until all modules of the project are completed. During the simulation, new peers can continuously join the group at any level depending on join policy and existing peers can also apply for update of levels. Dynamic leave of peers is also simulated. As a new peer joins he is allotted to a new module or a currently running module based on the allocation policy.

7.2 Simulations and Analysis

We performed different simulations to prove the validity and capability of our framework.
Simulation 1:

In the first set of experiments we ran the simulations for an initial 40 peers and then allowed 100 peers to join. The job allocation policy here was a simple FCFS policy and we allowed 300 jobs to enter the system. We tested various join policies. In one of the policies we permitted peers to join without any constraints. Then we ran the tests by putting constraints on the compositions of different types of peers. It was observed that when the peer composition was 40-40-20 i.e 40 member peers, 40 admission peers and 20 control peers out of a total of 100 peers, then initially the peers attain a good job success rate but with increase in number of transactions and number of peers, the time taken increases considerably. On the other hand the composition of 20-60-20 gives good results as no. of peers increases.

We repeated this for different join policies by putting constraints on the number of control peers and admission peers in each policy. We increased the total peer join to 300 peers. Various runs allowed us to come to the conclusion that depending on the type of peers entering the system (which can be measured by their trust values), the group performs better with 20 percent control peers upto a certain threshold of total peers, beyond which it performs better if control peers are increased to 30 percent. We then gave runs with a dynamic join policy wherein the policy has no restriction on control peers upto a total of 100 peers in the system, then switches to a policy of 40-40-20 from 100 to 200 peers and to a policy of 40-30-30 from 200-300 peers. Results show that the dynamic join policy gives better results as expected.

![Figure 7.2: Static vs dynamic join](image)

Graphs in fig 7.2, 7.3 are some sample outputs of static versus dynamic join policies in presence of simple FCFS job allocation policy. Graph in fig 7.3 shows that if we restrict the control and admission peers to 20 percent each, so that member peers are 60 percent then with increase in no of peers the group takes more time to complete tasks whereas a group composition of 40-40-20 gives better results.

Simulation 2:

In the second set of experiments we tested the system for varying job allocation policies. We designed 3 job allocation policies viz job_alloc1, job_alloc2, and job_alloc3. In job_alloc1 when a peer enters the system if his SPR is less than 40 then an easy job is allocated to him, if his SPR lies between 40 to 60 then a moderate job is allocated and if his SPR is greater than 60 then a difficult job is allocated. Further in case a peer with SPR less
than 40 enters and there is no easy job in the queue then the next level complex job will be allocated and so on. Thus a peer is not kept idle at any time and all jobs are allocated.

In policy job_alloc2 again the jobs are allocated as per the values in table. However here if a peer with SPR less than 40 enters the system and there is no easy job in queue then that peer is kept idle and the next peer is fetched. When all peers are fetched, the system checks for idle peers to allocate jobs. Previously allocated peers whose trust value might have increased can now be allocated the difficult jobs. Thus job allocation in this policy is done exactly according to the access rights of a peer.

Policy job_alloc3 is a combination of the above two policies. A peer with less SPR is not allocated a difficult job but a peer with higher rating may be allocated a moderate or easy job if a difficult job is not in queue.

A sample prolog rule for above policies is

\[
\text{job_req(Request,Difflevel,Rate,SPR)} :\text{joballoc(Request)},\text{Difflevel}::=1, \text{SPR} < 40, \text{Rate} < 40
\]

Here we started with 100 peers in the system and allowed a maximum of 300 peers. We defined a project which contains 1000 modules defined as jobs of different complexities, namely easy, moderate and difficult. The group chooses a join policy which ensures that there are 30% ok peers, 40% average peers and 30% expert peers. We then gave runs for different job compositions and varying job allocation policies. It was observed that when maximum difficult jobs entered the system (job composition 10-10-80) then job_alloc1 gave better results since difficult jobs were distributed amongst all peers, whereas when percentage of easy jobs was dominant (job composition 80-10-10) then job_alloc3 gave better results as seen in graphs if fig 7.4 and 7.5. Thus we concluded that with varying job compositions different job allocation policies allow the group to give optimum performance. We then varied the job allocation policy dynamically allowing the control peers to switch between policies depending on the job composition. Graph in fig 7.6 shows that this dynamism in job allocation allowed the group to perform better.

Graph in fig 7.6 is a comparison of all the static and dynamic job allocation policies with simple FCFS policy, which takes maximum time.

**Simulation 3:**

In the third set of experiments we introduced malicious peers into the group. We modeled malicious peers as peers who do not give satisfactory behavior. As coders they develop
code with larger number of bugs and give incorrect ratings as reviews and moderators. We gave runs for 5%, 10%, 15%, and 30% malicious peers for varying job allocation policies and varying compositions of peers and jobs. Results show that as the percentage of malicious peers is increased the completion time also increases since malicious peers are detected and their trust value is decreased and they are not allocated jobs in our framework. Graph in fig 7.7 shows that in presence of 30% malicious peers and with a higher % of difficult
jobs entering the system (10-80-10) job alloc1 gives best results. However in this case switching between policies dynamically does not give the best results after a substantial increase in number of malicious peers as can be seen from graph in fig 7.7.

![Graph](image1)

Figure 7.7: 30 percent malicious peers with varying job allocations

We then gave runs for an average job composition for both 10% and 30% malicious peers and compared the static job allocation policy with the dynamic policy. Results show in case of static policy the completion time is much larger as jobs are allocated to under performing peers, whereas in case of dynamic policy jobs are allocated based on appropriate trust values of peers.

![Graph](image2)

Figure 7.8: 10 percent malicious peers static vs dynamic

In both the cases better results were obtained by switching dynamically between the job allocation policies as seen in graphs in fig 7.8 and 7.9. Further for 30 percent malicious peers all the jobs could not be allocated using the static policy.

Simulation 4

In the fourth set of experiments we compared our work with that of TM[40] which is a trust model for P2P ecommerce based on Direct and Indirect trust. In the TM model attributes for peers are not considered, nor are peers segregated into a doubted list and black list. We performed a comparative analysis on a network of 100 nodes with 800 transactions in each simulation. The parameters used for comparison were

1. Average trust value of good peers
2. Average trust value of malicious peers

3. Transaction success ratio in presence of varying percentage of malicious peers

4. No of times the malicious peers are selected for transacting with increasing percentage of malicious peers.

We took runs for 5, 10, 15, 30 and 40 percent malicious peers in presence of static policy and dynamic policy. Results (see figure 7.10) show that in our model the trust value of good peers remains constant with increase in % of malicious peers and is better as compared to the TM model. Trust value of malicious peers degrades gradually in our model with increase in % of malicious peers whereas in case of TM model the trust value of malicious peers does not change much with increase in % of malicious peers.

Graph in Fig. 7.11 shows the trust value of malicious peers decreases considerably with our dynamic task allocation policy whereas for the P2P model[40] the malicious peers are able to maintain their trust value. Thus in presence of our dynamic trust policy, gradually with increase in interactions the group is able to isolate the malicious peers as their trust value falls beneath a certain threshold and they would not be allocated any further tasks.
7.3 Conclusion

We have proposed a framework for collaborative groups which has a dynamic policy driven trust based access control model. The framework is flexible and peers are able to self regulate their behavior in presence of dynamic policies. The trust metric is a tunable metric with context specific weighted attributes.

Our experiments prove that we have a flexible framework for modeling dynamic collaborative groups. The framework permits us to evaluate and measure different group metrics like job success, average trust value of peers, effect of varying join policies, varying job-allocation policies, effect of varying percentage of malicious peers and so on. Different applications can be modeled and given as inputs to our simulator. The framework can be used to test the evolution of different collaborative and self-organizing groups in the presence of varying dynamic policies. Use of prolog to model the policies permits dynamic switching between policies which results in optimum group performance in most cases. Admission control is achieved through prolog policies as the group can prioritize join requests and postpone them if required. Our adaptive trust model integrated with our dynamic policy model helps us to mitigate the effect of malicious peers and they are gradually ejected from the group.

Collaborative applications like F/OSS, multi-player online gaming and others can use this framework to test the evolution of peer groups and decide which policies to use to optimise group performance.
Bibliography


[16] Wikipedia, the free encyclopedia


46


[34] 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


[42] PeerSim Simulator Documentation From http://peersim.sourceforge.net/


[44] F/OSS Documentation: http://foss.in
