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

Fifth 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

August 23, 2010
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

Users of traditional P2P applications such as content sharing, IP telephony, video audio conferencing, wikipedia and others are increasingly getting organized into collaborative peer groups. Such groups generally have constantly changing application requirements and varied security requirements, and require a secure and reliable group communication system to ensure that they evolve and adapt according to interactions within the group. Being dynamic and self-organizing in nature in nature, collaborative groups have special security requirements not addressed by typical access control mechanisms. We proposed 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. 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.
Summary of work done during last year and Thesis Status

0.1 Problem Definition

We have modeled a framework for collaborative peer groups which supports dynamic join, leave, update of peers. Admission control and dynamic policy driven access control based on evolving trust have been achieved. We are able to study the performance of such dynamic peer groups with different compositions in the presence of dynamic policies. Our flexible framework, allows peers to self regulate their behavior.

0.2 Work Done till Jan 2009

We proposed an integrated framework for dynamic peer groups with authentication, admission control and access control. This was published in WISA 2007. We then proposed and implemented a dynamic access control model for this framework. We simulated the framework and the dynamic policies using simulator Peersim integrated with Prolog. We simulated the F/OSS application and got some preliminary results which got published in IEEE proceedings of ICC 2009.

0.3 Work Done from Jan 2009 upto August 2010

We improved upon the framework to build a generic model to suit different applications. This is detailed in Chapter 3 of the report. We proposed a state model for the same. We proposed a trust model for the framework which supports multiple attributes and modeled the above for F/OSS application. Details are in Chapter 4. We did several experiments modeling dynamic policies for join, update and task allocation. We got several graphs to show that our dynamic policies help the group to perform more efficiently. A paper on the same was published in Springer proceedings of CNSA 2010. We did preliminary modeling of malicious peers and got a few results.

0.4 Current work

We are currently working on the modeling of malicious peers in our framework, and on building a Data Architecture for our model. We hope to be able to show the diversity of our model to suit another application.
# Contents

0.1 Problem Definition ................................................. i
0.2 Work Done till Jan 2009 ........................................... i
0.3 Work Done from Jan 2009 upto August 2010 ......................... i
0.4 Current work ....................................................... i

## 1 Introduction

1.0.1 Motivating Applications ....................................... 1

## 2 Collaborative peer groups

2.0.2 Security in groups ............................................. 3
2.0.3 Flat model groups vs hierarchial groups ....................... 3
2.0.4 Requirements of P2P groups .................................. 4

## 3 Generic Application Framework for Collaborative Peer Groups

3.1 Generic Model .................................................... 5

3.1.1 Group Charter .................................................. 6
3.1.2 Group Communication Semantics ................................ 6
3.1.3 Group Activities ............................................... 6
3.1.4 Group Roles and System roles .................................. 8

3.2 Group Operations and Maintenance .................................. 9

3.2.1 Creation of Group .............................................. 9
3.2.2 New Peer Join .................................................. 9
3.2.3 Peer evict ..................................................... 10
3.2.4 Member Freeze ................................................ 10
3.2.5 Group Removal ................................................ 10

3.3 Group Goals ........................................................ 10

3.3.1 Metrics for Group Efficiency .................................. 10

3.4 Detailed working of the framework .................................. 10

3.4.1 Group Discovery and Advertisement .............................. 11
3.4.2 Admission Request ............................................. 11
3.4.3 Authentication ................................................ 12
3.4.4 Access Control ................................................ 12
3.4.5 Key Management ................................................. 12

3.5 Threat Model in P2P groups ....................................... 13

## 4 Trust Model

4.0.1 Requirement for Trust Dynamics ................................ 15
4.0.2 Decentralized Trust Management ................................ 16

4.1 Layered reputation scheme .......................................... 17
### 5 Dynamic Policy Driven trust based Access Control DPDTBAC

5.1 Hybrid Access Control Model
- 5.1.1 Dynamic Policy Driven Trust Based Access Control (DPDTBAC)
- 5.1.2 Modeling Policies using Prolog

5.2 State Model
- 5.2.1 Formal Model of Self Adaptation

5.3 Multi-policy Optimization using Collaborative Reinforcement Learning
- 5.3.1 Adaptive Reinforcement Learning for Trust
- 5.3.2 Generation of Reinforcement Value

### 6 Modeling F/OSS

- 6.0.3 Task Allocation
- 6.0.4 Peer Roles

### 7 PeerSim Simulator

- 7.1 Introduction
- 7.2 PeerSim Architecture
- 7.3 Config File
- 7.4 PeerSim Simulation Life Cycle

### 8 Implementation and Results

- 8.1 Interprolog System
  - 8.1.1 Example Java Program
  - 8.1.2 Overall working of framework with prolog
- 8.2 Application framework
  - 8.2.1 Peer Behavior
- 8.3 Implementation Details
  - 8.3.1 Simulations and Graphs
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Policy based framework</td>
<td>25</td>
</tr>
<tr>
<td>5.2</td>
<td>PDTBAC State Model</td>
<td>26</td>
</tr>
<tr>
<td>7.1</td>
<td>General Functioning</td>
<td>33</td>
</tr>
<tr>
<td>7.2</td>
<td>Cycle-based</td>
<td>33</td>
</tr>
<tr>
<td>8.1</td>
<td>Overall Framework</td>
<td>37</td>
</tr>
<tr>
<td>8.2</td>
<td>Mapping of Roles and Levels</td>
<td>38</td>
</tr>
<tr>
<td>8.3</td>
<td>Average Rating for oversmart vs honest peers</td>
<td>39</td>
</tr>
<tr>
<td>8.4</td>
<td>Job Success with static and dynamic policy</td>
<td>39</td>
</tr>
<tr>
<td>8.5</td>
<td>Job Success with varying Job Allocation</td>
<td>40</td>
</tr>
<tr>
<td>8.6</td>
<td>Job Success with varying Job Compositions</td>
<td>41</td>
</tr>
<tr>
<td>8.7</td>
<td>Job Success with varying Job Compositions and SPR</td>
<td>41</td>
</tr>
<tr>
<td>8.8</td>
<td>Job Success with varying Job Compositions and varying join</td>
<td>42</td>
</tr>
<tr>
<td>8.9</td>
<td>Varying job allocation policies for difficult jobs</td>
<td>42</td>
</tr>
<tr>
<td>8.10</td>
<td>Dynamic vs static job allocation policies</td>
<td>43</td>
</tr>
<tr>
<td>8.11</td>
<td>Dynamic vs static join policies</td>
<td>43</td>
</tr>
<tr>
<td>8.12</td>
<td>Job Success for static versus dynamic join</td>
<td>44</td>
</tr>
<tr>
<td>8.13</td>
<td>Comparison of 10 percent vs 30 percent malicious peers</td>
<td>44</td>
</tr>
<tr>
<td>8.14</td>
<td>Comparison of performance with and without dynamic policies in presence of malicious peers</td>
<td>45</td>
</tr>
</tbody>
</table>
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. 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 and normally assumes that all peers have equivalent capabilities. However the dynamic nature of peers and the changing topology of the network 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.

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

We propose a decentralized and integrated framework for authentication, admission control and access control in peer groups. Our model permits deployment of access control policies flexibly and dynamically. New policies 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 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.

1.0.1 Motivating Applications

One motivating example is the Free and Open Source Software (F/OSS) self-organizing P2P group F/OSS 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 [32].

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
Collaborative peer groups

Peer to peer online communities are truly distributed computing applications in which peers (members) communicate with each other directly to exchange information, distribute tasks or execute transactions. They can be implemented on top of a P2P network e.g. Gnutella or on top of a client server architecture e.g. online auction sites like eBay. Such applications are generally conceptualized as a global overlay network without any logical segregation or segmentation of resource availability.

Users of traditional P2P applications such as content sharing, IP telephony, video audio conferencing, wikipedia and others are increasingly getting organized into collaborative peer 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.

All peers are members of the main overlay which has information about the group and group membership. The communication primitives associated with such groups are Reliable message delivery to proximal group members or a subset thereof, message aggregation of peers and discovery of peers.

2.0.2 Security in groups

A secure group is a collection of members who are authorized to access a set of information or resources. Group confidentiality protects group information from being disclosed to non-members. Forward secrecy ensures that evicted members cannot access future information. Backward secrecy ensures that new members cannot access past information. Group integrity deals with protection of group information from modification by non-group members.

2.0.3 Flat model groups vs hierarchial groups

A flat group has no group structure or relationships between group members. They are generally used for individual collaboration or information distribution such as web broadcasts of pay-per-view. Certain members may advertise a security policy designed to meet certain goals that other members can also use.

Typically a member/peer of a secure group represents a person but it can represent a sensor, a PDA or a piece of software. Generally the assumption is that all peers have the
same group communication capabilities and all peers can both send and receive information.

2.0.4 Requirements of P2P groups

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 focus of P2P groups is Admission control. How to decide whether new peers may be admitted to the group or not. Another 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

Thus 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 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.
Chapter 3

Generic Application Framework for Collaborative Peer Groups

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. Communication primitives used are unicast, multicast and broadcast. 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 framework has the flexibility to change rules and make it adaptive. Peers can self regulate their behavior based on learning experience in the group.

3.1 Generic Model

P2P collaborative groups are formed by set of peers with some common interest and controlled membership. We make the following assumptions for our model:

Assumptions

- The system is asynchronous i.e there is no global clock.
- Domain specific rules and policies reside in the local memory of a peer.
- Communication primitives used are unicast, multicast and broadcast.
- Reliable message delivery is assumed.
- No peer symmetry, i.e peers do not have equivalent functionalities.
- Peers hold a "template" that defines 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.
3.1.1 Group Charter

Each group has a group Charter with information about:

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

The meta policy describes the roles permitted in the group and the rules associated with each role. A well formed group charter could have some initial rules as follows:

- Well defined rules for adding new peers and discarding malicious peers.
- Members should be able to share 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.

3.1.2 Group Communication Semantics

The central concept of group communication is that of a view, i.e. a snapshot of membership in a group. Virtual Synchrony in GCS is the property that every two processes that participate in the same two consecutive view changes, deliver the same set of messages between the two changes. Virtual Synchrony semantics VS and Extended Virtual Synchrony semantics are two well known group communication models. The VS semantics ensures that messages are delivered in the same view they were sent in. The EVS semantics, on the other hand, allows message delivery in a different view than it was sent in, as long as the message is delivered in the same view to all members. We assume VS for all our group communications.

3.1.3 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

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
- Punish actions
- 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.
3.1.4 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.

Each peer (MP, AP or CP) has its own 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.

Thus in our framework

- 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.
Domain Specific Role

These are system roles at hierarchical levels, depending on the application domain, access rights/capability and trust value. So, for example for an 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).

\[ R = DR \cup GR \] where DR = domain role and GR = group role and \n\[ PA \in P \times R \]

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 focuses 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.

3.2 Group Operations and Maintenance

3.2.1 Creation of Group

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. 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.

3.2.2 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 peer 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 view.
3.2.3 Peer eviction

A peer $p_i$ 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.

3.2.4 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.

3.2.5 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.

3.3 Group Goals

The goals of collaborative peer groups are

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

3.3.1 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

3.4 Detailed working of the framework

The group has an initial admission control policy based on a dynamic threshold i.e $t$-out-of-$n$ where $t$ (no of peers collaborating) grows or shrinks in tandem with $n$ (current group size). The admission policy could be changed collaboratively by peers and this change is enforced using voting. Peers wishing to join the group declare an initial self-proclaimed rating and request for some task allocation. Based on the join access policy he may be allowed to join at a lower level and would be allotted some role based on his self-proclaimed intrinsic capability. A group membership certificate is issued to a joining peer which contains tokens signed by the existing peers. A new peer joining the group is assigned a minimum trust level. Once in the group peers are periodically rated by other peers in the group and signed rating certificates are saved by some of the peers having higher functionality. Trust value of a peer at any point of time is calculated based on the direct interactions as well as recommendations that he has received from other peers.
as well as the trust value of the recommending peer. Rating certificates are signed by recommending peers and hence cannot be modified. Each time a new peer joins or an existing peer leaves, group rekeying is done so as to ensure backward and forward secrecy. The framework defines policies for a new peer to join, or leave and for an existing peer to update its level. The access policy would decide the role a peer assumes, based on the request of the peer and his credentials as well as rating calculated by the other peers. A peer may have some maximum potential or skill set and could rise only upto that potential. Each peer maintains information related to

- its own private-public key pair
- Signatures from other peers
- List of peers trusted by him

Group access is done in stages. Group Discovery, Group registration, Authentication, Access Control and Key management.

3.4.1 Group Discovery and Advertisement

A peer willing to serve as Control peer can create a new group and define an Initial Group Charter. The peer can then advertise and publish his group along with his charter which contains documents that specify the parameters of the group, such as group type, admission policies, group name, group members etc. Subsequent peers can then find this group and join it. The group advertisement should be periodically broadcast by the peers of the group.

3.4.2 Admission Request

The process by which a peer applies to be accepted into a group. During this the peer may receive credentials(keys, passwords, tokens) that it may use later to prove group membership. The registration process requires knowledge of existence of the group. 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. Given that each peer has his own certificate which could be self signed or signed by a CA, a peer credential is 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.

\[ P_{new} \rightarrow P_i : \{JoinREQ\}_{SK_{new}}, Cert_{new} \]

where

\[ Cert_{new} = UUID_{new}, PK_{new}, RC_{new}, \{H[UUID||PK_{new}]\}_{SK_{new}} \]

The Join req consists of a tuple GR, DR. For a new peer his RC_{new} field will contain a single entry which is his self proclaimed rating SPR.
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.

3.4.3 Authentication

If the peer receiving this request is a member peer, he would forward it to an Admission peer who 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 $P_K$ 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.

We assume that each joining peer stores a local copy of the policy database. When a CP or an AP receives a join-request from a new peer it checks the policy file for the join policy. If all parameters match then the later then gives a signed membership token to the new peer.

\[ P_i \rightarrow P_{new} : \{GC_{new}\}_{G_{key}} \]

Since this is signed with the current group key which is unknown to the peer the later cannot modify it.

3.4.4 Access Control

We define a dynamic policy driven trust based hybrid access control model for the peers in our framework. The initial group policies are defined in a Group Charter, normally by the creator or the group. As peers join, the policies can be dynamically modified by control peers in the group. Thus the peers are able to self regulate and self organize, based on trust and composition. Trust is also dynamic and adaptive. Details of this model are given in Chapter 4.

3.4.5 Key Management

After the peer receives the signed Group Membership certificate it is the task of the CP to perform the group rekeying at the necessary level. The new peer is allowed to join and his public key and unique user ID (i.e hash of his user ID and public key) is broadcast to the group. CP invokes the rekeying algorithm (TGDH) and the new group key computed is sent to this peer encrypted with his public key, and to all other peers encrypted with
the old group key. Every member keeps a list of keys and the associated set of members that share that key. The list is updated whenever a new key is generated.

The entry of the new peer is broadcast to all peers of the group. The membership token $GC_{new}$ would contain details of the access rights granted to the joining peer.

### 3.5 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 [19].
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. 15.
Chapter 4

Trust Model

Trust is defined as a peers belief in attributes such as reliability, honesty and competence of the trusted peer. Trust can be broadly categorized by the relationships between the two involved peers. A peers belief in another peers capabilities, honesty and reliability based on its own direct experiences, it measures whether a service provider can provide trustworthy services; References refer to the peers that make recommendations or share heir trust values. It measures whether a host can provide reliable recommendations. Reputation of a peer defines an expectation about its behavior, which is based on recommendations received from other peers or information about the peers past behavior within a specific context at a given time. It can be decentralized, computed independently by each peer after asking other peers for recommendations.

One of the major challenges for collaborative P2P systems is the ability to manage risks involved in interacting and collaborating with previously unknown and maybe malicious peers. Reputation based Trust Management Systems derive the trust-worthiness of a peer from past history and behavior. Reputation is an evaluation of behavior done by third parties reporting the quality of past interactions. It provides a frame of reference that makes the environment predictable. However existing systems do not provide adequate reaction to quick changes in peers’ behavior and are therefore not able to successfully cope with dynamic malicious peers.

Trust management has been emerging as an essential complementary to security mechanisms in self organizing and collaborative networks. Self-organizing networks are open, dynamic, and prone to a lot of security threats. Without the protection of security mechanisms, it is impossible to run crucial applications properly. In a trust management system, evidence about nodes, such as honest, selfish, and malicious behaviors, is collected to conduct trust evaluation (i.e., trust computation, trust combination) among nodes. Based on trust evaluation, decisions can be made in order to encourage the interactions between honest nodes, punish selfish nodes, and exclude malicious nodes. That is, evidence collection, trust evaluation, and decision making are three major components of a trust management system. Evidence can be one dimensional or multi-dimensional

4.0.1 Requirement for Trust Dynamics

Trust is the firm belief in the competence of an entity to act independently, securely and reliably within a specified context. It is conditionally transitive and can be multi-dimensional and depends on the context. Trust develops over time, based on experiences. Positive experiences can lead to increase of trust while negative experiences can lead to decrease of trust. Trust can also vary over time. Thus dynamics of trust
relates to evolution of trust over time and in the face of new experiences. Reputation is an expectation about an individual’s behavior based on information about or observations of its past behavior [17]. Reputation-based trust management allows each peer to manage its own trust, hence there is no global centralized map of trust relationships, and it isn’t convenient to manage.

Main requirements of a dynamic trust metric as summarized by [?] are

- Trust metric must be sensitive to new experiences.
- Positive and negative changes of trust are not symmetric
- Sensitivity to new experiences should not depend upon total number of experiences
- While trust must be sensitive to new experiences the long term behavior must be taken into account.
- Trust metric should detect and penalize both sudden misbehavior and possibly long term oscillatory behavior.

Thus what is needed is a flexible trust metric.

Based upon the approach adopted to establish and evaluate trust relationships between peers, trust management has been classified into three categories:

- **Credential and policy-based trust management** where peers use credential verification to establish a trust relationship with other peers. e.g Policy maker, Keynote.
- **Reputation-based trust management** where peers establish trust relationship with other peers and assign trust values to those relationships. Trust value assigned to a trust relationship is typically a function of the combination of the peer’s global reputation and the evaluating peer’s perception of that peer. e.g Eigen[9] trust model
- **Social network-based trust management** where social relationships between peers are taken into account when computing trust relationships. e.g Regret, NodeRanking.

### 4.0.2 Decentralized Trust Management

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

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

Different models have been proposed. **Complaint based model** Alberer and Despotovic were the first to propose a reputation based trust management system for a decentralized peer-to-peer infrastructure [16].
This model relies on negative feedback or complaints to convey reputation information. Peers do not store information about successful interactions, only negative experiences are stored. Complaints are then forwarded to other peers. When a peer wants to evaluate the trustworthiness of a target peer, it first searches its own history to locate complaints registered by itself. It then queries other peers. However there is no measure to prevent peers from inserting arbitrary complaints about other peers. This scheme is adopted by P-Grid data structure.

**Voting based Reputation Model** XREP [?] is a distributed protocol that allows these reputation values to be maintained and shared among the servants. It consists of the phases of resource searching, resource selection and vote polling, vote evaluation, best servant check, and resource downloading.

**eBay Reputation Management System** This does not include a trust computation mechanism [10]. It presents the reputation information about a user and leaves trust determination and decision making to its users.

**Credence** uses a voter correlation scheme that weighs peers opinions[14]. NICE platforms includes trust algorithms that enable good peers to form robust co-operative groups with other good peers[13]. **Marti and Garcia** propose a limited reputation sharing scheme to reduce the no. of failed interactions between peers. **EigenTrust** proposed by Kamvar et al. is a reputation system that computes local trust values which are distributed in order to help reduce the number of inauthentic files in the network, so that the impact of malicious peers is minimized. It aggregates the local trust assessments of all peers in the network in an efficient and distributed manner, but introduces significant computational overhead. **PeerTrust** is a reputation based trust model for e-commerce communities[12]. It focuses on incorporating the credibility of recommenders in the evaluation of trust. It also includes the number of transactions and transaction community context to make trust evaluation more accurate.

Most research on reputation mechanisms for peer-to-peer systems use reputation values as a selection criterion and not as a reward scheme. Universal participation and cooperation should be encouraged by rewarding highly reputable peers by providing them with better quality of service, while at the same time taking care about computational overhead.

### 4.1 Layered reputation scheme

We propose a layered model with a hierarchy of privileges, so peers are encouraged to exhibit good behavior and reach the upper layers of the community, which provides them with a better environment. Moving up the layers requires improving reputation score, which is done mainly by providing content of good quality. However this would be application specific. A peer could be moved down the reputation layers, resulting from a decrease in his reputation score computed by the group. In this incentive based reputation system different context categories can be incorporated into the basic trust metric with a weight function. For example for an e-commerce transaction, peers can submit feedback about each other with different parameters such as quality of product, delivery time, payment reliability, etc. The transaction cost could also be taken into account. The final trust would then be a weighted aggregation of feedback along these different parameters multiplied by the transaction cost for high-value transactions. Similarly for a software development community peers would submit feedback along dimensions such as quality of
software, Lines of Code, completion time and reliability.
We propose a context based trust model that determines the weight function for such different parameters and integrates the different extensions into our trust metric.

4.1.1 Context sensitive Layered Reputation model

We assume some basic principles in our Trust Model.

- A peer’s trustworthiness value depends on good behavior and enhances slowly, but drops rapidly if it exhibits bad behavior.
- If the trust value of a peer falls below a certain threshold value then he is either removed from the group or his level is demoted.
- Calculation of trust value includes a peer’s direct experiences as well as recommendations from other peers

Basic Trust Metrics

The Basic trust metrics in our model are:

1. **Feedback Rating** Reputation based systems rely upon feedback to evaluate a peer’s trustworthiness. For e.g in eBay, buyers and sellers rate each other after every transaction. We include some attributes in this feedback and associate weights with these attributes. So for e.g in a F/OSS application, the attributes associated are various aspects of the S/W quality such as Lines of Code LOC, No of Bugs, Price, Delivery Time, etc. Let \( a_i = a_1, a_2, a_3, \ldots a_n \) be the set of attributes. Let \( f_E(x, y) \) denote an evaluation given by peer x for peer y at a transaction. Then

\[
f_E(x, y) = (f_{a_1}(x, y), f_{a_2}(x, y), f_{a_3}(x, y), \ldots f_n(x, y))
\]

where \( f_{a_i}(x, y) \epsilon [0,1] \) is the feedback score given by peer x about peer y for attribute \( a_i \). Relative importance assigned to each attribute can be modelled as weight \( w_{a_i} \) such that \( \sum w_{a_i} = 1 \). Peer y evaluates peer x against attributes \( a_i \) and the rating thus computed is multiplied by \( w_{a_i} \) which is the weight associated with attribute \( a_i \).

2. **Reputation** This is a metric measuring overall quality of previous transactions between peer x and y. Assume that peer x stores upto n feedback ratings of previous transactions with peer y. Then reputation of peer y is calculated by peer x as

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

where \( f_{E_i}(x, y) \) 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. The decay factor is important because trustworthiness is not always the same and may change with time.

3. **Indirect Reputation** 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...
\[ k = (1, 2, ...n) \] 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 \( y \).

\[
T_{ID}(x, y) = \frac{\sum_{i=1}^{n} I(x, y).R(k, y).T(k)}{\sum_{i=1}^{n} I(x, y)}
\]

(4.2)

4. **Context Factor** To add incentives for rating we add the factor

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

where \( F_x \) is the total number of feedback ratings given by peer \( x \).

5. **Credibility of evaluating peer** The evaluator’s credibility is important for a peer to decide whether to accept the reputation value or not. If an evaluator has low credibility, his evaluation will be ignored. Thus the effect of malicious peers can be avoided to some extent.

### 4.1.2 Design of Trust Model

We calculate a peer’s trustworthiness from the direct and indirect reputations and determine the global reputation.

**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.

\[
T(D) = \frac{\sum I(x, y).R(x, y).T(y)}{\sum I(x, y).T(y)}
\]

(4.3)

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 \( y \), \( T(y) \) denotes the trust value of peer \( y \).

**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.

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

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.
Chapter 5

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. In the context of dynamic self-organizing collaborative groups we define policy as:

Policy(P):- Set of rules/requirements associated with a group/peer/service/resource or domain. Thus it can be represented as a set 

\[ P = (A_P, TP, PP, MP, AP, SP, OP) \]

- \( A_P \) → authentication policy
- \( TP \) → Trust policy
- \( PP \) → Privacy policy
- \( AP \) → Admission policy
- \( MP \) → Management policy
- \( SP \) → Security policy
- \( OP \) → Other policy

Trust is not a fixed value and can change dynamically depending on the behavior of the peer and the context in the environment. In chapter 3 we give the details of our trust policy. We further define a set of Global Meta Policies and a set of Context specific policies. The Management policies are part of global policies and specify policies for join, leave, update etc. 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 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.
5.1 Hybrid Access Control Model

Our hybrid access control model is based on an integration of collaborative roles and trust. The trust parameters in our model build upon the PeerTrust 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.

5.1.1 Dynamic Policy Driven Trust Based Access Control (DPDTBAC)

Our model focuses on the genetic evolution of groups based on group composition. In DPDTBAC it is possible 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 the 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.

5.1.2 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. 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

\[ Ro(uo) :- R1(u1), \ldots, Rn(un) \]

where Ri are predicates and ui are (possibly empty) tuples of variables and constants. The head of the rule is \( Ro(uo) \) 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.

We define here two sample policies for a software ecosystem for controlling access to review scores and model them using Prolog.

1. During submission phase a developer may submit a code c for project p
2. During review phase, reviewer r may submit a review for code c if r is assigned to review codes under project p.
3. During assessment phase, reviewer r can read scores for code c if r has submitted review for c.
4. Authors may never read scores.

The second policy varies in rules 2 and 3
- During review phase, reviewer r may submit review for a code c if r does not have any conflict of interest with c.
During assessment phase, reviewer r can read scores for code c if r has submitted review for c and r is not conflicted with c.

If the application allows conflict of interest to change after code assignment, the semantic change is considerable, because if a reviewer who is initially assigned a code and submits a review, but later the moderator learns that the reviewer was conflicted with the code, then by first policy, the reviewer can read the scores of the conflicted paper.

The above policies are modeled in prolog as follows:

\[
\text{Allow}(d, \text{submit\_code}, c) : - d(\text{developer}), c(\text{code}), \text{phase}(\text{submission})
\]

\[
\text{Allow}(r, \text{review\_code}, c) : - r(\text{reviewer}), c(\text{code}), \text{assigned}(r, c)
\]

\[
\text{Allow}(r, \text{read\_scores}, c) : - r(\text{reviewer}), c(\text{code}), \text{has\_reviewed}(r, c), \text{phase}(\text{assessment})
\]

\[
\text{Deny}(d, \text{read\_scores}, c) : - d(\text{developer}), c(\text{code}).
\]

This policy governs the use of actions submit\_code, submit\_review, and read\_scores based on information from the environment. Predicates are Allow, Deny, has\_reviewed, assigned and so on. The environment in this case is the software ecosystem, credentials of end-users and run-time factors like current time, passage of time etc. Transitions in the policy’s environment could be triggered by various conditions.

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

**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.

**Modeling Rules for Join**

An example of rules for peer with unique identity to join as a member peer in a group at level 1 of a groups is

\[
\text{join}(\text{join}). \text{update}(\text{update}).
\]

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

verify(Npeer, Request, Rl, Level, Rate, SPR) :- join(Request),
member(Rl), Level =:= 2, Rate >= 3, assert(belongs(Npeer, Level)).

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 level.

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)).

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.

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.

update_engine(Levels, MPs, APs, CPs) :- CPs > 20,
retract((verify(Npeer, Request, Rl, Level, Rate, SPR) :- update(Request), maximal(Rl), Level =:= 3, Rate >= 7)),
assert((verify(Npeer, Request, Rl, Level, Rate, SPR) :- update(Request), maximal(Rl), Level =:= 3, Rate >= 9)).
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.

5.2 State Model

5.2.1 Formal Model of Self Adaptation

We introduce a formal model of self organization in distributed groups as the adaptation of the group to improve desired system properties in the current environment.
**System State**

The set of initial states of the peers’ comprises the system state. At time $t$ it is denoted as

$$S_t = (S_i^t)_{i \in N_t, e \in S}$$

$N_t$ is the set of peers at time $t$. $S_i^t$ are the initial states of each peer. $S$ is the set if all possible system states. If $e_0, e_1, e_2, ..., e_n$ represent environment state then system behavior is modelled as a sequence of pairs of system state and total environment state. A system state transition is caused by transitions of individual states resulting in change of policy triggered by events like receiving feedback from peers, and the local environment. $(s_1, e_1) \rightarrow (s_2, e_2) \rightarrow (s_3, e_3) \ldots \ldots$

We introduce an abstract function $behavior$ which measures the quality of some desired system property for a particular state of the system and the system’s environment.

$$behavior : SX \xi \rightarrow R$$

The $behavior$ function measures how well the system state is matched to the environment. Self organizing systems should be engineered to adapt towards states that maximize system behavior. We define $S^*(e)$ as the set of optimized states.

$$S^*(e) = \{ s e S : behavior(s, e) = maxbehavior(s', e) \}.$$ 

So at time $t$, set of optimal system states is

$$S_t^* = S^*(et).$$

We apply this system model to our application of self-organizing groups F/OSS.

At a specific instant the system is in state $S_i$ and environment $e_i$ where policy $P_i$ applies and group events like Join, Leave, Update, Review, Submit, Moderate and so on keep occurring. These group events have been labeled in the figure 5.2 as $E_1, E_2, E_3$. The inputs to the system are good peers, average peers and malicious peers. Control Peers, Admission Peers and Member peers have been labeled as CP, AP and MP respectively.

![PDTBAC State Model](image)

**Figure 5.2**: PDTBAC State Model

26
Some control peers periodically monitor the Group Performance Index (GPI) which could be measured by factors like the current trust value of peers in the group, percentage of member peers and control peers in comparison with total group size and permissible joins as low-level peers within a stipulated time frame. IF GPI falls below a certain threshold, a control peer measuring group behavior could trigger an internal event like a request for policy change which would be done in consensus with other control peers at that level and group would now go to state $S_j$ with policy $P_j$. Group events $E_1,...,E_n$ would continue to occur in this new state with variations decided by the new policy. Thus state changes are triggered internally due to the effect of external events on the group.

### 5.3 Multi-policy Optimization using Collaborative Re-inforcement Learning

Self-organizing algorithms are suitable for engineering large scale decentralized systems as they enable system to deal with the lack of global knowledge and central control. These algorithms could find a wider application in engineering decentralized and ubiquitous systems once they are also capable of dealing with optimizing the system’s performance towards multiple policies. All agents in a collaborative multiagent system can potentially influence each other. It is therefore important to ensure that the actions selected by the individual agents result in optimal decisions for the group as a whole.

In reinforcement learning (RL), an autonomous agent associates actions with system states, in a trial-and-error manner, and the outcome of an action is observed as a reinforcement that, in turn, causes an update to the agents optimal policy using a reinforcement learning strategy.

In Collaborative Reinforcement Learning (CRL), agents perform actions that affect the environment and receive variable rewards for actions they perform. Agents learn the best actions to perform for given environmental conditions and entity characteristics, based on the rewards received. Changes in an agents environment provide feedback into the agents state transition model and connection cost model, while changes in an agents optimal policy provides collaborative feedback to the cached values of its neighboring agents using advertisement. As a result of the different feedback models in CRL, agents can utilize more information when learning an optimal policy in a distributed system. Collaborative feedback also enables agents to learn from their neighbors to solve collective system problems. CRL has been used to optimize routing and load balancing in peer to peer networks. We would like to use CRL techniques to optimize our group behavior.

#### 5.3.1 Adaptive Reinforcement Learning for Trust

Agents, typically control peers in our framework, model their environment by modeling the outcome of their actions. They have a list of actions to perform, and on performing an action a peer/agent receives feedback from the environment (other peers) and evaluates the quality of its action using this feedback. It then tries to select best possible action based on consequences of previous action. To decide which action is better Reinforcement Learning is used. Thus the peer or agent has an opportunity to select from a variety of actions which leads to rewards or punishments. The primary goal if to maximize the reinforcement value.
5.3.2 Generation of Reinforcement Value

Incentives should be given upon submission of good jobs and on giving honest and correct ratings. Incorrect ratings/evaluations and refusal to rate should be punished. For each opinion requested the control peer/agent calculates the associated error and generates a reinforcement value as:

\[ R = I_{\text{max}} \times (t - \text{opErr}), \text{opErr} < t \]
\[ \text{maxP} \times (\text{opErr} - t), \text{otherwise} \]

where

- \( R \) = Reinforcement value
- \( t \) = threshold for separating rewards and punishments
- \( \text{opErr} \) = error associated with opinion
- \( I_{\text{max}} \) = maximum incentive
- \( \text{maxP} \) = maximum punishment
Chapter 6

Modeling F/OSS

To model our P2P collaborative group we chose the self organizing group F/OSS which represents a group of like minded participants to develop software systems and related artifacts intended to be shared freely. Hundreds of F/OSS systems are now in widespread use by thousands of end-users, and some of them (e.g. Mozilla Web browser, Eclipse) entail millions of lines of source code. Participants in this group could be core developers or code contributors, reviewers, module owners or moderators and so on. Tasks in the group could be development of software modules of different difficulty levels, content distribution, resource sharing, publish/subscribe and postings news-group. The participants communicate via online discussion forums, threaded email messages, newsgroup postings etc. F/OSS systems, hyperlinked artifacts and tools and project web sites serve as venues for socializing, building relationships and trust, sharing and learning with others. What emerges from this is a kind of multi-project ecosystem, where ongoing development and evolution of one F/OSS system gives rise to propagated effects, changes in one of more projects linked with it. Thus software evolution in a multi-project F/OSS ecosystem is a process of co-evolution of interrelated and inter-dependent projects, people, artifacts, tools, code and project specific processes.

In our model we segregate peers into different levels based on their level of contribution. For simplicity we map developers to level 1 in the group, reviewers to level 2, and moderators to level 3. In the absence of a centralized mechanism the challenges are recognizing good peers and differentiating them from bad ones, storing and accessing reputation values in a decentralized fashion, measuring the credibility of reported ratings, encouraging universal participation and co-operation.

Most research on reputation mechanisms for use reputation values as a selection criteria and not as a reward criteria. Not only should good behavior be rewarded but bad behavior should be penalized. Our model provides a hierarchy of privileges so that peers are encouraged to exhibit good behavior to reach the upper layers of the community.

6.0.3 Task Allocation

We assume a Poisson distribution of task arrival. Tasks are modeled here as software modules of different difficulty levels 1, 2, 3 etc. Peers are allocated tasks based on availability and capability of the peer as per his self proclaimed rating. Peers then get rated based on their contribution or according to the code that have submitted. Completion of modules and arrival of the same for review or moderation is also as per Poisson Distribution. Each module has an expected time of completion.
6.0.4 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. We map the Developers to level 1 peers, Reviewers to Level 2 peers and Moderators to level 3 peers. 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 ∪ 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. If a level 1 peer is assigned a particular module, it has no access to software/tools for other modules.

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

- **R3 = CP ∪ 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 could be assigned more than one sub-group depending on capability. 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.

For our simulations we made some assumptions about peer behavior.

1. Expert peers/Control peers: A control peer behaves as an expert peer. If he is a developer, probability of bugs is very less, code quality is good and submission time is less. If he is a reviewer then ratings are accurate.

2. Average peers/Admission peers: Average peers in the role of Developers are those whose performance is low as compared to the expert peers but better than the OK peers.

3. Ok peers/Member peers: Ok Coders are considered as member peers only as they are not capable of managing the group behavior. If they are at developer level, they make some mistakes but finish their work on time and if they are at reviewer level because of less experience they may give good ratings to the bad codes also.

6.1 Domain Specific Access Policy

To model F/OSS in our framework we need specify some domain level policy to perform the work according to that domain. Sample policies require information such as assignment
of projects to reviewers, assigning moderators to projects reviewed and so on. We model such interactions through events that share same names as actions in policy requests. For e.g. a transition labeled join req() would correspond to a join request sent to the policy. One sample policy for the F/OSS application could have rules like:

- Developer d in Member role can be updated to reviewer r if he has submitted at least t codes and percentage accepted is greater than some threshold.
- Reviewer r can submit review for project c if he is assigned c and there is no conflict.
- Reviewer may be updated to role of moderator if he has successfully reviewed x projects.
- A new level could be introduced if the number of maximal role peers at highest level exceed some threshold t and all agree to introduce a new level.

Some of these rules are modeled as:

update_level(CodesSubmitted, PercentAccept, Level, Role) :- member(Role), CodesSubmitted > 6, PercentAccept > 50, Level =:= 1.

update_level(CodesReviewed, PercentAccept, Level, Role) :- member(Role), CodesReviewed > 10, PercentAccept > 50, Level =:= 2.

update_level(CodesSubmitted, PercentAccept, Level, Role) :- admission(Role), CodesSubmitted > 20, PercentAccept > 50, Level =:= 1.

update_level(CodesReviewed, PercentAccept, Level, Role) :- admission(Role), CodesReviewed > 20, PercentAccept > 70, Level =:= 2.

introduce_level(Role, Level, CPs, NPeers, High_Level, Vote) :- maximal(Role), CPs > 40, NPeers > 40, Level == High_Level, Vote =:= 100.
Chapter 7

PeerSim Simulator

7.1 Introduction

PeerSim [30] 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. Its every component can be dynamically loaded through the configuration file and almost every component can be replaced with alternative implementations.

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

7.2 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.
7.3 Config File

For enabling the simulation engine to understand our protocols and controls we define a configuration file which is a simple ASCII file which defines the components that we want to use in the simulation and how they should interact with each other. Every comment is prefixed with a # at the beginning of the line. There is no order in the configuration file for the instructions.

```
1 #random.seed 1234567890
2 network.size 40
3 simulation.experiments 1
4 simulation.endtime 10000
5 simulation.logtime 1
6 #=================================== protocols ===========================
7
8 protocol.link IdleProtocol
9 protocol.p JoinProtocol
10 protocol.p.transport tr
11 protocol.p.link link
12
13 protocol.urt UniformRandomTransport
14 protocol.urt.mindelay MINDELAY
15 protocol.urt.maxdelay MAXDELAY
16
17 protocol.tr UnreliableTransport
18 protocol.tr.transport urt
19 protocol.tr.drop DROP
20 #=================================== initialization======================
21
22 init.0 Initializer
23 init.0.protocol p
24 init.0.groups 1
25 init.0.SWILocation/usr/lib/pl-5.6.55/bin/i386-linux/pl
26
27 init.1 WireInetTopology
28 init.1.protocol link
29 init.1.peerProtocol p
30
31 #=================================== control ===========================
32 control.0 TrafficGenerator
33 control.0.protocol p
34 control.0.link link
35 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 WireInetTopology 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

7.4 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.
Chapter 8

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.

8.1 Interprolog System

InterProlog is middleware for Java and Prolog, providing method/predicate calling between both, either through the Java Native Interface or sockets; the functionality is basically the same in both cases. InterProlog’s innovation to this problem is its mapping between (serialized) Java objects and their Prolog specifications. Java Serialization API does the most of the work on java side while prolog side is built upon DCG that analyses/generates serialized objects. A definite clause grammar (DCG) is a way of expressing grammatical relationships.

It has small part dedicated to each of the prolog systems (XSB and SWI) and each Prolog system has a specific PrologEngine subclass. To use this we have to first install XSB or SWI Prolog System on our system. We have used SWI [31] Prolog.

8.1.1 Example Java Program

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);
Figure 8.1: Overall Framework

System.out.println(b);

Above 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.

8.1.2 Overall working of framework with prolog

Figure 8.1 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 [30]. 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.

8.2 Application framework

We simulated a typical F/OSS application with core developers, reviewers, documenter, moderators etc. 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.

8.2.1 Peer Behavior

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.

The mapping between roles, levels and the entities of F/OSS model is shown in the figure 8.2.

8.3 Implementation Details

We modeled the P2P groups using a java based overlay simulator namely PeerSim 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. 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.

8.3.1 Simulations and Graphs

Some of the simulations that have been done are as follows:

**Simulation1:** Here we started with 40 peers in the system and allowed a maximum of 200 peers. Peers declare an initial SPR and are allocated jobs which are project development modules based on simple FCFS job allocation policy. Events such as join and leave occur dynamically as per the policies. Simulation ends when 1000 modules finish.
Figure 8.3: Average Rating for oversmart vs honest peers

**Analysis** The graph in figure 8.3 is a plot of the average trust rating of oversmart versus honest peers. It is observed that for peers who declare an SPR which is much higher than their actual rating, the group eventually finds it out and the average trust value of the peer decreases. Thus for peers who over estimate their potential, the group eventually decreases their average trust value. Under estimation of initial trust value gives better performance than over estimation. The average trust value computed is used in the global policy to eject peers who give a very low performance or provide false ratings.

**Simulation 2:** In this simulation we start with 40 peers in the system and allow a maximum of 300 peers. A total of 1000 modules are allowed to enter the system using a Poisson distribution.

Figure 8.4: Job Success with static and dynamic policy

We compare the performance of static versus dynamic policy. In the static join policy we restrict the total no of expert peers to 20% of the group size. Thus when a peer wishes to join in the role of expert peer, he is permitted to do so only if current group composition has less than 20% expert peers. In dynamic join policy we do not keep any restrictions till the group size reaches 100. Once the group size reaches 100, the join policy changes dynamically to restrict the expert peers to 20% of group size and again for group size between 200 to 300 the policy changes to permit 30% expert peers. This is done to study the effect of varying expert peers on group size.
Analysis The graph in Fig 8.4 shows that initially till 50-60% of the job completion the two plots are very similar but after that as expert peers increases the time of completion decreases. Thus dynamic policy gives better results as compared to static.

Simulation3: Here we started with 100 peers in the system and allowed a maximum of 150 peers. The project contains 1000 modules. We defined jobs of different complexities as inputs, namely easy, medium and complex. Jobs enter the system in the ratio (60,20,20) meaning that 600 jobs are of easy complexity, 200 are medium and rest 200 are complex. We assume that most of the peers are truthful and have provided their accurate SPR.

The Job allocation to peers is varied as shown in table 1. In policy Alloc1, Easy jobs are allocated to peers having SPR between 0 to 40, Medium jobs between 40 to 70 and Difficult jobs between 70 and 100.

<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>0-60</td>
<td>60-80</td>
<td>80-100</td>
</tr>
<tr>
<td>Alloc3</td>
<td>0-80</td>
<td>80-90</td>
<td>90-100</td>
</tr>
</tbody>
</table>

Table 8.1: Variation of job allocation

Analysis The graph in Fig 8.5 shows that the policy Alloc2 gave the best results as sufficient number of peers were available for easy jobs. Policy Alloc1 gave the worst result because of the less availability of peers for easy jobs. Initially till 50-60% of job completion Alloc1 gave better results because of proper distribution of peers for each job.

Simulation4: Initially we started with 100 peers in the system and then we permitted a maximum of 200 peer joins for this project.

The group choses a join policy which ensures that there are 30% ok peers, 40% average peers and 30% expert peers. The Job composition is varied as shown in table 2.

<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>0-60</td>
<td>60-80</td>
<td>80-100</td>
</tr>
<tr>
<td>Alloc3</td>
<td>0-80</td>
<td>80-90</td>
<td>90-100</td>
</tr>
</tbody>
</table>

Analysis Here we have taken composition of peer roles as (30, 40, 30), so there are equal number of OK and Expert peers in the system. The job compositions are varied in each policy. Thus in Compos 1, jobs arrive in the order of 20% easy jobs, 20% medium jobs and 60% difficult jobs. The job allocation policy is such that the difficult jobs are allocated to expert peers and easy jobs are allocated to OK peers. The graph in Fig 8.6
Table 8.2: Variation in job composition

<table>
<thead>
<tr>
<th></th>
<th>Easy</th>
<th>Medium</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compos1</td>
<td>20</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>Compos2</td>
<td>40</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Compos3</td>
<td>60</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

shows that in Compos2, the project completed very fast. This is because of the proper distribution of jobs and peers. Compos1 gives better results than Compos3 because of the large number of expert peers which decreases the overhead per expert peer. Thus job completion rate improves in a group which has a more stringent role and job allocation policy based on initial SPR of peer.

**Simulation 5 and 6:** In simulation 5 we plotted graphs for static versus dynamic job allocation policies by fixing SPR at 40 and 60 respectively whereas in simulation 6 we kept both join policies as well as job policies dynamic.

Graphs in Fig 8.7 and Fig 8.8 show that in both cases dynamic policy gives better
result than static.

Figure 8.8: Job Success with varying Job Compositions and varying join

**Simulation 7** We defined 3 job allocation policies. Policy 1 allocated the jobs as per table 8.2. A peer with SPR 70 for example, is not allocated a job if there is no complex job. In policy 2, when a peer enters first the job matching with his SPR is searched for, if not present then the next lower level job is allocated and so on. It was observed that when the percentage of difficult jobs entering the system was greater, policy 1 gave better results and when the percentage of easy jobs entering was greater, policy 3 gave better results. We then varied the job allocation policy dynamically, as per the type of jobs entering the system.

Figure 8.9: Varying job allocation policies for difficult jobs

The graphs show that the dynamic job allocation policies gave better results than static.

**Simulation 8 and 9** Here we gave varying join policies for a fixed job allocation policy and plotted the results for static versus dynamic join.
Graphs in fig 8.11 and fig 8.12 show that fully dynamic join gives best results.

**Simulation 10** We modeled malicious peers as peers who do not give satisfactory behavior. As coders they submit code with large no of bugs and as raters they always give wrong ratings. We gave different runs for 10 percent and 30 percent malicious peers. It was observed that in both cases the dynamic policy gave better results than static.
Figure 8.12: Job Success for static versus dynamic join

Figure 8.13: Comparison of 10 percent vs 30 percent malicious peers
Figure 8.14: Comparison of performance with and without dynamic policies in presence of malicious peers
Bibliography


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


[32] F/OSS Documentation: [http://foss.in](http://foss.in)
