Dynamic Policy Driven Trust Based Access Control for Collaborative Peer Groups

Madhumita Chatterjee, G.Sivakumar, and Menezes Bernard
Indian Institute of Technology, Mumbai, India
madhumita@cse.iitb.ac.in, siva@cse.iitb.ac.in, bernard@cse.iitb.ac.in

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

To address above mentioned issues we propose an integrated framework for dynamic collaborative groups with authentication, admission control and access control. The framework supports a dynamic policy driven trust based access control model. The framework has the flexibility to dynamically change access control policies based on the behavior and trust level of peers in the group and the current group composition. Peers can self regulate their behavior based on an adaptive trust mechanism. Our trust metric is a tunable metric which supports multiple attributes. Our model provides incentives to good peers while malicious peers are gradually isolated and eventually ejected from the group. Functionality of members in a group is also dynamic and the group can dynamically prioritize requests for join.

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

1 Introduction

Web based collaborative groups are becoming increasingly popular as peers with common interests form a network among themselves and automatically tend to create interest groups among each other, called communities. Some examples of intrinsic communities being formed, are Yahoo Groups [12] and Google Groups [13]. Applications like F/OSS, file-sharing, online gaming, video/audio conferencing, collaborative work-space, virtual meetings, distance learning environments, discussion forums and board rooms are examples of applications that are organized as peer groups [15] [16] [17]. Such communities or groups need a communication model in which in which all the peers must collaborate in order to provide the basic services such as content or messages. The group is governed by a set of rules that describe the conditions required to be part of the group.

Security in such dynamic collaborative groups is governed by membership control, authentication, access control and key management. Membership control and authentication is required to allow only authorized members to join the group. In the dynamic scenario where peers are constantly leaving and joining this becomes a difficult task. Another major challenge is access control since decisions need to be made based on collaborations from all peers and feedback mechanisms based on trust. A number of authorization systems have been developed to provide access control to shared resources in distributed environments e.g PRIMA [20], CAS [35] and Akenti [21]. These systems do not however address purely decentralized collaborations. Secure
Spread [23], Antigone project [22], and Secure group layer SGL [19] are examples of secure group communication systems. However they are not fully distributed and do not have a decentralized and dynamic access control mechanism. Thus they are not typically suited for peer to peer networks.

1.1 Challenges and issues of P2P groups

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

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

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

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

A second approach is to let group members handle the admission procedures themselves. So any subset of at least t members can jointly decide to admit a new member, via a voting protocol which is based on (t,n) threshold signature scheme where n\(\geq t\) is the current group size.. In [4] a distributed CA model is used where the CA private key is distributed among a specific subset of peers without complete replication. Each peer knows only part of the key and collaboration is required to retrieve it. The distribution is done using Shamir’s secret sharing scheme. In a (t,n) threshold scheme, the secret is distributed to n peers allowing to compute the secret with the data of any t peers, but obtaining no information about the secret in case t-1 collude. The challenges are determining the size of t in a dynamic environment. Besides, traditional secret sharing has its disadvantages as the secret shares which have been distributed remain unchanged, posing a threat to backward secrecy. Further malicious secret share holders may collude.

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

A third major problem is resource management and scheduling problem.

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

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

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

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

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

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

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

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

2.1 F/OSS

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

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

2.2 Multi Player Online Gaming

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

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

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

3 Generic Model for Collaborative Peer Groups

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

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

3.1 Assumptions

We make the following assumptions for our model:

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

3.2 Peer Characteristics

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

– Peers have an initial self proclaimed rating which is the initial trust value
– Peers are assigned different roles based on their functionality.
– Peers in a group can belong to different levels based on capability. A Group Role assigned to a peer is independent of the level which the peer is in.
– Peers are individually capable of performing tasks of authentication, voting, access control, key management etc.
– Peers can compute their own public-private key pair and provide self signed certificates, binding their identity with their public key.

3.3 Group Charter

Each group has a group Charter with information about:

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

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

### 3.4 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
Task Oriented Activities  These can be classified as:

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

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

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

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

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

3.5 Group Roles and System Roles

A peer can either a service provider or a rater. Quality of a peer as a service provider is independent of the quality of the peer as a rater. The additional functions that a peer is capable of performing in a group are storing and verifying certificates, authentication, voting, updating of levels, key management etc. The extent to which a peer performs these tasks is decided by the role the peer wants to play in the group. We define two types of Roles viz Group Roles and Domain Specific Roles.

Group Roles  A peer can take on 3 different Group roles.

- Member peer  A peer in this role is a minimal functionality peer who participates in the normal group activities but does not contribute to the admission of new peers, nor updating of levels of existing peers. Thus this peer is only a service provider.
- Admission Peers  Those which are allowed to register new group members.
- Control peer  The control peers are the super nodes of the framework. Every group must have at least one control peer. These peers are responsible for broadcasting essential messages like
  1. Joining of a new peer
  2. Policy change(if it happens)
3. Updated ratings of peers. Thus a peer in this role would have all the functional components and would participate in periodically updating levels of existing peers and also permitting change of role. These peers are also responsible for monitoring group activity and keeping a track of group performance index GPI. If GPI falls below a certain threshold then a CP can call for a consensus of AP’s or MP’s to decide on a policy change. Framing of new policies like adding a new level to the group can also be done by a CP.

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

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

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

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

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

4 Group Operations and Maintenance

4.1 Group Initialization and Advertisement

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

4.2 New Peer Join

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

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

Admission Request

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

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

\[
P_{\text{new}} \rightarrow P_i : \{\text{JoinREQ}\} s_{K_{\text{new}}}, \text{Cert}_{\text{new}}
\]

where

\[
\text{Cert}_{\text{new}} = \text{UUID}_{\text{new}}, \text{PK}_{\text{new}}, \text{Rating}_{\text{new}}, \{H[\text{UUID} || \text{PK}_{\text{new}}]\} s_{K_{\text{new}}}
\]

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

Authentication

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

\[
P_i \rightarrow \text{All} P_n : \{\text{JoinREQ}\} s_{K_{\text{new}}}, \{\text{Cert}_{\text{new}}\} s_{K_i}, \text{VoteReq}
\]

Voting and Authorization

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

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

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

4.3 Peer evict

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

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

4.5 Member Freeze

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

4.6 Group Removal

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

5 Group Goals

The goals of collaborative peer groups are

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

5.1 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

6 Hybrid Access Control Model

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

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

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

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

The Basic trust metrics in our model are:

**Reputation** This is a metric measuring overall quality of previous transactions between peer x and y. We represent $f_E(x, y)$ as the feedback given by peer y for peer x for a transaction. Let $a_i = a_1, a_2, a_3, ......a_n$ be the set of attributes used for computing the feedback. 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 $f_{a_i}(x, y)\in[0, 1]$ is the feedback score given by peer y about peer x for attribute ai. Relative importance assigned to each attribute can be modelled as weight $w_{ai}$ such that $\sum w_{ai} = 1$. Assume that peer y stores upto n feedback ratings of previous transactions with peer x. Then reputation of peer x is calculated by peer y as

$$R(x, y) = \frac{\sum_{i=1}^{n} \alpha^i . f_{E_i}(x, y)}{\sum_{i=1}^{n} \alpha^i} \quad \text{(1)}$$

where $f_{E_i}(x, y)$ denotes the ith feedback given by peer y to x and $\alpha\in[0, 1]$ 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.

**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.

$$T(D) = \frac{\sum I(x, y).R(x, y).Cr(y)}{\sum I(x, y).Cr(y)} \quad \text{(2)}$$

where $R(x, y)$ is the reputation which can be calculated using equation 1, $I(x, y)$ is the number of times peer x interacted with peer y, $Cr(y)$ denotes the credibility of peer y.
**Indirect Trust** If a peer does not have sufficient number of transactions with a resource it can ask the opinion of other nodes to gain an overall evaluation $T_{ID}$. Assume that each user $x$ receives job ratings for peer $y$ from $n$ references $k = (1, 2, \ldots, n)$ and all nodes use the same decay factor $\alpha$. Then user $x$ can compute the reputation $f_E(k_i, y)$ of each indirect peer $k_i$ to resource $y$.

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

(3)

**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 $x$. It is a positive factor for correct ratings and a negative factor for incorrect ratings. Thus we increase the credibility factor of peers who give proper recommendations and decrease the credibility of peers giving incorrect recommendations.

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

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

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

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.

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

7 Modeling Malicious Peers

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

- Simple malicious peer. We model these peers as those who under-perform in the task allocated to them in the group, and provide poor ratings to others.
- Hypocritical peers. These peers behave properly for a period of time in order to build up a strongly positive reputation, then begin defecting. They could be random or structured. Random peers act maliciously with some probability say 20%. Structured peers follow a fixed pattern, say provide one malicious service after four good ones.
Colluding Peers. Malicious peers who co-operate with each other and increase each others rating while decreasing the ratings of other peers.

To model malicious peer behavior each peer builds a knowledge base represented by three list structures viz: Doubted List, Black List and Trusted List. The doubted List contains unique user ID (UUID) and public key of peers who have suddenly given deviated performance. If the difference in trust value of some peer X as calculated by peer Y in earlier transactions is greater than some threshold $\epsilon$ then a peer is included in the doubted list.

If peers in the doubted list continue giving bad performance then they are moved to Black list and the trust policy does not permit transactions with peers in this list. The Black List is purged according to an ageing policy. This strategy helps us to slowly remove peers whose misbehavior is dependent on some pattern. A peer in the doubted list is given a chance to improve his performance by allowing transactions and is moved to Trusted list if the trust value crosses the desired threshold. The control peers have a view of these lists and can broadcast the information about malicious peers after consensus.

7.1 Dynamic Policy Driven Trust Based Access Control (DPDTBAC)

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

Multiple policies can apply to a group, and the system can switch between policies over time, influenced by the system’s state or environment. Some policies can have higher priorities than others. Policies can apply to different levels of the system and can be global or apply to only parts of the system.

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

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.
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.

7.2 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. Cassandra [35] is an authorization language that defines the actions of activating a role and deactivating a role. Users can thus write state-dependent and implicitly state manipulating policies, but this rather ad-hoc approach is inflexible. Some languages such as Ponder [36] support obligation policies. An obligation is a task to be executed after evaluating and enforcing an access request. However it does not provide a precise semantics for the state changes. Some work has been done on analyzing security properties in dynamic role-based systems, in the context of the role-based authorization language RT [37] and Administrative RBAC (ARBAC) [38], where members of administrative roles can modify the role membership and privilege assignments. In [41], policies written in Datalog can refer to facts in the authorization state, as in our model. Hezberg et al. propose in [40], a prolog-based trust management language, but do not focus on dynamically changing policies with the state of the environment.

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

$$Ro(uo):-R1(u1),.......,Rn(un)$$

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:

```
Allow(d,submit_code,c):-d(developer),c(code),phase(submission)
Allow(r,review_code,c):-r(reviewer),c(code),assigned(r,c)
Allow(r,read_scores,c):-r(reviewer),c(code),has_reviewed(r,c),phase(assessment)
Deny(d,read_scores,c):-d(developer),code(c).
```

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

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

\[
\text{member(member). admission(admission_peer). maximal(maximal_peer).}
\]

\[
\text{verify}(Npeer, Request, Rl, Level, Rate, SPR) :- join(Request),
\]

\[
\text{member(Rl), Level =:= 1, SPR >= 40, assert(belongs(Npeer, Level))).}
\]

\[
\text{verify}(Npeer, Request, Rl, Level, Rate, SPR) :- join(Request),
\]

\[
\text{member(Rl), Level =:= 2, Rate >= 3, assert(belongs(Npeer, Level))).}
\]

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 rating.

\[
\text{verify}(Npeer, Request, Rl, Level, Rate, SPR) :- belongs(Npeer, Level - 1), update(Request), admission(Rl), Level =:= 2, Rate >= 5, 
\]

\[
\text{retract(belongs(Npeer, Level - 1))}, 
\]

\[
\text{assert(belongs(Npeer, Level))).}
\]

\[
\text{verify}(Npeer, Request, Rl, Level, Rate, SPR) :- belongs(Npeer, Level - 1), update(Request), maximal(Rl), Level =:= 3, Rate >= 8, 
\]

\[
\text{retract(belongs(Npeer, Level - 1))}, 
\]

\[
\text{assert(belongs(Npeer, Level))).}
\]

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

\[
\text{update_engine}(\text{Levels}, \text{MPs}, \text{APs}, \text{CPs}) :- \text{CPs} > 20,
\]

\[
\text{retract((verify}(Npeer, Request, Rl, Level, Rate, SPR) :- 
\]

\[
\text{update(Request), maximal(Rl), Level =:= 3, Rate >= 7)),
\]

\[
\text{assert((verify}(Npeer, Request, Rl, Level, Rate, SPR) :- update(Request), 
\]

\[
\text{maximal(Rl), Level =:= 3, Rate >= 9)).}
\]

\[
\text{update_engine}(\text{Levels}, \text{MPs}, \text{APs}, \text{CPs}) :- \text{CPs} < 20,
\]

\[
\text{retract((verify}(Npeer, Request, Rl, Level, Rate, SPR) :- 
\]

\[
\text{update(Request), maximal(Rl), Level =:= 3, Rate >= 9)),
\]

\[
\text{assert((verify}(Npeer, Request, Rl, Level, Rate, SPR) :- update(Request), 
\]

\[
\text{maximal(Rl), Level =:= 3, Rate >= 7)).}
\]

Here update_engine function can be used to update the new rule by retracting the old rule and then assert a new rule. update_engine takes 4 inputs as Levels(no of levels), MPs(% of member peers), APs(% of admission peers), CPs(% of control peers).
To address privacy issues the domain plays an important role. Domain or context specific policies refer to the set of rules/regulations/requirements of a domain to which an entity must conform to in order to be in that domain. To implement privacy based access, a variable can be attached to the information to indicate the privacy level of information. So if full privacy is chosen then the information will be sent with encryption and signature. Domain specific policies are part of a dynamic policy database and could be modified by peers, depending on peer behavior, group composition and evolving trust of peers in the group.

Thus we have a policy based framework which interacts with the evolving trust parameters of peers in the group. Figure 2 depicts the interaction between global policies, application specific meta policies and the trust engine for peers having multiple roles in multiple levels. All peers maintain their local view of the policies and any policy change is broadcast by the control peers. Good peers rebuke the misbehaving peers by degrading their trust value and including them either in the doubted list or in the black list. Transaction is avoided with the peer in the black list. Thus, malicious peers are slowly isolated from the network. Incentive is provided to the good peers who give good service by increasing their trust value.

8 Modeling F/OSS

To model our P2P collaborative group we chose the self organizing group F/OSS [45] Active developers in such communities join and propose patches to existing code. Projects are evaluated by moderators on the basis of technical merit and elegance of contribution. Such systems and communities co-evolve with the contribution of a large number of participants [44].

8.1 Peer Roles

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

- R1 = MP ∪ Developer. Peers in this Role are responsible for contributing new features and fixing bugs. They can contribute to the group by developing and submitting code, but have no access to view/modify any code/resources submitted by other peers.
- R2 = AP ∪ 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/inspect code submitted by Level 1 users belonging to a particular sub-group which they have been assigned. They cannot modify code. The peers at this level too can take on the additional role of Admission Peer or Control Peer.
- Level 3 peers can modify code, reviews, fix bugs and take the final decision regarding code acceptance. Additionally they could take on Roles of Admission Peer and or Control Peer.

8.2 Task Allocation

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

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

Table 1. Variation of job allocation

8.3 Trust parameters and attributes

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

1. P → price of software.
2. D → Delivery time
3. R → Lines of Code
4. Q → Quality

8.4 Modeling Domain Specific Policies

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

In Alloc2, when a peer with rating between 0-55 enters the system, he is allocated an easy job if available, else he is not allocated a job at all. Thus peers are allocated jobs strictly according to trust values.
We formulated rules in prolog [?] to represent these policies. We are able to express dynamic rules at runtime, as well as change existing policies depending on the status of the group at runtime by exploiting the assert and retract clauses of Prolog. Thus the system switches between policies over time depending on the job composition, group composition and current trust values of peers in the group. The framework thus gives peers the flexibility to self regulate their behavior based on learning experience in the group. Sample domain specific policies to update role from Developer to Reviewer are given below:

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

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

A policy to introduce a new level is

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

9 Implementation and Results

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

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

9.1 Example Java Program

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

```java
PrologEngine engine = new SWISubprocessEngine("/usr/lib/pl-5.6.55/bin/i386-linux/pl"); File f = new File("verify.P"); engine.consultAbsolute(f);
// or consultRelative (to the class location), or consultFromJar(to the jar location),...
Object[] var = new Object[]{4,1}; boolean b =
engine.deterministicGoal("ipObjectSpec('java.lang.Integer',Obj1,[Peer],_),"+
"ipObjectSpec('java.lang.Integer',Obj2,[Level],_), belongs(Peer,Level)",
"[Obj1, Obj2]", var);
System.out.println(b);
```
9.2 Overall working of framework with prolog

Figure 3 shows the interaction between the events and policies in our framework. External events like join, leave, and internal events like level update, or application specific events like submitting a review, or submitting code, in case of F/OSS might trigger some other events, like change in policy. Change in policy in turn affects the way in which events like join, leave, update occur. All events are simulated in peersim simulator [42]. Simulator triggers the policy file which is prolog rules. Database contains all the information about all the peers. Prolog rules use the database and according to the changes in database, prolog rules or policy might change or be modified.

10 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.

10.1 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.

11 Implementation Details

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

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

11.1 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.

Analysis The graph in figure 4 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.

**Simulation2:** 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.

![Graph](image)

**Fig. 5.** 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 5 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 of medium complexity 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>Alloc</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 2.** Variation of job allocation

Analysis The graph in Fig 6 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>Composition</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>

**Table 3. Variation in job composition**

**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 % dif-
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 7 shows that Compos2, gives better results because of the proper distribution of jobs and peers. 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.

![Fig. 8. Job Success with varying Job Compositions and SPR](image)

Graphs in Fig 8 and Fig 9 show that in both cases dynamic policy gives better result than static.

![Fig. 9. Job Success with varying Job Compositions and varying join](image)

**Simulation 7** We defined 3 job allocation policies. Policy `job_alloc1` 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 `job_alloc3`, 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 job_alloc3 gave better results (see figure 10) and when the percentage of easy jobs entering was greater, policy job_alloc1 gave better results.

![Simulation 8](image.png)

**Simulation 8** We gave runs for 300, 400 and 500 peers in the system with varying job allocation policies and varying job compositions. It was again observed that when the percentage of difficult jobs entering the system was greater, policy job_alloc1 gave better results and when the percentage of easy jobs entering was greater, policy job_alloc3 gave better results. We then gave runs where the control peers trigger a policy change as per the type of jobs entering the system i.e dynamically switching between the policies.

![Fig. 11. Dynamic vs static job allocation policies](image.png)
The graph in Fig 11 shows that the dynamic policy allows the group to perform better as compared to different static allocation policies.

**Simulation 9** Here we gave varying join policies for a fixed job allocation policy and plotted the results for static versus dynamic join.

Graph in fig 12 shows that fully dynamic join gives best results.

![Graph showing job success for static versus dynamic join](image)

**Fig. 12.** Job Success for static versus dynamic join

**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 again with varying job allocation and join policies.

![Graph showing comparison of 10 percent vs 30 percent malicious peers](image)

**Fig. 13.** Comparison of 10 percent vs 30 percent malicious peers
Graphs in figure 13 and 14 show that in presence of malicious peers also dynamic policies give better performance as compared to static.

![Graph](image1)

Fig. 14. Comparison of performance with and without dynamic policies in presence of malicious peers

Graphs in Fig. 15 and Fig 16 are plots of the percentage of job completion vs time in the presence of 10 percent and 30 percent malicious peers respectively. Results show in case of static policy the completion time is much larger as jobs are allocated to under performing peers, whereas in case of dynamic policy jobs are allocated based on appropriate trust values of peers. Further for 30 percent malicious peers all the jobs could not be allocated using the static policy.

![Graph](image2)

Fig. 15. 10 percent malicious peers static vs dynamic

![Graph](image3)

Fig. 16. 30 percent malicious peers static vs dynamic
Simulation 11 We compared our work with that of TM [40] which is a trust model for P2P ecommerce based on Direct and Indirect trust. In the TM model attributes for peers are not considered, nor are peers segregated into a doubted list and black list. We performed a comparative analysis on a network of 100 nodes with 800 transactions in each simulation. The parameters used for comparison were
1. Average trust value of good peers
2. Average trust value of malicious peers
3. Transaction success ratio in presence of varying percentage of malicious peers
4. No of times the malicious peers are selected for transacting with increasing percentage of malicious peers.

We took runs for 5, 10, 15, 30 and 40 percent malicious peers in presence of static policy and dynamic policy. Results (see figure 17) show that in our model the trust value of good peers remains constant with increase in % of malicious peers and is better as compared to the TM model. Trust value of malicious peers degrades gradually in our model with increase in % of malicious peers whereas in case of TM model the trust value of malicious peers does not change much with increase in % of malicious peers.

![Fig. 17. Aggregate trust values of peers with 10 percent malicious peers](image)

Graph in Fig. 18 shows the trust value of malicious peers decreases considerably with our dynamic task allocation policy whereas for the P2P model [40] the malicious peers are able to maintain their trust value. Thus in presence of our dynamic trust policy, gradually with increase in interactions the group is able to isolate the malicious peers as their trust value falls beneath a certain threshold and they would not be allocated any further tasks.

![Fig. 18. Average trust ratings of malicious peers](image)

The graph in fig 19 shows the ratio of successful transaction in both the models with increase in percentage of malicious peers. Graph shows that in our model the successful transaction ratio does not degrade with increase in malicious peers while in the TM model it degrades considerably.
12 Conclusion

We have proposed a framework for collaborative groups which has a dynamic policy driven trust based access control model. The framework is flexible and peers are able to self regulate their behavior in presence of dynamic policies. The trust metric is a tunable metric with context specific weighted attributes. The dynamic polices make the trust model adaptive. Experiments show that dynamic policies give better group efficiency than static policies. Trust values of malicious peers decrease with time, thus making it possible to eject malicious peers from the group when the trust value drops beneath a specific threshold. Collaborative applications like F/OSS, multi-player online gaming and others can use this framework to test the evolution of peer groups and decide which policies to use to achieve a better job success rate.

References

7. Thomas Repantis, Vana Kalogeraki: Decentralized Trust Management for AdHoc Peer-to-Peer Networks, MPAC, November 27-December 1, ACM 2006
16. Wikipedia, the free encyclopedia

Fig. 19. Transaction success ratio


32. PeerSim Simulator Documentation From http://peersim.sourceforge.net/


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


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

43. XSB xsb.sourceforge.net/manual1/index.html

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