Collaborative Editing with Contract over Friend-to-Friend Networks

Hien Thi Thu Truong, Mohamed-Rafik Bouguelia, Claudia-Lavinia Ignat, Pascal Molli

Collaborative Editing with Contract over Friend-to-Friend Networks

Hien Thi Thu Truong
INRIA Nancy-Grand Est
Hien.Truong@loria.fr

Mohamed-Rafik Bouguelia
Henri Poincaré University
Mohamed.Bouguelia@loria.fr

Claudia-Lavinia Ignat
INRIA Nancy-Grand Est
Claudia.Ignat@loria.fr

Pascal Molli
Nantes University
Pascal.Molli@univ-nantes.fr

ABSTRACT

Instead of delegating control over private data to single large corporations, in friend-to-friend (F2F) systems users take control over their data into their own hands and they communicate only with users whom they know. In this paper we propose a push-pull-clone model for trust-based collaborative editing with contract deployed over F2F network. People specify contracts when they share data to their friends. A log auditing protocol is used to detect user's misbehavior if they do not respect contracts after they received data. Users maintain locally trust values on other users and these values are adjusted according to auditing results. We evaluate proposed model by using PeerSim simulator and experiment results show the feasibility of our model.

INTRODUCTION

Most platforms hosting social services such as Facebook or Twitter rely on a central authority and place personal information in the hands of a single large corporation which is a perceived privacy threat. Users must provide and store their data to vendors of these services and have to trust that they will preserve privacy of their data, but they have little control over the usage of their data after sharing it with other users. Peer-to-peer networks allow decentralization of these social services. Friend-to-friend (F2F) networks is a category of peer-to-peer networks where users have control over their data as they allow direct connection only between users who know each other. Restricting connections only between users that know each other solves a set of security problems in peer-to-peer networks such as the issue that a user appears under many identities. In F2F networks users have a unique identity and when a user collaborates with a friend it means that friend is well-known and trusted.

Dan Brickklin[1] has coined the term of F2F networking as "The simplification of restricting a network to computers you know, trust, and control is one method of dealing with trust issues in sharing". If users misbehave and their misbehavior is detected, the unique identification of users allows a punishment procedure to be adopted for these users. However, the challenge is how to detect if users misbehave and how to design the punishment procedure.

In this paper we focus on the application of F2F network for collaborative editing of shared documents. Collaborative editing systems allow a group of people to produce works together through individual contributions. Collaborators can edit together on articles, proposals, memos, etc. There are a wide range of users of CES such as students, professors, authors of a paper, journalists, editor, internal bloggers, etc. Handling the document in collaborative environment manner requires interactions between users. With F2F computing infrastructure, users only connect to trusted friends. Trust is a key ingredient for collaborative relationships of people and it requires repeated interactions to build trust based on experiences over time. Trust is mutable and not completely transitive between friends, and it is required to have a mechanism to update trust eventually. Distributed version control systems such as Git[2] and Mercurial[3] adopt a variation of the friend-to-friend model of collaboration where users pull changes from other users that they trust. These systems are mainly used by developers in open-source projects. Even if a trust network is built between developers based on previous experience, trust is not explicitly evaluated.

In a friend-to-friend collaboration it is very difficult to ensure that after data is shared with other users, these users will not misbehave and violate data privacy. Usage control mechanisms model contracts that users receive together with data which refer to what happens to data after it has been released to authorized people, for example, how they may, should and should not use it. However, existing usage control approaches rely on some central authorities that can check violation of contracts. These solutions are not suitable for F2F based collaboration where no user has control over another user to audit his behavior and a user can observe only behavior of collaborating users.

[1] http://www.bricklin.com/f2f.htm

[2] http://git-scm.com/

[3] http://mercurial.selenic.com/

In this paper, we propose a novel trust management mechanism adapted for friend-to-friend collaborative editing with contracts. Each user maintains a local workspace that contains local data as well as modifications done on the shared data and contracts related to usage policy of the shared data. The logged modifications and contracts are shared with other users. Each user performs a log auditing mechanism for detecting misbehaving users and adjusts their trust values according to audit result.

The paper is structured as follows. We start by presenting a motivating example. We then go on by presenting related approaches. We give an overview of the collaboration model that we propose, of the logging mechanism and the properties it has to ensure and of the contract specification. We then briefly describe our solution to assess trust. We also provide some results of the simulations to evaluate the efficiency of our mechanism. We end our paper with some concluding remarks and directions for future work.

EXAMPLE

Let us give a simple example for illustrating how collaborative editing with contract is deployed over a F2F network. Let us consider a F2F network composed of five users A, B, C, D and E in which each user is at least in one friend relationship with another user. Users share and edit collaboratively a document. At the beginning this F2F network is built based on social trust between users and connections are established only between friends. Friends trust their friends with different trust levels that are updated according to their collaboration experience. For instance, A has B and C as friends, however, A has different trust levels for B and C and thus gives them different contracts over the shared document. For instance, A gives to B the permission to edit the document, while he/she gives to C only the permission to view the document. Also, A gives C an obligation to send feedback while this obligation is not required to B. Receivers are expected to follow these contracts. Otherwise, their trust levels will be adjusted once misbehavior is detected. To our knowledge there is no existing model for collaborative editing which considers trust with self-auditing and updating mechanism of trust levels.

RELATED WORK

Our approach based on trust management is different from access control mechanisms. Instead of enforcing granted rights a-priori as in traditional access control mechanisms, we check contracts a-posteriori once data has been shared. Similarly, there exist some optimistic access control approaches [11] that check a-posteriori access policies. In these approaches, if user actions violate granted rights, a recovery mechanism is applied and all carried-out operations are removed. Usually, this recovery mechanism requires a centralized authority that ensures that the recovery is taken by the whole system. However, the recovery mechanism is difficult to be applied in F2F system where a user does not have knowledge of the global network of collaboration. Generally, access control mechanisms aim at ensuring that systems are used correctly by authorized users with authorized actions. Rather than ensuring a hard system security, we adopt a flexible trust management mechanism that helps users collaborate with other users they trust.

In the last decade friend-to-Friend (F2F) computing emerged as a new paradigm in which users only make direct connections with people they know. Freenet [1] is a well-known F2F network whose main design goals are censorship resistance and anonymity. Freenet can operate in the classical F2F (darknet) mode where the users can only join the network only if they know a member of the network, but also in the opennet mode in which anyone can join. In [2] authors outline solutions of using F2F computing for building decentralized social networks, for data access control and for data storage. In [8] authors describe the F2F computing framework to spontaneously setup Desktop Grids with friends or colleagues via instant messaging. While most of existing F2F networks are used for file sharing and instant messaging, we propose F2F computing for deploying trust-based applications such as decentralized collaborative editing with trust.

Trust management is an important aspect of the solution that we proposed. The concept of trust in different communities varies according to how it is computed and used. Our work relies on the concept of trust which is based on past encounters [7]. With F2F networks users bring social trust into the system. However trust is not immutable and it changes time to time. Thus a trust model still be useful for F2F network. Various trust models for decentralized systems exist such as NICE model [10], EigenTrust model [4]. As taxonomy of trust in [6], trust model includes information gathering, scoring and ranking, response. Most of existing P2P trust models propose mechanism to update trust values based on direct interactions between peers while we use log auditing to help one user evaluates others either through direct or indirect interactions. Our mechanism for discovering misbehaved users can be coupled with any existing trust model in order to manage user trust values.

Keeping and managing event logs is frequently used for ensuring security and privacy. This approach has been studied in many works. In [3], a log auditing approach is used for detecting misbehavior in collaborative work environments, where a small group of users share a large number of documents and policies. In [5, 9], a logical policy-centric for behavior-based decision-making is presented. The framework consists of a formal model of past behaviors of principals which is based on event structures. However, these models require a central authority that has the ability to observe all actions of all users. This assumption is not valid in a F2F setting. Our proposed log-auditing mechanism works for a F2F collaboration and its complexity compared to the centralized solution comes from the fact that each user has only a partial overview of the global collaboration and can audit only users with whom he collaborates. Therefore, a user can take decisions only from the information he possesses from the users with whom he collaborates.

SYSTEM MODEL

In this section we are going to present collaborative editing system based on push-pull-clone model deployed over the F2F network. The collaboration implies the interactions between users under given contract. Our work focuses on logging mechanism which mainly point are to ensure causality of not only editing operations but also of contracts and to perform synchronizing correctly. We move toward first with the infrastructure of F2F network applied for our push-pull-clone model.

Push-pull-clone model over F2F computing

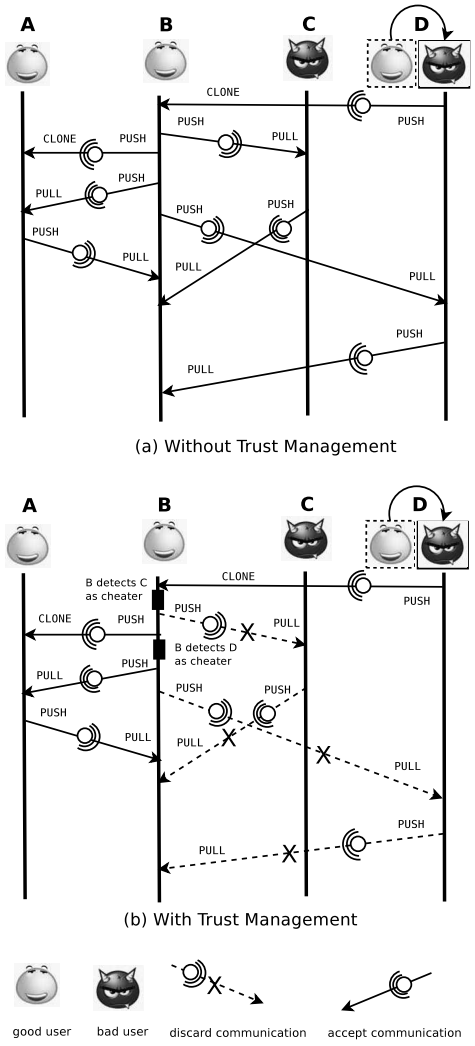

Figure 1. Push-pull-clone model in peer-to-peer network. Without trust management, users might push private data to malicious user, and pull respectively. With trust management, users discard all communications to/from malicious users since such users are detected as bad users (e.g. when D changes from good to bad).

Push-pull-clone is a native direct pair-wise communication between users. Users can collaborate without having to give everyone direct write access to their document. To work with others a user simply sets up a local workspace for his own work, and uses a F2F network to PUSH his document to trusted friends. Others can then get the document by doing CLONE the document from the user's workspace to have independent local workspace for the document of the user, do their changes locally and publish them by doing PUSH. The user then does PULL to get the changes into his local workspace. The push-pull-clone model is used for efficient distributed collaboration and it was already implemented in distributed version control systems such as Git, Mercurial.

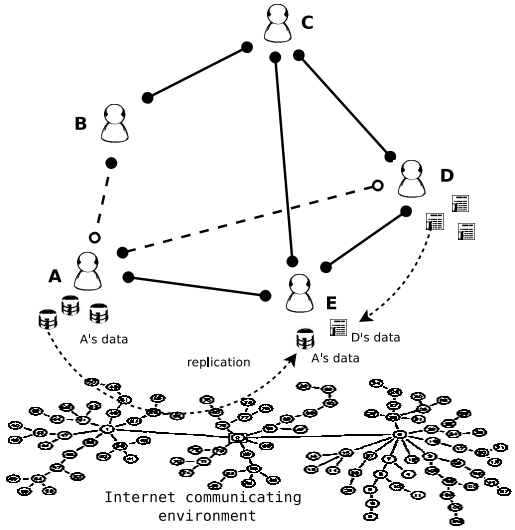

Figure 2. Friend-to-Friend network with trust. Collaborations are established between friends only. No collaboration between A and B (also A and D) since A might trust B but B does not trust A (respectively A and D). Trust is not transitive between friends (between A, E, and D).

In Figure 1, there are two scenarios of push-pull-clone model for collaborative editing with and without trust management. In the former without trust management, users share and reconcile replicas with any one another regardless such user is good or bad. Such collaboration could be in danger of accepting bad content from malicious users who attempts to steal information or disturb the cooperation. The later scenario supports only push-pull-clone interactions between trusted users (so-called friends). It works also with the case a user changes his behavior from good to bad, thus his trust or reputation becomes less, therefore his communications in collaboration will be discarded.

In Figure 2 sharing data as well as cooperation are performed between friends only. The network structure is not immutable since each member can change behavior over time. Thus, their trust levels are adjusted mainly based on their past behavior. A misbehavior detection mechanism and trust model to manage trust levels of friends are mandatory to maintain correctly such network. Next, we present our solution serving for auditing misbehavior and updating trust levels.

Log

Our system keeps document as a log of operations that have been done during collaborative process. The log records more about the process of collaboration itself, and not just output document. From the log it is possible to see who contributes what, and when. The outcome of collaborative editing needs to be a document and it could be obtained by

replaying the editing operations from the log.

The document on which users collaborate together is replicated at sites of all users. In our model the changes are propagated in weakly consistent manner. It means user can decide when, with whom and what data to be sent and synchronized. We use the push-pull-clone communication between replicas and the ordered exchange of operations stored in each replica logs. A replica log contains all operations that have been generated locally or received from other users. Users store operations in their logs in an order that is consistent with generated order. Pair-wise communication supports the synchronization of any two users. The term *event* presents editing operations and communication action such as *share*. The following rules are used for logging:

- Each new event generated locally is added to the end of local log in the order of occurrence.
- The events from remote log that should be merged are added to the end of local log in the order they appeared in the remote log.
- When a user shares the document with other users the *share* events together with the specified contracts are logged in the local log of the receiving user. However, we make the assumption that a user is unwilling to disclose with all collaborating friends all the sharing events and contracts that he specifies to a certain friend. That is why sharing events and contracts are not kept in the local log of the sending user.

We use anti-entropy update for new changes from others logs received. New changes are appended to the end of the log. A user might discard changes from a received log in case of contract conflict. The merging mechanism ensures causality of editing operations as well as between operation and contract. We discuss in detail about these properties in next parts.

Causality preservation

Many distributed systems consider causality preservation as a requirement and they use techniques such as state vectors and causal barriers for ensuring it. In our model, it is mandatory to respect the following causalities.

- Causality between local operations is respected by adding events to the log in order they were generated.
- Causality between local operations and received remote operations re respected by anti-entropy updating for new changes. Also the propagating of the whole log ensures that order of events are not changeable.
- Causality between contracts and operations must be ensured in order to enforce users following given usage condition. Otherwise their violations are detected to perform judgments. Log itself cannot prevent users from changing this causality up on their purpose. Thus we use signature-based authenticators to ensure correctness of the log.
- Causality between received contracts and contracts give to one another in order to ensure user should use data based on usage policy (through contract) he receives. This causality also is preserved with authenticators.

Contract specification

Contract is a need for trustful collaborative interactions. In our collaborative editing model, contract expressed usage policy which are the *obligation*, *permission*, *omission* and *prohibition* (or forbidden) which one user gives to others when he shares data. These four notions are objects of deontic logic. The specification of contracts needs to capture the causality of deontic objects, thus it needs to use temporal logic. The contracts in our model are built on the top of a basic deontic logic, in combination with operators from temporal logic, and operation-based.

Deontic logic is the logic concerned with normative notions such as obligation, permission, omission, prohibition, etc. Deontic itself when restricted to specific domain is a practical powerful specification tool, and if combined with other useful temporal concepts. Due to the huge scope of things can be expressed by deontic logic, we restrict our work with ing deontic modalities of Obligation, Permission, Forbidden, Omission (denoted as O, P, F, M).

The contracts are expression of deontic notions over operations in CES. If *op* is an operation then the primitive contract c_{op} based on *op* is defined using the modalities: F_{op}, O_{op}, P_{op}, M_{op} and to be respectively read "contract c says that it is forbidden to do the operation op (or obliged, permitted, omitted)". The composite regulation (simply called *contract*) C can be composed of one or several primitive contracts. In order to obtain single composite contract, primitive contracts are merged together, however, the conflict arises in merging primitive contracts together. Primitive contracts are conflict in cases: $\mathrm{O}_{\mathrm{op}} \nmid \mathrm{O}_{\neg\mathrm{op}}$, $\mathrm{P}_{\mathrm{op}} \nmid \mathrm{F}_{\mathrm{op}}$, $\mathrm{O}_{\mathrm{op}} \nmid \mathrm{M}_{\mathrm{op}}$. Conflict can be resolved by detecting conflict contracts and giving them the priority of one over the other (so-called dominance). User can also overrides his own contracts in collaboration process with one another user.

AUDITING AND TRUST ASSESSMENT

We consider a collaborating system where each user is supposed to respect given contract. If it does, then we call the user *trusted*; otherwise we call the user *distrusted* or *suspected*. There are two ways in which a user cannot be *trusted*: it can either do actions violating contract or ignore an obligation. Ideally, if a user misbehaves in either of these ways, other users should detect his misbehavior. A user is considered as distrusted if it violates a contract, and a user is considered as suspected if he does not prove that he conforms to an obligation. For instance, a user that receives the obligation *"should send back"* the document, but he never fulfills this obligation is considered suspected. Note that user u withdraws the suspected indication on user v when he finds that user v fulfilled the obligation he was suspected for.

Cheaters may try to hide their misbehavior by modifying the log. Briefly, a user u is considered $cheater$ if he modifies the log that consequently affects auditing result. For

instance, u removes some obligations that he does not want to be enforced to follow. The log auditing mechanism should guarantee also that users cannot modify the log. To prevent log modification, we use authenticators for patches of operations. Due to the space limitation, we do not present in this paper our solution about generation and verification of authenticators. The audit mechanism can generate only four types of results: *trusted*, *distrusted*, *suspected* and *cheater*.

The auditing protocol is performed at any time at local sites. We denote $Trust_u^{log}(v)$ as the trust value that a user u assigns to user v. All users are set an initial default trust value such as real life social trust. The user u updates value $Trust_u^{log}(v)$ for user v mainly based on the result of log analysis. In order to manage trust values for user v, we can use an existing decentralized trust model. When an assessed user v is detected as *distrusted* or *suspected*, his local trust value is recomputed by assessing user u using a trust model. The total local trust values could be aggregated from log-based trust, reputation and recommendation trust. The trust computation to get total trust values varies from trust models. Details of trust model are not presented in this paper.

Our approach for trust assessment uses log auditing. The violation in case a cheater copies the content of a document in order to create a new one and then claims to be the document owner can not be detected by log auditing. However, the log could be used to discover the history of actions on document that helps to detect cheaters.

EVALUATION

In this section, we evaluate the feasibility of the proposed model through simulation using PeerSim[4] simulator. We focus first on the ability of detecting cheaters; then we estimate the overhead generated by the usage of contracts.

We setup the simulation with a network of 200 users where some of them are defined as cheaters. Due to the unavailability of real data traces of F2F collaboration as well as traces with contracts, we generate randomly the data flow of collaboration during the simulation, i.e operations, contracts and users with whom to share. One interaction is defined as the process of sharing a log with the specified contracts, from one user to one another. Since the total number of interactions generated must be pseudo uniformly distributed over all users, we let one user perform sharing to not exceed 3 other users at each step. Similarly, the number of operations and contracts generated by one user each time is at most 10 operations and 3 contracts (if we consider only 3 types of actions in our system: insertion, deletion and sharing).

Experiment 1: misbehavior detection

To evaluate the ability of misbehavior detection, we check first the ability to detect a selected cheater according to the total number of interactions performed by all users. The estimation is performed on the collaborative network with 60 cheaters (30% of users are cheaters). The auditing process is performed after each synchronization with another user.

[4] http://peersim.sourceforge.net/

We select randomly one cheater to be audited, and we observe how many percent of users can detect him. Figure 3 shows the results collected after each cycle. We can see that the cheater is detected by a few users at the beginning, then more user can detect him when more interactions are performed.

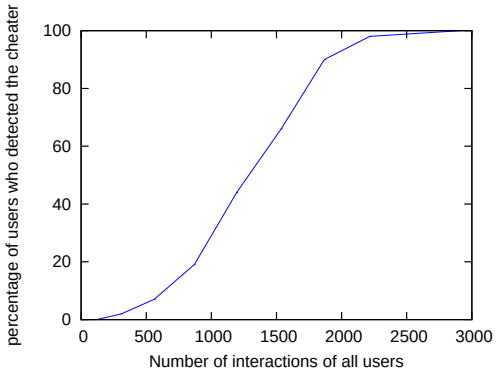

Figure 3. Ability to detect one selected cheater according to the total number of interactions in the collaborative network.

Second, we check the percentage of cheaters that can be detected. We select randomly one honest user from the network to observe how many percent of cheaters he can detect. Figure 4, shows the result according to the number of synchronizations done by the selected user with others. We can see from the graph, that up to 20% of cheaters are detected after the first four synchronizations (audit done four times), and after the fifth synchronization more than 80% of cheaters are already detected. Only about 10% of cheaters may require more interactions to be detected.

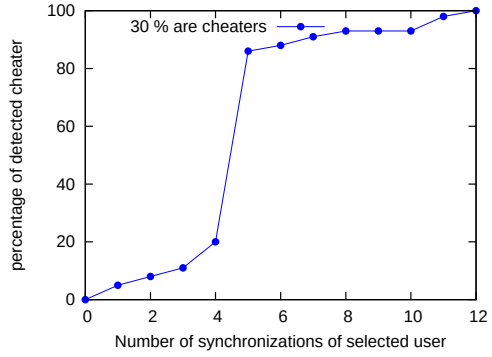

Figure 4. Percentage of detected cheaters according to the number of synchronizations done by the selected honest user.

In order to have a global view about the evolution of the percentage of detected cheaters, we compute the average value of detected cheaters over all users of the collaborative network. Figure 5 shows, on average, how many percent of cheaters are detected by one user. We perform the experiment in case of a low, medium and high population of cheaters in the network (respectively 5%, 30%, 80% of cheaters). The results show that the system still functions well in case of a high/low population of cheaters.

Experiment 2: overhead estimation

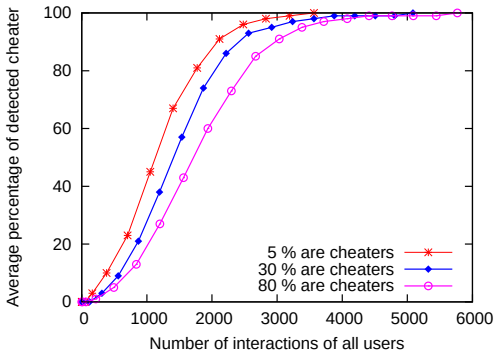

Figure 5. Average percentage of detected cheaters according to the total number of interactions in the collaborative network.

In this experiment we evaluate the time overhead generated by the usage of contracts for the synchronization and the auditing process. We compare two collaborative models: with and without contracts. In the model with contracts, the synchronization includes merging logs which contain operations and contracts, replaying editing operations, and auditing misbehavior. In the model without contracts, the synchronization is simply performed by anti-entropy update.

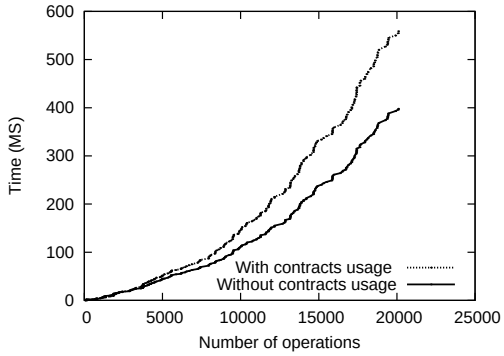

Figure 6. Synchronization time with growing of number of operations

We perform an experiment where we use the same data flow for each model, and we compute the total time (T) of all the synchronizations performed by a given user. i.e. T = $\sum t_i$, where t_i is the time required for the i^{th} synchronization. Figure 6 shows the result according to the number of operations in the local log. From the obtained results, we can see that the time overhead generated by the usage of contracts, is reasonable, since the difference between the time computed for both models, increase slowly with the number of operations. Our experimental evaluation shows the feasibility of the proposed method for detecting cheaters during collaboration over F2F network even with a high population of dishonest users.

CONCLUSION

We presented a push-pull-clone collaborative editing model over a friend-to-friend network where users share their private data by specifying some contracts that receivers must follow. Trust values are adapted according to users' past behavior regarding conformance to received contracts. Modifications done by users on the shared data and the contracts that must be followed when data is shared are logged in a distributed manner. A mechanism of distributed log auditing is applied during collaboration and users that did not conform to the required contracts are detected and therefore their trust value is updated. Any distributed trust model can be applied to our proposed mechanism. We implemented the proposed collaboration model and we performed simulations using PeerSim peer-to-peer simulator. Experiment results show the feasibility of our model.

REFERENCES

1. I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.
2. W. Galuba. Friend-to-Friend Computing: Building the Social Web at the Internet Edges. Technical Report LSIR-REPORT-2009-003, EPFL, 2009.
3. J.G.Cerderquist, R.Corin, M.A.C.Dekker, S.Etalle, J. Hartog, and G.Lenzini. Audit-based Compliance Control. *Information Security*, 9(1):37–49, march 2007.
4. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web*, 2003.
5. K. Krukow, M. Nielsen, and V. Sassone. A logical framework for history-based access control and reputation systems. *Journal of Computer Security*, 16(1):63–101, 2008.
6. S. Marti and H. Garcia-Molina. Taxonomy of Trust: Categorizing P2P Reputation Systems. Working Paper 2005-11, Stanford InfoLab, 2005.
7. L. Mui and M. Mohtashemi. A computational model of trust and reputation. In *Proceedings of the 35th Hawaii International Conference on System Science (HICSS)*, 2002.
8. U. Norbisrath, K. Kraaner, E. Vainikko, and O. Batrasev. Friend-to-Friend Computing - Instant Messaging Based Spontaneous Desktop Grid. In *ICIW*, pages 245–256, 2008.
9. M. Roger and J. Goubault-Larrecq. Log Auditing through Model-Checking. In *Proceedings from the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, 2001.
10. B. B. Seungjoon Lee, Rob Sherwood. Cooperative Peer Groups in NICE. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 50(4):523 – 544, 2006.
11. G. Stevens and V. Wulf. A new dimension in access control: studying maintenance engineering across organizational boundaries. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, CSCW '02, pages 196–205, New York, NY, USA, 2002. ACM.