University of Zurich
Department of Informatics

Thomas Bocek
Dalibor Peric
Fabio Victora Hecht
David Hausheer
Burkhard Stiller

Towards a Decentralized Voting Mechanism for P2P Collaboration Systems

TECHNICAL REPORT — No. ifi-2009.02

March 2009

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

T. Bocek, D. Peric, F. V. Hecht, D. Hausheer, B. Stiller: Towards a Decentralized
Voting Mechanism for P2P Collaboration Systems
Technical Report No. ifi-2009.02, March 2009
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Towards A Decentralized Voting Mechanism for P2P Collaboration Systems

Thomas Bocek, Dalibor Peric, Fabio Hecht, David Hausheer, Burkhard Stiller

Department of Informatics IFI, University of Zurich, Switzerland
bocek,peric,hecht,hausheer,stiller@ifi.uzh.ch

Abstract. Peer-to-peer (P2P) systems achieve scalability, fault tolerance, and load balancing with a lower-cost infrastructure, from which collaboration systems, such as Wikipedia, could benefit from. A major challenge in P2P collaboration systems, however, is to maintain article quality after each modification at the presence of malicious peers. Therefore, to allow modifications to take effect will be only possible, if a majority of previous editors approve changes by voting. This determines a challenge in P2P systems, due to the absence of a central authority. Thus, this paper proposes the fully decentralized voting mechanism PeerVote, which enables users to vote on modifications in articles in a P2P collaboration system. Simulations and experiments show the scalability and robustness of PeerVote, even at the presence of malicious peers.

1 Introduction

Peer-to-peer (P2P) systems inherently support redundancy, scalability, fault tolerance, and load balancing [20, 16]. P2P systems support these features at a lower cost than client/server systems, since every participating user contributes with resources. These advantages incited the emergence of different P2P-based applications, including audio and video streaming [28], file sharing [7], and storage [24, 27]. A collaboration system, such as Wikipedia [26], could also benefit from the aforementioned characteristics of P2P systems [3]. In a P2P collaboration system, users share the resources necessary to host and distribute articles that can be modified by any other user.

A major challenge in P2P collaboration systems is to assure that user-generated content quality is being maintained or improved after each modification, despite the lack of a central authority. Since article quality is a highly subjective measurement, a user-based voting is needed, which allows users to vote on whether or not a proposed modification shall be accepted.

The contribution of this paper is PeerVote, a decentralized voting mechanism that provides a method to maintain quality of content and its modifications. The proposed voting mechanism ensures that every modification to a document has the approval from the majority of previous editors. This helps to prevent vandalism, editing wars, and deliberate censorship. PeerVote has been implemented in a P2P collaboration application and evaluated on top of a structured P2P network using a simulator and EMANICSLab [10].

2 Related Work

One way of achieving the goal of improving quality of content in P2P collaboration systems is by using recommendation or reputation systems. [1] proposes a content-driven reputation system for Wikipedia, which is based on automatic analysis of edit-changes of Wikipedia articles and side-steps any user-based rating. The deriving author reputation can be used to predict the quality of future articles of such authors. [14] outlines the integration of a recommendation system in Wikipedia, which allows users to give their preference about the article and consult its general score, whereas the resulting system would still remain centralized. Although such systems can be used in conjunction with PeerVote, their focus is on the document itself, while PeerVote's main concern is on document's update compliance. While several P2P Wiki approaches exist [18, 25, 17, 23], none of these approaches support user-based voting.

Different voting and consensus reaching algorithms have been proposed, with distributed systems and electronic voting as their main applications. In distributed systems, the main purpose of voting mechanisms is to ensure consistency among replicated data, usually by achieving an agreement between replica holder entities. Examples of such consensus protocols are the two-phase commit protocol [12], the weighted voting [11], and the decentralized weighted voting [19]. With the exception of the latter, such voting schemes have the disadvantage of being centralized. Another disadvantage of those schemes is that they are synchronous, thus, their application for human-based voting would require all voters to express their preference simultaneously, which is technically unfeasible for large numbers of participants.

Fully decentralized voting protocols, such as the inexact voting over wide area networks [13] and the Deno voting protocol [5] have been proposed. The first approach shows a message complexity quadratic with respect to the number of voters, which is not scalable. The Deno voting protocol, while being scalable, does not guarantee that updates commit in a bounded time.

Secret ballot protocols, or electronic voting protocols, implement a democratic voting system on electronic equipment. Some existing implementations [6, 21] address specific security concerns, like eligibility, privacy, individual and universal verifiability, fairness, robustness, and receipt-freeness, but they work in a centralized fashion.

Table 1 shows a comparison among these voting mechanisms. None of these mechanisms reviewed support a decentralized user-based voting.

3 PeerVote Design

The design of PeerVote defines roles and their interaction. Each role defines a set of actions and interacts with one or more other roles. PeerVote defines six roles: user, storage, editor, mediator, tracker, and voter roles (cf. Figure 1) and one peer can have one or more roles, depending on the action of a user.

Table 1. Comparison of distributed voting mechanisms

	Decentra-lized	**Scalable**	**Time-bounded**	**Asyn-chronous**	**User-based**
2-Phase Commit	No	Yes	Yes	No	No
Weighted Voting	No	Yes	Yes	No	No
Decentralized Weighted Voting	Yes	Yes	Yes	No	No
Secure Ballot	No	Yes	Yes	Yes	Yes
Inexact Voting	Yes	No	Yes	Yes	No
Deno Voting	Yes	Yes	Yes	No	No
PeerVote	Yes	Yes	Yes	Yes	Yes

Tracker Role: A tracker peer stores references to storage peers and evaluates voting meta data. A tracker can reject references with wrong or inconsistent voting meta data. References have a time to live value and if peers stop (re)publishing, these references are removed.

Storage Role: A storage peer stores and periodically (re-)publishes documents. Publishing is carried out by searching for close trackers to the document's id and storing addresses of storage peers and meta data. These documents are stored by peers that either created or viewed them.

User Role: A user interacts via a user interface with its user peer. The role of the user peer is to search for, download, and display documents. A search is carried out by searching trackers, which are close to the document's id. These trackers reply with references to storage peers, where a user peer can download this document from. If the user peers publish downloaded documents, they will also become storage peers.

Editor Role: An editor peer has modified a document. The editor hands over the change proposal to the mediator peer, which initiates the voting session.

Mediator Role: A mediator peer is responsible for a voting session. After a document is changed, the mediator contacts its voting peers and requests them to review and sign the result. If a majority of those voters approve the change, the modified document is published. In the publishing process, references and the voting meta data are stored on tracker peers.

Voter Role: A voter peer can vote on a modified document. Effective voters are editors of previous modifications of this document or editors with many approved votes. If a modification is accepted, the result is generated and signed.

3.1 Use Case

A typical use case for PeerVote is shown in Figure 1, where a user searches for a document including the keyword "P2P". The peer finds a tracker with addresses of peers holding documents with the keyword “P2P”. After reading, the user finds a wrong statement and corrects it. Following the majority of previous authors agreeing on this correction, the modified document is published.

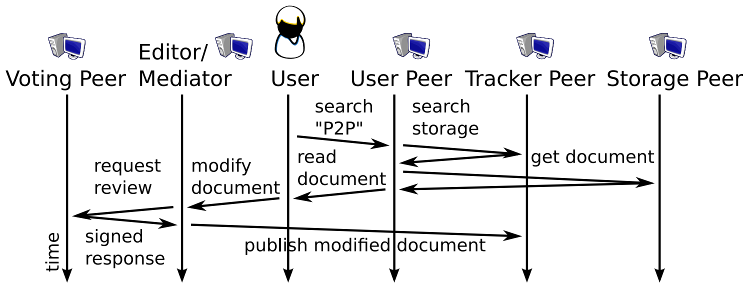

Fig. 1. PeerVote message sequence chart

3.2 Storage Design

The storage of documents can be either direct storage as in CFS [8] or Oceanstore [15], or indirect using a tracker such as in BitTorrent [7]. Direct storage stores the document on peers close to this document’s id, while a tracker-based storage stores a reference on peers close to the document’s id. The advantages of the tracker is that documents are stored on peers that have either edited or viewed that document. Thus, an editor peer in a tracker-based solution is also a storage peer. It can be assumed that in this case an editor has an incentive to store the document, because the user has invested time in this document and wants it to be published. With direct storage, a random peer stores the document with no incentives to store it. A tracker-based solution balances the load by design. The more popular a document is, the more references the tracker has. For direct storage, popular documents need to be identified and those documents need to be replicated on its neighbors, which requires an additional replication mechanism. The advantage of direct storage is that a strategy can be implemented to preserve unpopular documents, which requires an additional replication mechanism in the tracker-based case. Since PeerVote supports both approaches and both approaches have advantages and disadvantages, a decision for one approach has to be made case by case.

3.3 Voting Algorithm and Data Structure

A voting session is initiated, when an editor submits a proposal and becomes a mediator. As a mediator, it starts the voting phase by sending the proposal to all previous editors or to editors with many edits, if no previous editors exist in case of a new document. These editors can vote, if the change shall be accepted. The voting session is open for a specific amount of time. During this time, voters can review and vote for or against the change. If a voter does not vote, it is considered as a negative vote. Thus, there are no benefits of avoiding to contact peers for a review. A vote is accepted, if signed voting results reach a threshold. Figure 2 shows the pseudo code for the voting scheme, with the methods `voting()` and `incoming()`. The voting method is used by a mediator to start a voting session, while the incoming method is called on voting peers (previous editors).

```
//start voting session
voting(Document d, time t) {
  //previous editor are stored in meta data
  MetaData m=d.getMetaData();
  List resultVotes;
  //ask previous editors to review document
  for editor in m.previousEditors() {
    resultVotes.add(requestReview(editor,d));
  }
  //this is blocking, use a thread and a
  //future object to make it non-blocking
  wait(t);
  return resultVotes;
}
//handle incoming voting requests
incoming(Voterequest r, Document d) {
  //display a popup for the user
  notifyUser();
  //ask the user to vote for or against
  boolean vote=displayAndReview(d);
  if(vote) {
    result=sign(d.getMetaData())
    //reply sends the result to the requester
    r.reply(result)
  }
  else {
    //reply sends null to the requester,
    //which means that the modification
    //was not approved
    r.reply(null)
  }
}
```

Fig. 2. Voting scheme pseudo code

```
//verify the meta data before
//adding to storage
addAndVerify(MetaData m1, NodeAddress n) {
  MetaData copyM1 = m1;
  MetaData m2=latestLocalMetaData()
  while (true) {
    //checks if version matches and previous
    //editors are in place
    if (isNextVersion(copyM1, m2)) {
      setLatestVersion(copyM1)
      //reached latest
      if (copyM1.equals(m1)) {
        //verification successful
        return true;
      }
      //start from the beginning
      m2 = copyM1;
      copyM1 = m1;
    }
    else  {
      copyM1 = copyM1.getPreviousMetaData();
      if (copy == null) {
        //reached first versios
        break;
      }
    }
  }
  //verification failed
  return false;
}
```

Fig. 3. Voting meta data verification

Storage and user peers need to evaluate the voting meta data to detect unreviewed changes. The information required for this evaluation is stored in the voting meta data. The voting meta data in PeerVote consists of versioning information, hash code of the content of the document, document identification, and signed voting results. The document identification is constant for all changes of the same document. The voting meta data has the structure as shown in Table 2.

It is important to verify the consistency of previous voting sessions. Otherwise, a malicious peer could replace, add, or drop signatures to pretend to have less editors or to be a previous editor. Thus, voting meta data is verified with the following rules. A new version can have at most one new editor, a new version contains all IDs of previous editors, and all versions have the signed re-

Table 2. Voting meta data structure and example values

Nr	*Version*	*ID*	*Hash*	*Signatures (Signature, Signee)*	*Previous*
#1	1	0x123	0x234	(0x134,0x456)	
#2	2	0x123	0x235	(0x135,0x456) (0x136,0x567)	#1

sults of the voting sessions. These rules are verified as shown in Figure 3, which shows the pseudo code for verification of voting meta data. The voting scheme uses public-key cryptography for peer identification and signatures to prevent forgery. A public key is exchanged on first contact.

In such a decentralized system, peers can modify documents concurrently, because they operate independently. Concurrency in this voting scheme for change proposals uses a first come, first served scheme. This means that for conflicting changes, the peer that collected the votes first, publishes the change proposal and all other conflicting change proposals with the same version number are discarded. The user which submitted the failed change proposal is notified and can update to the latest version of the document and propose the change again.

3.4 Voting Scheme Example

The following example explains the design and the pseudo code, presented in Figure 3 with example values. While the new document (#1) in Table 2 with the id 0x123 and hash value 0x234 has no reference to previous meta data, the modified document (#2) has a reference to (#1), which means that (#2) is based on (#1). For the modified document (#2), the hash value 0x235 changed, because of the modified content. The signature in the new document (#1) is based on the hash value 0x234 to confirm that its content was approved by peer with the id 0x456. This signature belongs to node 0x456 in this example. All signatures of the modified document (#2) show that two voters have approved this modification, the original editor, and a second editor with the id 0x567. The signature for document #2 is based on the hash 0x235.

4 Implementation and Evaluation

PeerVote has been implemented in a P2P collaboration application. This P2P collaboration application is based on TomP2P [2], a Distributed Hash Table (DHT) and tracker implementation.

4.1 P2P Collaboration Application Implementation

Publishing a new document requires first to search for a tracker, which is responsible for storing the voting meta data and the reference to the storage peer. A tracker is responsible for a document, if the node id is close to the id of the document. For new documents, the voting meta data is filled and stored with the reference to the storage peer on the tracker. A new document is accepted without any further validation because it is assumed that a new document is never published by a malicious peer.

Downloading a document starts with searching for trackers with an id close to the id of the document. These trackers contain addresses of nodes that store the document. After those trackers are found, addresses and voting meta data are downloaded. The addresses from the trackers are used to download the document.

After a document is modified, all previous editors are requested to review the change. All voting peers that are online answer this request as described in the incoming method in the pseudo code of Figure 2. If a sufficient number of voters have signed the modified document, the document is published. For publishing a modified document, the tracker verifies voting meta data as described in the `addAndVerify` method (cf. Figure 3). If the verification is successful, the reference to the modified document and voting meta data will be stored. Further search requests to this tracker return the modified meta data and its reference to the storage peer.

4.2 Simulation and Experimental Settings

Simulations and experiments have been run to investigate scalability, fault-tolerance, and robustness. This is performed by using various combinations of the following parameters: number of nodes, number of malicious nodes, number of voting sessions, and number of change proposals. Since user-based voting in P2P collaboration applications have not been studied before, parameters, such as churn (10%), concurrency (50%), and number of publishing peers (10%, 20%, 30%), have been selected by the authors. Malicious peers can propose incorrect modifications, vote against correct modifications, or vote randomly. Malicious peers that vote randomly, simulate users that vote without reviewing, because users may pretend to be active without investing resources for reviewing. Malicious peers that vote against correct proposals simulate users that intend to publish unsuitable information, *e.g.,* spam or contradicting information. In the current setting, the voting meta data is not verified by user peers, because malicious storage peers have not been implemented. New documents do not have wrong information regardless of the publishing peer being malicious. A voting session lasts at most 45 seconds and a peer replies, if online, within 10 seconds to finish the simulation and experiments in a reasonable time. The voting meta data is accumulated during these changes. There is a 50% chance for peers to propose changes concurrently in a voting session. Currently, in the P2P collaboration application changes based on the same version are always conflicting and never merged.

Signatures are required to sign the result of a voting session. The current implementation generates a hash of the modified data as a replacement for the signature, because signing and verifying is CPU intensive and many peers are simulated on one machine, which makes the CPU a bottleneck. Since malicious peers do not exploit this, a signature can be verified by hashing the modified data. Thus, a distribution of public keys is not required.

PeerVote's evaluations are based on more than 1,000 peers. Churn has been set to 10%, which means that peers fail to reply in 10% of the time. Experiments and simulations have been been run 10 times each and the averages have been calculated including the standard deviation.

All simulations have been performed on 2 Intel(R) Xeon(R) quad-core CPUs, 2.83GHz, with a Java HotSpot(TM) 64-Bit Server VM (build 11.2-b01, mixed mode). The parameters set for the simulation with java were -d64 -Xmx10G, which allows to run in 64bit mode and use 10 GB RAM. The simulation required 2 days to complete. All experiments were run on 14 EMANICSLab machines. EMANICSLab is a research network established among European research partners and consists of 7 different partner sites. The hardware used in EMANICSLab is heterogeneous with different CPU models and RAM sizes. The experiment required 2.5 days to complete.

4.3 Simulation Results and Discussion

Figure 4 shows the accumulated number of DHT and voting messages per peer with an increasing number of peers, from 100 to 1000. The number of published documents is set to 10%, 20%, and 30% respectively, proportional to the number of peers, which means that for 20% and 100 peers, 20 documents are published, for 1000 peers, 200 documents are published. A document is changed 2 times. All change proposals are carried out by 30% of the peers. This means that for 100 peers, 30 change proposals are submitted for 20 documents. Thus, some proposals are submitted concurrently. Malicious peers are not present in this simulation. Figure 5 shows the accumulated number of voting messages per peer with increasing number of peers using the aforementioned parameters.

Figure 4 indicates a logarithmic behavior for all three curves. Furthermore, Figure 5 shows a constant number of messages from 100 to 1,000 peers, and that the number of voting messages doubles if the publishing peers double. This indicates that the voting algorithm is scalable with an increasing number of peers. While the voting messages have a small number of messages (maximum of 4.75), the overall number of messages, including DHT overhead have a maximum of 575. Thus, this indicates that the DHT overhead contributes most to the number of messages.

Figure 6 shows the graph for voting traffic with an increasing number of documents, from 10% to 100% proportional to the number of peers. The number of peers shown are 400, 500, and 600. Voting parameters are set as described in the previous simulation. Malicious peers are not present. Figure 7 shows a graph for voting traffic with an increasing number of change proposals with the same settings as for Figure 6.

Both figures show for 400, 500, and 600 peers, that the 3 traffic graphs overlap. This indicates that the voting traffic is independent of the number of peers. The voting traffic is dependent on the number of documents (Figure 6) and the number of change proposals (Figure 7).

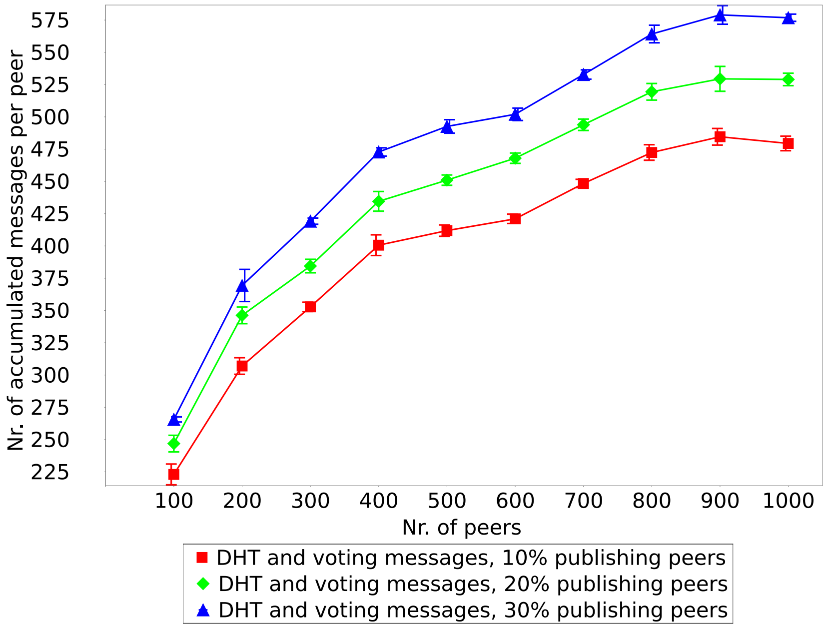

Fig. 4. Number of voting and DHT messages per peer (Simulation)

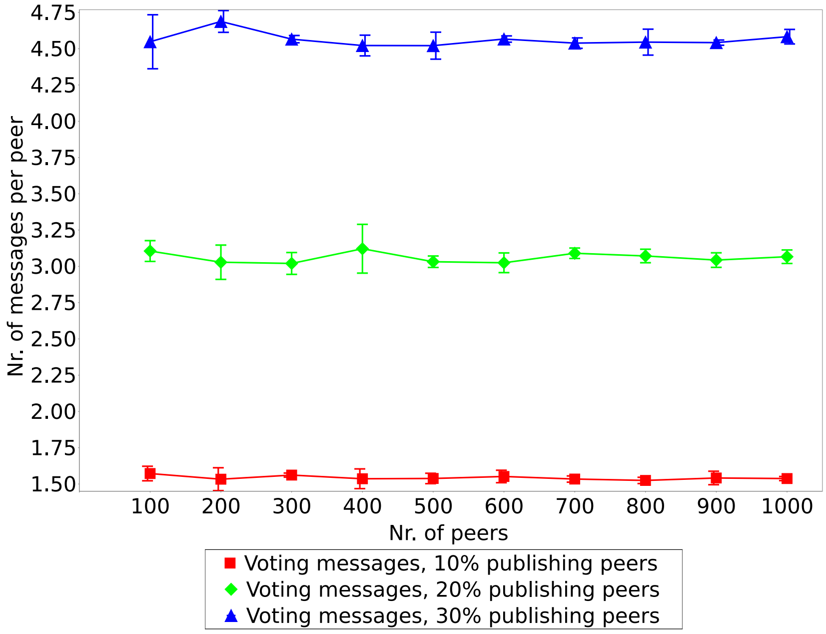

Fig. 5. Number of voting messages per peer (Simulation)

Figure 6 shows that traffic increases with linear complexity for an increasing number of peers proposing a change, while Figure 7 shows that traffic increases with polynomial complexity for an increasing number of change proposals per peer. The graph in Figure 7 is explained due to the increasing number of previous authors, which increase the message size and also the number of messages, because all previous peers need to be contacted. Thus, the list of previous authors for a document needs to have a fixed capacity.

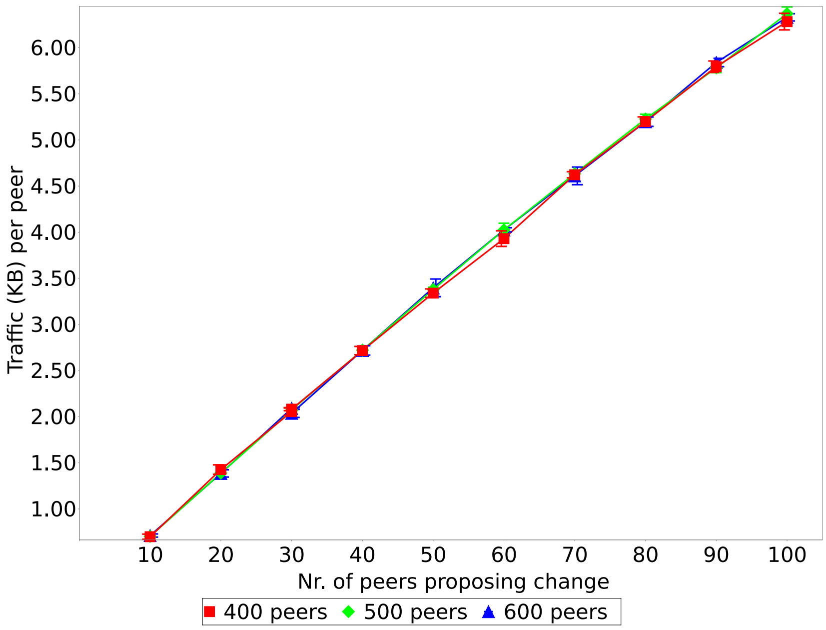

Fig. 6. Voting traffic for increasing document numbers per peer (Simulation)

To evaluate malicious behavior, up to 50% malicious peers have been simulated. Figure 8 shows an increasing number of malicious peers. The number of peers are set to 500. The number of published documents is set to 100. Both graphs are decreasing and start at 100% correct documents. The more malicious peers join, the more incorrect changes are present. The graph with random votes (Figure 8) has less incorrect changes compared to the graph with malicious votes. For 50% of malicious peers (250 peers), 65% correct documents are stored, while for 50% of random voting peers, 82% correct documents are stored. More than half of the documents are correct, because the initial document is always correct, regardless if a peer is malicious or not.

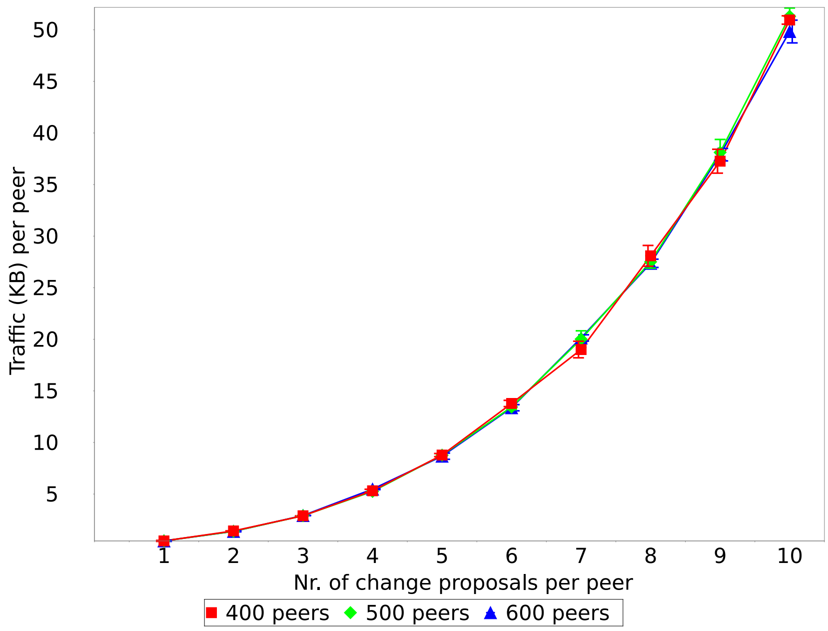

Fig. 7. Voting traffic for increasing change proposals per peer (Simulation)

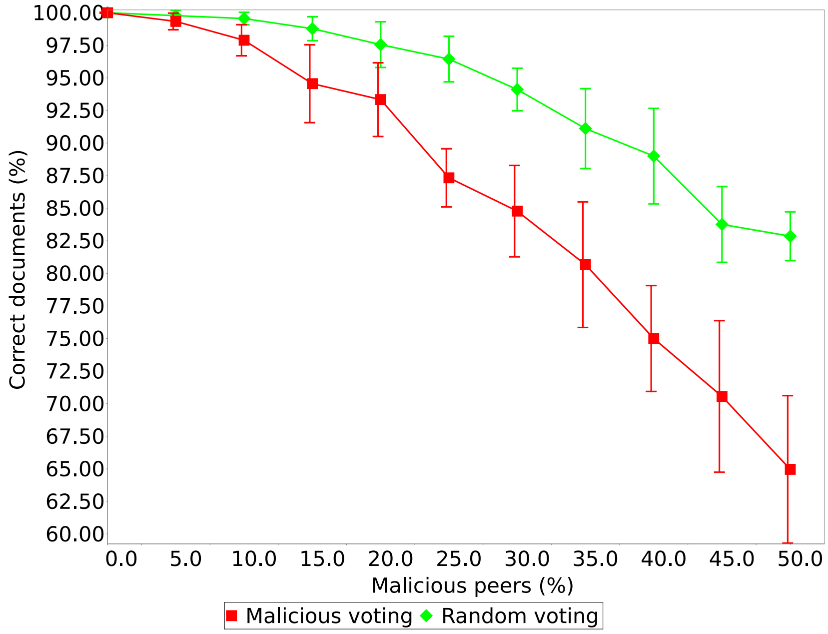

Fig. 8. Effect on documents with malicious and randomly voting peers (Simulation)

4.4 Experimental Results and Discussion

Figure 9 shows the accumulated number of DHT and voting messages per peer with an increasing number of peers, from 294 to 1078, while Figure 10 show the accumulated number of voting messages per peer with the same setting. The number of published documents is set to 20% proportional to the number of peers. The other parameters for the experiments on EMANICSLab are the same as for the simulation. Figure 9 shows the same logarithmic behavior as in the simulation (cf. Figure 4), while Figure 10 shows the same constant behavior as in the simulation (cf. Figure 5). The error bars in both figures are larger as expected due to the real world conditions on EMANICSLab.

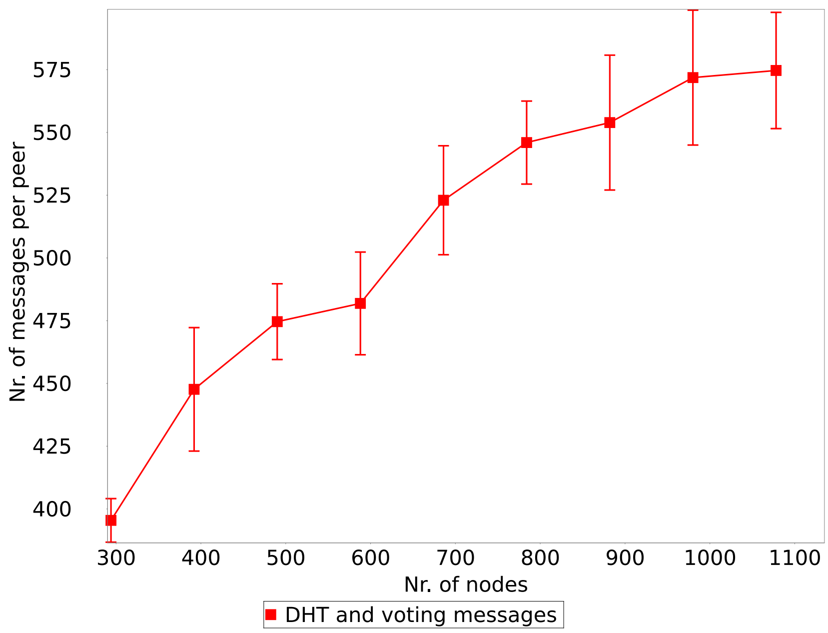

Fig. 9. Number of voting and DHT messages per peer (EMANICSLab)

Figure 11 shows the graph for voting traffic with an increasing number of documents, from 10% to 100% proportional to the number of peers, while Figure 12 shows the graph for voting traffic with an increasing number of change proposals. The number of peers for both experiments is 490. The other parameters for both experiments are the same as for the simulation. Both figures show similar behavior as in the simulation (cf. Figure 6 and Figure 7). This indicates that the algorithm works similar under real world conditions.

Figure 13 shows an increasing number of malicious peers, with up to 50% malicious peers, to evaluate malicious behavior on EMANICSLab. The number of peers are set to 490, while the other parameters for this experiment are the

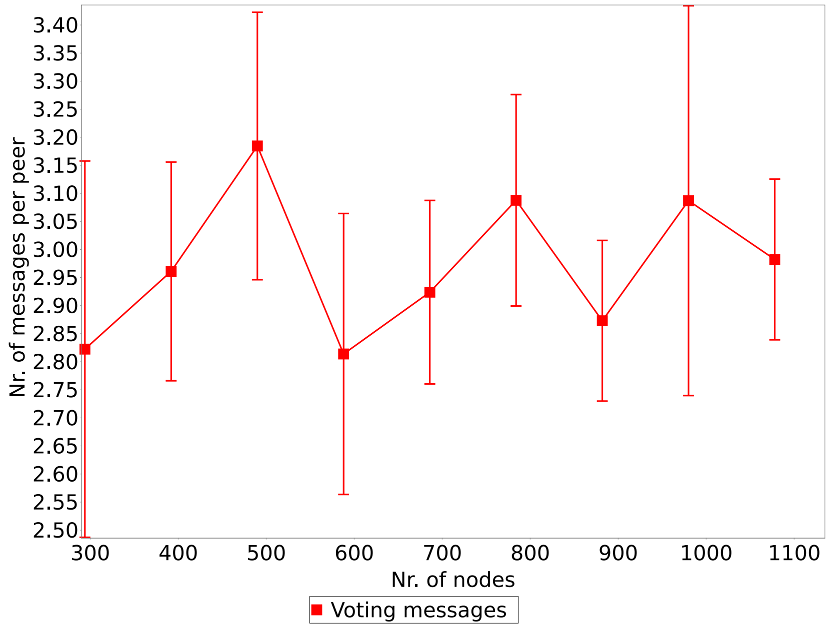

Fig. 10. Number of voting messages per peer (EMANICSLab)

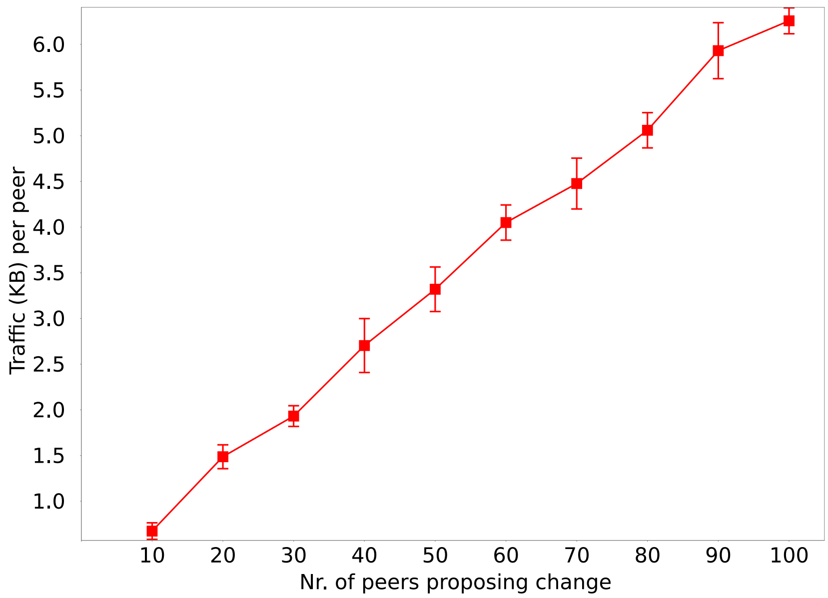

Fig. 11. Voting traffic for increasing document numbers per peer (EMANICSLab)

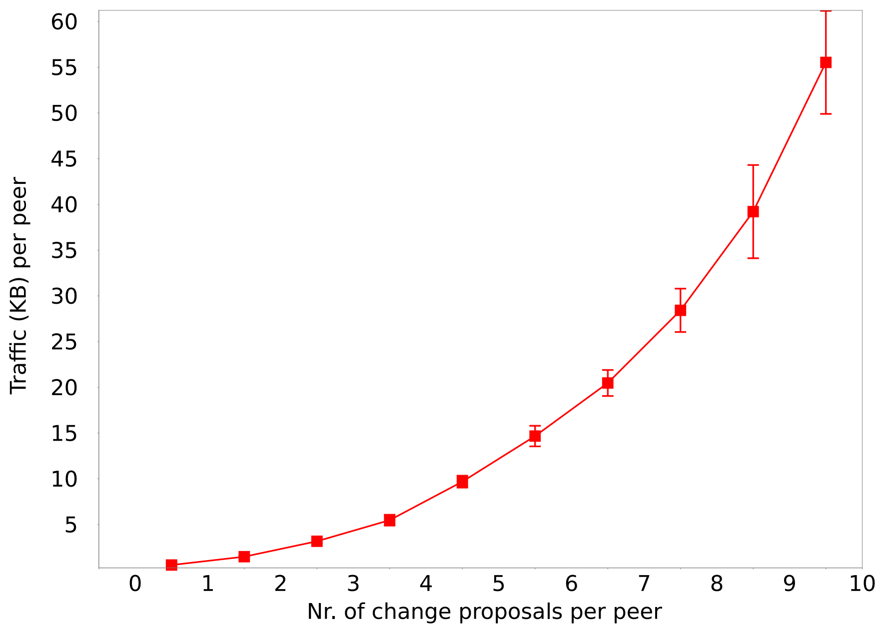

Fig. 12. Voting traffic for increasing change proposals per peer (EMANICSLab)

same as for the simulation. This graph is again similar as the graph in the simulation (cf. Figure 8). The large error bars and the peak at 35% for malicious voters are due to the real world conditions on EMANICSLab.

4.5 Comparison of Experimental and Simulation Results

Figure 14 and Figure 15 show the comparison of those simulation results above with the experimental results from the EMANICSLab implementation. Both settings have from around 300 to less than 1,100 peers and 20% published documents. While Figure 14 shows number of voting and DHT messages per peer, Figure 15 shows the number of voting messages per peer.

The number of messages for the DHT and voting messages is higher on EMANICSLab in Figure 14. The higher message number is due to peers on EMANICSLab, which may fail to reply or which send a reply too late, because of other CPU or network intensive experiments running on EMANICSLab nodes. The larger error bars for EMANICSLab experiments also reflect this. The number of voting messages is lower on EMANICSLab in Figure 15 because message delivery may fail. This has the effect that not all previous peers participate in a voting session and the message number of voting messages is lower in these cases.

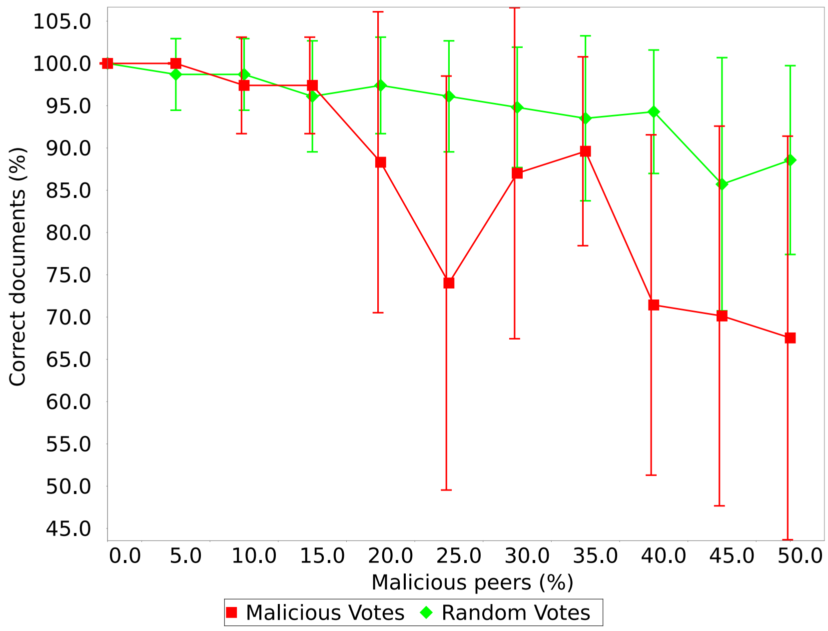

Fig. 13. Effect on documents with malicious and randomly voting peers (EMANICSLab)

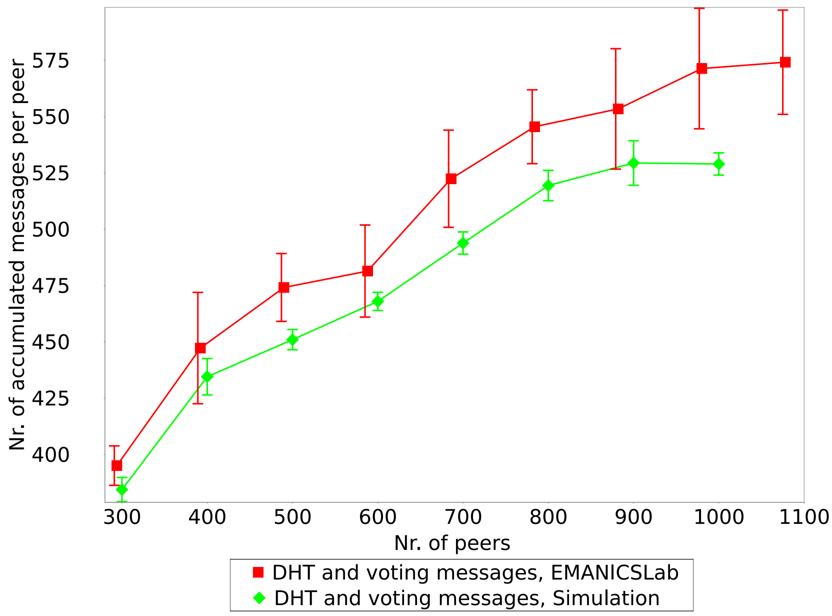

Fig. 14. Experiment on EMANICSLab and simulation comparison for the number of voting and DHT messages per peer

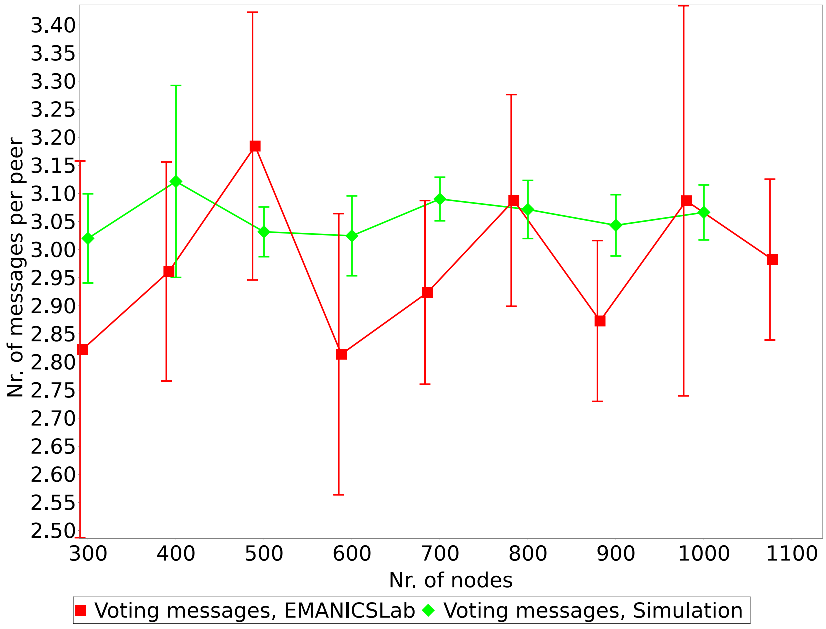

Fig. 15. Experiment on EMANICSLab and simulation comparison for the number of voting messages per peer

5 Summary, Conclusions, and Future Work

This paper presented PeerVote, a decentralized voting mechanism in a P2P collaboration application. Experiments and simulations with a prototypical implementation showed that PeerVote is scalable and robust, even with the presence of random and malicious voting peers. Evaluations showed that overall traffic increases logarithmically and that voting messages are independent on the number of peers. Furthermore, experiments on EMANICSLab showed that the algorithm deployed in a real network has similar traffic characteristics as in the simulation.

In contrast to P2P recommendation systems, which are typically used for recommending complete documents, PeerVote is used for managing changes of a document in a decentralized collaboration application. PeerVote allows previous authors to review changes. However, new documents and early proposals are prone to malicious behavior because only few or none previous editors exist. A possible solution is to combine a recommendation system with PeerVote. A second issue is that authors can block changes forever, if they do not respond to a review request. If for a first change proposal the previous author does not respond, then a majority is never reached. Thus, such threshold needs to be time-based, where old articles need less votes for a change. A third issue is that the list of previous authors grows, resulting in a polynomial traffic graph. This could be solved by limiting the capacity of this list.

The voting mechanism is not Sybil attack [9] proof, since it does not prevent a user from acquiring multiple node identities. Mechanisms to prevent or detect acquisition of multiple identities have to be implemented. Such mechanisms may be implemented with certificates binding peer identifiers to real-world identities, either with a trusted third part or with a web of trust. Another mechanism to limit multiple identities is by exchanging or paying with resources to participate or not be excluded [4].

Future work will investigate how the PeerVote mechanism can be combined with an incentive scheme. With such a combination, each vote can be traded to encourage peers to review changes or to store and provide documents. Further work will also investigate PeerVote in other application domains. In other application domains more voting choices could be offered to voters.

Acknowledgements This work has been performed partially in the framework of the EU IST Project EC-GIN (FP6-2006-IST-045256) as well as the EU IST NoE EMANICS (FP6-2004-IST-026854). Many thanks go to Lennart Svensson, who did initial work on this topic in his Master's thesis [22].

References

1. T. B. Adler and L. de Alfaro. A content-driven reputation system for the wikipedia. In *Proceedings of the 16th international conference on World Wide Web (WWW '07)*, pages 261–270, New York, NY, USA, 2007.
2. T. Bocek. TomP2P - A Distributed Multi Map. http://www.csg.uzh.ch/publications/software/TomP2P, 2009.
3. T. Bocek and B. Stiller. Peer-to-Peer Large-scale Collaborative Storage Networks. In *Proceedings of the 1st International conference on Adaptive Infrastructure, Management and Security (AIMS 2007)*, Oslo, Norway, 2007.
4. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Operating Systems Review*, 36(SI):299–314, 2002.
5. U. Cetintemel and P. J. Keleher. Light-Weight Currency Management Mechanisms in Deno. In *Proceedings of the 10th International Workshop on Research Issues in Data Engineering (RIDE)*, pages 17–24, San Diego, CA, USA, February 2000.
6. D. Chaum. Blind signature system. In *CRYPTO'83: Advances in Cryptology*, page 153, New York, USA, 1983.
7. B. Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, Berkeley, CA, USA, June 2003.
8. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, Chateau Lake Louise, Banff, Canada, October 2001.
9. J. R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, pages 251–260, London, UK, 2002. Springer-Verlag.
10. EMANICSLab. http://emanicslab.csg.uzh.ch/.
11. D. K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP)*, pages 150–162, 1979.

12. J. Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481, London, UK, 1978. Springer-Verlag.
13. B. Hardekopf, K. Kwiat, and Upadhyaya. Secure and Fault-Tolerant Voting in Distributed Systems. In *Proceedings of the 2001 IEEE Aerospace Conference*, volume 3, Big Sky, Montana, USA, March 2001.
14. T. Korsgaard and C. Jensen. Reengineering the Wikipedia for Reputation. In *Proceedings of the 4th International Workshop on Security and Trust Management (STM 08)*, pages 71–84, Trondheim, Norway, June 2008.
15. J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of the 9th international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Cambridge, MA, USA, November 2000.
16. P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, London, UK, March 2002.
17. J. C. Morris and C. Lüer. DistriWiki: A Distributed Peer-to-Peer Wiki. In *Proceedings of the 2007 International Symposium on Wikis (WikiSym '07)*, pages 69–74, Montreal, Quebec, Canada, October 2007.
18. P. Mukherjee, C. Leng, and A. Schürr. Piki - A Peer-to-Peer based Wiki Engine. In *Proceedings of the 2008 8th International Conference on Peer-to-Peer Computing (P2P '08)*, pages 185–186, Washington, DC, USA, September 2008.
19. M. Rodrig and A. LaMarca. Decentralized weighted voting for P2P data management. In *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access (MobiDe '03)*, pages 85–92, San Diego, CA, USA, 2003.
20. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.
21. A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.
22. L. Svensson. Decentralized Secure and Incentive-compatible Voting In P2P Networks. Master's thesis, Communication Systems Group, IFI, University of Zurich, Switzerland, March 2007.
23. G. Urdaneta, G. Pierre, and M. van Steen. A Decentralized Wiki Engine for Collaborative Wikipedia Hosting. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies (WEBIST)*, Barcelona, Spain, March 2007.
24. B. Warner, Z. Wilcox-O'Hearn, and R. Kinninmont. Tahoe: A Secure Distributed Filesystem. http://allmydata.org/ warner/pycon-tahoe.html, March 2008.
25. S. Weiss, P. Urso, and P. Molli. Wooki: A P2P Wiki-Based Collaborative Writing Tool. In *Proceedings of the 8th International Conference on Web Information Systems Engineering (WISE)*, pages 503–512, Nancy, France, 2007.
26. Wikipedia. http://www.wikipedia.org.
27. Wuala, your files online. http://wua.la.
28. Zattoo — TV meets PC. http://zattoo.com/.