

PeerMint: Decentralized and Secure Accounting for Peer-to-Peer Applications

David Hausheer¹, Burkhard Stiller^{2,1}

¹ Computer Engineering and Networks Laboratory TIK

Swiss Federal Institute of Technology, ETH Zurich, Switzerland

² Department of Informatics IFI, University of Zurich, Switzerland

{hausheer, stiller}@tik.ee.ethz.ch

Abstract. P2P-based applications like file-sharing or distributed storage benefit from the scalability and performance of completely decentralized P2P infrastructures. However, existing P2P infrastructures like Chord or Pastry are vulnerable against selfish and malicious behavior and provide currently little support for commercial applications. There is a need for reliable mechanisms that enable the commercial use of P2P technology, while maintaining favorable scalability properties. PeerMint is a completely decentralized and secure accounting scheme which facilitates market-based management of P2P applications. The scheme applies a structured P2P overlay network to keep accounting information in an efficient and reliable way. Session mediation peers are used to minimize the impact of collusion among peers. A prototype has been implemented as part of a modular Accounting and Charging system to show PeerMint's practical applicability. Experiments were performed to provide evidence of the scheme's scalability and reliability.

1 Introduction

Emerging peer-to-peer (P2P) applications benefit from the large amount of resources provided by many individual peers. Using sophisticated techniques for aggregation and replication of these resources, P2P-based systems are able to provide a much higher robustness and performance than traditional client/server-based applications. For example, file-sharing applications like eMule [8] or BitTorrent [5] are able to provide access to huge amount of content in a reliable way. At the same time, an increasing number of applications make use of basic P2P network infrastructures like Chord [20] or Pastry [17] and benefit from the good scalability properties of these systems.

However, many existing P2P infrastructures and applications suffer from peers which behave in a selfish or malicious fashion [9], [18]. Moreover, there is currently little support for commercial applications for which appropriate accounting and charging mechanisms are required. Compared to centralized systems, accounting of resource usage is much more complex in a distributed environment and misuse of such mechanisms is more difficult to prevent.

PeerMint is a decentralized accounting scheme that provides the ability to keep track of contribution and consumption of resources by peers. The scheme uses remote peers to store and aggregate accounting information in a trustworthy and scalable way. The aggregated accounting information can be used to enforce fair sharing of resources

between peers or as a basis for additional charging and payment mechanisms. A structured P2P overlay network is applied to map accounts onto a redundant set of peers and organize them in an efficient and scalable manner. Other than similar work (cf. [1], [15], [16], [22]), the proposed scheme uses session mediation peers to maintain session information about transactions between peers. This minimizes the impact of colluding peers trying to increase their account balance without actually contributing resources. Additionally, PeerMint provides economic flexibility by supporting the use of different types of tariffs. The proposed scheme is secure in that it ensures the availability and integrity of the accounting data. However, it does not provide confidentiality or privacy, as every peer is, in principle, able to access the accounting data of any other peer.

PeerMint has been designed and implemented as an accounting mechanism within a generic and modular Accounting and Charging (A&C) system for P2P applications [11]. The system separates generic accounting functionality such as session maintenance and account configuration from individual underlying accounting mechanisms. This enables the implementation of alternative accounting schemes with different properties, independently from existing core functionality. As the interface to the accounting mechanism is very generic, the proposed scheme could also be used in other environments.

The remainder of this document is structured as follows: Section 2 presents the main principles and requirements for an accounting mechanism for P2P applications. Section 3 introduces the concept of remote accounting that is adopted in PeerMint and gives an overview on the potential design space. A detailed description about the design of PeerMint on top of Pastry is given in Section 4, while Section 5 presents its implementation within the Accounting and Charging system mentioned. In addition, the scheme is evaluated with respect to scalability and robustness. Finally, Section 6 concludes the paper and discusses some open issues.

2 Requirements and Definitions

The following describes the core principles which underlie the accounting scheme proposed in this paper. The main goal of an accounting mechanisms is to ensure *accountability* [7] by providing a set of functionality that enables to account for the use of services or resources offered within a particular P2P application. As such it gives peers an incentive to contribute their own resources and serves as a basis to punish selfish behavior like free-riding. Vital accounting mechanisms are the processing of accounting events describing the amount of used resources, the application of respective tariffs, as well as the creation and maintenance of accounts to store and aggregate the accounting information and to keep track of the account balance. Accounting schemes may implement any specific type of accounting, from simple local or centralized accounting to more sophisticated remote or token-based accounting [10]. Individual accounting schemes usually fulfill specific requirements with respect to efficiency, scalability, and economic flexibility, as well as security and trustworthiness, among which there is always a trade-off. While, *e.g.*, local accounting scales perfectly with $O(1)$, it features very bad security properties, as a peer can easily modify its account locally. Such a scheme would therefore only be suitable in trusted environments or for purposes where

security is not important. Specific accounting mechanisms used in existing P2P applications are, *e.g.*, BitTorrent's tit-for-tat mechanism [5] or eMule's credit system [8]. Other related accounting mechanisms that have been developed for use in P2P systems are, *e.g.*, Karma [22], PPay [23], Mojo Nation [14] or the approach presented in [2]. Similar mechanisms have been developed for Grid computing [3], [21].

The main terms and concepts used to describe the proposed mechanism are introduced in the following. The term *session* refers to the use of a particular service or resource, *e.g.*, the download of a file or the use of some amount of computing power. A session has always two session partners, a provider peer and a consumer peer. Furthermore, *accounts* are repositories which can be used to keep and aggregate accounting information, *e.g.*, the amount of MBs uploaded or downloaded or the number of CPU cycles used. Two types of accounts are distinguished, *session accounts* and *peer accounts*. While session accounts are used to keep accounting information within a particular session, peer accounts aggregate information from several sessions, *e.g.*, the total amount of MBs uploaded and downloaded by a particular peer. Peer accounts may also be used to keep information about a peer's reputation or trustworthiness.

For every session there is a corresponding *tariff*. Its main purpose is to specify how service usage needs to be accounted for. As such it is used to process *accounting events* which are generated by the service instances running on both the provider and the consumer side of a session. A tariff is represented by a specific *tariff formula* and a set of *tariff parameters* which are previously agreed upon by both session partners. For example, a volume-based tariff could be used to account for a file download. A tariff may also contain further parameters that have to be computed dynamically during a session, *e.g.*, depending on the time of day or the balance of a particular account. Based on the result from a tariff evaluation a generic *balance update* is created and forwarded to a particular account. Note that the term balance update is used rather than *charge* to make clear that this does not necessarily imply a monetary payment. Balance updates are handled as accounting events and can thus be processed further. It depends on the applied tariff when and by how much the balance of a particular session or peer account is updated. Using tariffs, a variety of settlement schemes can be supported. For example, a peer account may be updated at the beginning or at the end of a session, which can be used to implement a pre-payment or post-payment scheme. In addition, more fine-grained updates of peer accounts are possible during a session, *e.g.*, after a particular amount of the service has been provided or a certain time has passed.

3 Remote Accounting Concept

This section introduces the basic concept of *remote accounting* as it is used by Peer-Mint. In general, remote accounting is based on the idea that accounts are held remotely on other peers. Remote peers are third-party peers, which are typically different from the peers currently providing or using a particular service that needs to be accounted for. Using remote accounting, accounts can be distributed and replicated over several peers, which, if designed appropriately, can increase the reliability and availability of the accounting data. In addition, a higher credibility or trustworthiness can be achieved, when many peers are involved in doing the accounting.

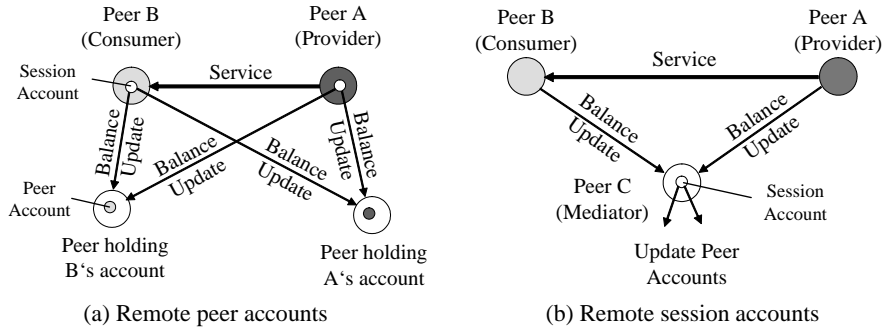


Fig. 1. Remote accounting examples

The example depicted in Figure 1(a) illustrates the basic concept of remote accounting. The scenario demonstrates two peers involved in a session. Both peers hold a local session account, while their peer accounts are held on two different remote peers. Whenever tariff evaluation results in a balance update affecting a peer account (*e.g.*, after a session was terminated), balance updates are forwarded to both peers holding the respective accounts. The remote peers collect the balance updates from both session partners. If both peers agree on the balance update (*i.e.* by sending equal balance updates), the peer accounts are updated accordingly. If, for any reason, the two peers disagree, the peer accounts would not be changed.

Unfortunately, the described accounting scenario faces a collusion problem. The two session partners could forward balance updates only to the peer holding the provider's account. Thus, the provider's account would be updated, while the consumer's account remained unchanged. To prevent this, peer account holders would need to interact with each other to decide whether the accounts need to be updated or not, which highly increases the accounting overhead.

The slightly different concept which underlies PeerMint is depicted in Figure 1(b). In this scenario a remote peer (mediator peer) is holding the session account for a particular service session on behalf of the respective provider and consumer peers. Both peers send their balance updates to the mediator peer which updates the session account accordingly. An ongoing session may immediately be terminated if the session partners disagree. In contrast to the previous scenario, the peer accounts which are typically held by other remote peers can only be updated by the mediator peer.

3.1 Design Space for Remote Accounting

The concept of remote accounting is very general and covers several potential subtypes. An overview on possible variants of remote accounting is given below.

Central Accounting. This is the simplest form of remote accounting and is only mentioned for completeness. Using this type of accounting, accounts would be kept on a centralized place, *e.g.*, on a large database residing on a central server. Central accounting is simple to maintain and control, and is usually highly trusted. However, the goal is to avoid any central elements in the network, as they represent a single point of failure and do not scale for a large number of peers.

Hybrid Accounting. Hybrid accounting features the simplicity of central accounting, while being more scalable with respect to the number of peers. In hybrid accounting a dedicated set of peers (so-called super-peers) are used as account holders. Super-peers are typically peers which are highly trusted by a group of peers (clients) attached to them. If the size of such a group is limited, the hybrid approach scales quite well. However, appropriate incentives need to be given to super-peers, to provide the extra accounting efforts. For instance, every peer may periodically pay a flat fee to its super-peer covering the costs for keeping and updating the accounting data.

Distributed Accounting. Distributed or decentralized accounting seems to be most promising approach for P2P applications, as it completely distributes the accounting load over all peers. Moreover, since all peers are equally involved in doing the accounting the scheme scales very well and no payments are necessary to compensate for any accounting costs. An important design dimension within distributed accounting is the redundancy of accounts. *Non-redundant* accounting describes the case, where every account is held by only one peer, while *redundant* accounting refers to accounts being replicated over several peers.

In the following, central and hybrid accounting will not further be investigated. Instead, the focus will be put on distributed accounting as adopted by PeerMint. The distributed redundant accounting case is discussed more detailed in the following section.

4 PeerMint Design

A non-redundant accounting approach as described in the previous section supersedes the need for any synchronization between accounts, however, it has some severe drawbacks. If for any reason a particular peer goes offline, accounts held by that peer would temporarily not be accessible anymore. Even worse, if a peer completely withdraws from the network, the corresponding accounting data would permanently be lost. Moreover, a malicious peer could easily modify and misreport the balance of an account it is responsible for.

Therefore, it is reasonable to introduce redundancy (cf. [19]), i.e. to replicate accounts over several peers to increase the robustness of the distributed scheme. Figure 2 illustrates the case of distributed redundant accounting as it is used in PeerMint. Both session and peer accounts are held by several independent peers. Provider and consumer peers involved in a session send their balance updates to a redundant set of m session mediation peers which are responsible for holding the session account for the current session (Phase 1). Each session mediation peer then checks if the two peers agree and updates the session account accordingly. Whenever a session account triggers a peer account update, the mediation peers send a balance update to the $2x p$ peers holding the respective peer accounts (Phase 2). The two phases may be repeated several times independently. To overcome byzantine failures (cf. [12]), the resulting account balance is agreed upon using majority decisions. Only if the majority of mediation peers report the same balance update, the peer accounts will be updated. Whenever a peer goes offline or permanently withdraws from the P2P network a new peer takes over its task. The new peer (shown as dashed circle) obtains the current balance from the other account holders.

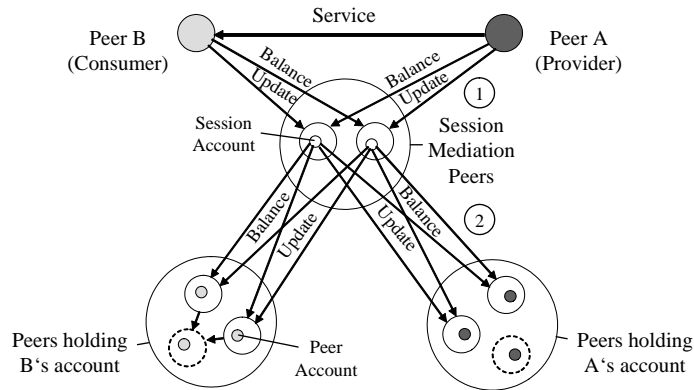


Fig. 2. Distributed redundant accounting in PeerMint

Implementing such an accounting scheme in an efficient and secure way is a complex task. Major difficulties are the mapping of accounts onto peers as well as necessary account maintenance and synchronization activities. These aspects will be addressed more detailed in the following.

4.1 Underlying Infrastructure

To map accounts onto peers, PeerMint applies a structured P2P overlay network. The scheme has been implemented on top of Pastry [17] for which an open source implementation (FreePastry) is available. However, in principle any other infrastructure (*e.g.* Chord [20]) could have been applied. This underlying infrastructure is used whenever a peer joins and leaves the network that interconnects all peers involved in PeerMint's accounting mechanism. It is assumed that all peers possess a public/private key pair which is used for peer identification and signing messages. The peers' public keys are certified by a trusted third party, which guarantees that a peer can only acquire a limited number of identities. The keys are certified offline, *i.e.* prior to joining the network, thus the certification process does not affect the performance of the accounting mechanism itself. An alternative method to create secure keys in a distributed way which may be adopted in future is presented in [4].

Every peer is assigned a unique 128-bit peer ID, which is calculated from the peer's public key using a secure hash function. Pastry provides an efficient prefix-based routing mechanism to find other peers in $O(\log_b(N))$ hops, where N is the number of peers in the overlay network. Every node has a routing table with $O(\log_b(N))$ rows which is continuously updated as peers join or leave the network. A number of n peers (called leaf-set) which are numerically closest to the current peer are part of this routing table. Similar to Karma [22], leaf-sets are used in PeerMint to map accounts onto peers as described in the following.

4.2 Scheme Configuration

Every peer that participates in PeerMint's accounting mechanism (*i.e.* typically all peers in a particular application that uses PeerMint as underlying accounting scheme) needs to configure an instance of the scheme locally. The scheme configuration speci-

fies the mapping function which is used to map accounts onto peers. It is important that all peers within a particular application use the same mapping function. Currently, the same hash function is applied as for calculating the peer ID, i.e. for peer accounts the hash value of the peer ID is used as key, while for session accounts a unique session key (session ID) is calculated by hashing the peer IDs of the two session partners combined with an additional timestamp. For every key there is a peer (root node) which is numerically closest to that key. The root node's leaf-set is used to hold the respective account.

Apart from the mapping rule, the peer ID of any other peer needs to be given to PeerMint which is used as bootstrap node to join an existing overlay network. Otherwise a new network is created. Also the number of redundant peers used as account holders can be configured. By default all peers in a leaf-set are used, but it is also possible to use a subset of the leaf-set or to extend the account holder set beyond the size of a leaf-set.

4.3 Account Creation and Setup

A new peer account is created when a peer joins PeerMint's accounting scheme for the first time. The peer contacts the responsible root node using Pastry's routing mechanism. Pastry routing is only used once in the beginning. All subsequent messages are directly sent over the underlying IP network. Every message exchanged between peers is signed by the sender's private key. The root node notifies all peers in the account holder set about the new peer account and sends their node handles (i.e. peer ID, IP address, and port number) back to the new peer. Peer accounts have an initial account balance, which is typically set to zero. However, a new account is created only, if none does exist so far for the same peer ID. A peer can also remove its account, if the account balance is positive. After a certain time of inactivity, a peer account is removed automatically. There is a fallback mechanism to create an account in the presence of a malicious root node. In this case, a peer tries to find another peer in the same leaf-set by recursively contacting peers on the path to the root node. This peer then temporarily takes over the role of the root node and notifies the corresponding peers. A peer can check if it is responsible for a particular account by verifying if the key of the account falls in the ID range of its own leaf-set (a different ID range applies if the account holder set is greater or smaller than the leaf-set).

Session accounts are created in a similar way. Before the session starts, the two session partners create an SLA which contains the tariff, the two peer IDs, and the session ID. The SLA is signed by both provider and consumer peer and sent to the corresponding root node responsible for the session account. The root node then forwards the SLA to all peers responsible for the new session account and again sends their node handles back to both session partners.

4.4 Account Balance Updates

Once all necessary accounts are created and set up, a session can start. An example for a session between two peers is given in Figure 3. The figure shows all peers involved in the session and the balance updates exchanged between them as described earlier. The two phases indicated correspond to those shown in Figure 2. The session partners regularly send balance updates to the corresponding session mediation peers (Phase 1).

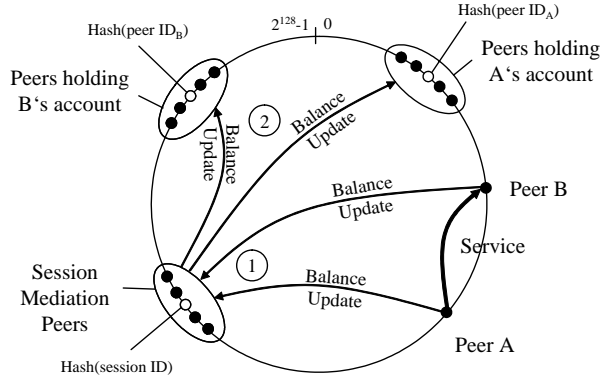


Fig. 3. Example session in PeerMint

The session mediators aggregate them and generate peer account updates as specified in the tariff. To be able to forward them to the peer accounts (Phase 2), the corresponding peer account holders are identified using Pastry routing. The signed SLA containing the session ID is used by the session mediation peers to prove that they are eligible to update the account.

4.5 Account Holder Set Maintenance

Peers may continuously join and leave PeerMint's overlay network. Whenever a new peer joins the network, Pastry notifies the corresponding instances of PeerMint that the leaf-set has changed. Subsequently, the new peer obtains the balance of all accounts it is responsible for. Based on this information, the new peer locally creates instances of these accounts and sets their balance based on the majority of peers reporting the same balance. If no majority decision can be taken (*e.g.*, because the account is currently being updated, and therefore too many peers report different values) the new peer retries to request the account balance until a consistent value is reported. Finally, if the new peer becomes involved in an ongoing session, it notifies itself to the session partners.

Similarly, when a particular peer goes offline (*i.e.* does not respond anymore within a certain time), Pastry notifies the corresponding peers about the change in the leaf-set. This means that other peers will become responsible for the accounts that were held by the peer which went offline. These peers obtain the current account balance in a similar way like a new peer joining the overlay.

A peer which does not reply on synchronization requests or account queries is considered as being offline. Whenever a peer goes offline, all peers within its leaf-set send a balance update to the leaving peer's account holders to decrease its reputation. This serves as an incentive for peers to stay online and behave correctly.

4.6 Account Queries

Any peer may query the balance of a particular peer account, *e.g.*, to check whether a peer which requests a service is credit-worthy. The same procedure as described in Section 4.3 is used to find the responsible peers holding the account. In contrast to peer accounts, session accounts can only be queried by corresponding session mediation peers

and the two peers involved in the session. However, there is no guarantee for the privacy of these accounts, i.e. it may be possible that an unauthorised peer is able to obtain the balance of a session account.

5 Implementation and Evaluation

The proposed scheme has been implemented in Java as part of an Accounting and Charging (A&C) system developed within the MMAPPS project [11], [13]. As such it implements a generic accounting scheme interface which is described in the following:

The first method of this interface, *configureScheme*, is used to initialize the scheme with vital information like the local peer's peer ID and further scheme-specific parameters such as the number of redundant peers used for the accounts. The method *createPeerAccount* creates a new peer account for the local peer. The same method can be used to create an account for the peer's reputation. The *createSessionAccount* method notifies the session mediation peers about a new session and hands over the corresponding SLA. Finally, *notifyBalanceUpdate* and *queryAccount* are used to update and query a particular account, respectively.

PeerMint uses the common API [6] to interact with its local instance of FreePastry, which is used as P2P communication infrastructure. As such it implements the three main methods *forward*, *deliver*, and *update*, through which a FreePastry node notifies forwarded and received messages, and changes in its leaf-set. All interactions between instances of PeerMint are completely encapsulated in Pastry messages, which are either routed over the overlay network (account holder lookup) or sent directly to the corresponding destination node (all other tasks). For each task there is a dedicated message. Based on their type, the messages are dispatched remotely and the corresponding methods are executed. Whenever a reply message is expected, e.g., containing a returned account balance, a new thread is created. The corresponding thread is started by the *WaitingThreadList* class, when the expected message has arrived or after a certain time-out has passed. In the latter case, the corresponding message is sent again.

PeerMint has been evaluated both analytically and through experiments with the prototype in respect of overhead, scalability, and reliability (i.e. resistance against malicious or faulty peers). The overhead and scalability of PeerMint was assessed by analyzing the storage space that is needed to keep the accounting data as well as the number of messages being exchanged between peers. As mentioned in Section 4.1, there is a basic overhead of $O(\log_b(N))$ for maintaining the routing table and sending messages over Pastry (cf. [17] for a detailed analysis of Pastry's efficiency and scalability).

5.1 Analytical Evaluation

Apart from the overhead to maintain the underlying infrastructure, the following effort is needed to operate PeerMint's accounting scheme. Recall that p is the number of account holders per peer account, and m is the number of mediation peers per session account. In addition, f describes the average fraction of peers being currently online, and s is the average number of ongoing sessions. The overhead of PeerMint is shown in Table 1. As it can be seen, the overall effort is moderate and often constant in relation to N . The highest effort is needed to update peer accounts. However, these updates are usu-

ally much less frequent than session account updates. With respect to the scalability, one can see that only account holder look-ups, needed for account creation and peer account updates or queries, depend on the size of the network.

Table 1. Overhead of PeerMint

Costs	Cost driver	Peer Accounts	Session Accounts
Message costs	Account creation	$p + O(\log_b(N))$	$m + 1 + O(\log_b(N))$
	Account update	$2mp + O(\log_b(N))$	$2m$
	Account synchronization	$p - 1$	$m - 1$
	Account query	$2p + O(\log_b(N))$	$2m$
Storage costs	Avg. #accounts per peer	p / f	ms / f

Reliability denotes the ability of a scheme to perform correctly in the presence of malicious and unreliable peers. In PeerMint reliability is achieved using redundant set of peers as account holders. The size of an account holder set, p or m can be adjusted based on the fraction of malicious or faulty peers in the network. PeerMint has a statistically guaranteed reliability, if the number of account holders is higher than $3r + 1$, where r is the number of malicious or faulty nodes in an account holder set. This is the optimum that can be achieved [12].

5.2 Experimental Results

To verify and complement the analytical results, the message overhead and reliability of PeerMint has been measured in a set of simulation experiments with the implemented prototype. The experiments were run with up to 1000 peers on a testbed of four Pentium 4 CPUs, 1.8 - 2.4 GHz, with 512 MB RAM using Java VM 1.4.2. In all experiments performed the number of sessions s was set to 2000. For each session a consumer and provider were assigned randomly, and each account was updated once per session.

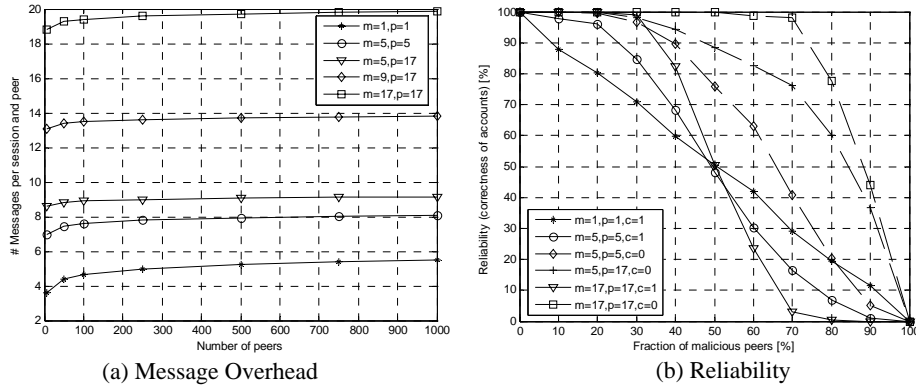


Fig. 4. Experimental results

Figure 4(a) shows the number of messages per session and peer for a varying number of peers N in the network. The size of all messages is around 1kB. It can be seen, that the messages overhead increases slowly in small networks, but levels off

when the network becomes larger. Thus, the system scales very well with the size of the network.

Figure 4(b) shows the reliability of PeerMint (measured in the amount of accounts held correctly) for a selected number of account holders m and p and the fraction of malicious peers in the network. Malicious peers were modeled as account holders which reported consistent ($c=1$) or arbitrary ($c=0$) false values (c relates to the amount of collusion among malicious peers). As it can be seen, PeerMint can correctly keep accounts with up to 30% malicious colluding peers by using 17 peer account holders and 17 session mediators. Thus, the increase of the number of account holders results in a high reliability. Without collusion, reliability can even be achieved with as many as 50% or more malicious peers.

6 Conclusions

Accountability is the key for a success of P2P systems. Based on PeerMint, the accounting scheme presented in this paper, it is possible to provide accountability for P2P applications in a secure and scalable manner. The paper described, how trustworthiness and resilience of accounting data can be achieved in the presence of malicious or faulty nodes, using redundant sets of independent peers. Implemented on top of an existing P2P infrastructure, PeerMint provides scalable accounting functionality at a moderate overhead. Its integration into a modular and generic Accounting and Charging system enables the flexible use of PeerMint for a variety of P2P applications. In future work, the presented accounting scheme will further be optimized and more extensively analyzed within real world environments.

Acknowledgements. This work has been performed partially in the framework of the EU IST project MMAPPS “Market Management of Peer-to-Peer Services” (IST-2001-34201), where the ETH Zürich has been funded by the Swiss Bundesministerium für Bildung und Wissenschaft BBW, Bern, under Grant No. 00.0275. Additionally, the authors would like to acknowledge discussions with all of their colleagues and project partners.

References

1. K. Aberer, Z. Despotovic: *Managing Trust in a Peer-2-Peer Information System*; Tenth International Conference on Information and Knowledge Management (CIKM'01), Atlanta, Georgia, USA, 2001.
2. A. Agrawal, D. Brown, A. Ojha, S. Savage: *Towards Bucking free-riders: Distributed Accounting and Settlement in Peer-to-Peer Networks*; Jacob School of Engineering Research Review, UCSD, February 2003.
3. A. Barmouta, R. Buyya: *GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration*; 17th Annual International Parallel & Distributed Processing Symposium (IPDPS 2003) Workshop on Internet Computing and E-Commerce, Nice, France, April 2003.
4. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach: *Security for structured peer-to-peer overlay networks*; Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, MA, USA, December 2002.

5. B. Cohen: *Incentives Build Robustness in BitTorrent*; Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA, June 2003.
6. F. Dabek, P. Druschel, B. Zhao, J. Kubiatowicz, I. Stoica: *Towards a Common API for Structured Peer-to-Peer Overlays*; 2nd International Workshop on Peer-to-Peer Systems (IPTP'03), Berkeley, CA, February 2003.
7. R. Dingledine, M. Freedman, D. Molnar: *Accountability*; In Peer-To-Peer: Harnessing the Power of Disruptive Technologies, O'Reilly & Associates, Chapter 16, pp. 217 - 340, 1st edition, March 2001.
8. The eMule Project: <http://www.emule-project.net/>, October 2004.
9. P. Golle, K. Leyton-Brown, I. Mironov and M. Lillibridge: *Incentives for Sharing in Peer-to-Peer Networks*; 3rd ACM Conference on Electronic Commerce, Tampa, Florida, USA, October 2001.
10. D. Hausheer, N. Liebau, A. Mauthe, R. Steinmetz, B. Stiller: *Token-based Accounting and Distributed Pricing to Introduce Market Mechanisms in a Peer-to-Peer File Sharing Scenario*; 3rd IEEE Conference on Peer-to-Peer Computing, Linköping, Sweden, September 2003.
11. D. Hausheer, J. Gerke, B. Stiller: *A Generic and Modular Accounting and Charging System for Peer-to-Peer Applications*; 14. Fachtagung Kommunikation in Verteilten Systemen 2005 (KiVS 05), Kaiserslautern, Germany, February 2005.
12. L. Lamport, R. Shostak, M. Pease: *The Byzantine Generals Problem*; ACM Transactions on Programming Languages and Systems, Vol. 4, pp. 382-401, July 1982.
13. MMAPPS: *Market Management of Peer-to-peer Services*; EU Project, <http://www.mmapps.org/>.
14. Mojo Nation: *Technical Overview*; http://www.mojonation.net/docs/technical_overview.shtml, January 2002.
15. T. Ngan, D. Wallach, P. Druschel: *Enforcing fair sharing of peer-to-peer resources*; 2nd International Workshop on P2P Systems (IPTPS), Berkeley, CA, USA, February 2003.
16. N. Ntamos, P. Triantafillou: *SeAl: Managing Accesses and Data in Peer-to-Peer Sharing Networks*; In Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P 2004), Zurich, Switzerland, August 2004.
17. A. Rowstron, P. Druschel: *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*. IFIP/ACM Middleware 2001, Heidelberg, Germany, November 2001.
18. J. Shneidman, D. Parkes: *Rationality and Self-Interest in Peer-to-Peer Networks*; 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, CA, USA, February 2003.
19. J. Shneidman, D. Parkes: *Using Redundancy to Improve Robustness of Distributed Mechanism Implementations*; In Proceedings of 4th ACM Conference on Electronic Commerce (EC'03), San Diego, CA, USA, May 2003.
20. I. Stoica, R. Morris, D. Karger, M. Kaashoek, H. Balakrishnan: *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*; ACM SIGCOMM 2001, pp. 149-160, San Diego, CA, USA, August 2001.
21. W. Thigpen, T. Hacker, L. McGinnis, B. Athey: *Distributed Accounting on the Grid*; 6th Joint Conference on Information Sciences, pp. 1147-1150, 2002.
22. V. Vishnumurthy, S. Chandrakumar, E. G. Sirer: *KARMA: A Secure Economic Framework for Peer-to-Peer Resource*; Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA, June 2003.
23. B. Yang, H. Garcia-Molina: *PPay: Micropayments for Peer-to-Peer Systems*; ACM Conference on Computer and Communications Security (CCS '03), Washington, DC, USA, October 2003.