PPay: Micropayments for Peer-to-Peer Systems

Beverly Yang Hector Garcia-Molina
{byang, hector}@db.stanford.edu
Stanford University

Abstract

Emerging economic P2P applications share the common need for an efficient, secure payment mechanism. In this paper, we present PPay, a micropayment system that exploits unique characteristics of P2P systems to maximize efficiency while maintaining security properties. We show how the basic PPay protocol far outperforms existing micropayment schemes, while guaranteeing that all coin fraud is detectable, traceable and unprofitable. We also present and analyze several extensions to PPay that further improve efficiency.

1 Introduction

In recent years, peer-to-peer (P2P) applications have emerged as a popular way to share huge amounts of data, compute cycles, knowledge/expertise, and other resources. For example, as of April 30 2003, the KaZaA [9] multimedia file sharing application reported over 4.5 million users sharing a total of 7 petabytes of data. The reason P2P systems can achieve such scale is their ability to pool together and harness large amounts of resources at the "edge" of the network (i.e., at the peers), rather than relying on expensive centralized resources.

Recently, the "killer" application of free multimedia file sharing has come under legal fire from the recording industry. As a result, new P2P applications are emerging, such as pay-per-transfer file sharing systems, Grid-style computing systems, web service catalogs/service discovery, data storage/archival systems, etc. The goal of these new applications are for the rightful owners of goods or services to be compensated; therefore they all share a common need: an efficient and secure *micropayment* mechanism, by which peers can purchase services from one another.

The problem with most existing micropayment schemes is the heavy load on the trusted, centralized broker. A broker is required to handle accounts, distribute and cash coins, provide security (such as double-spending detection), etc. Although payments need not be online, eventually the broker must take some action for every transaction; as a result, broker load is always $O(n)$ in the number of transactions. Brokers therefore present a scalability and performance bottleneck for any system using these micropayment schemes.

However, P2P applications have two main characteristics that we can exploit to address the above problem:

- First, peers serve as both *vendors*, who sell goods, and *buyers*, who purchase them. As a result, a *transferable* coin can be used in many transactions before the broker must be involved in cashing it.
- Second, and more importantly, are the massive resources available at the peers themselves. Like other P2P applications, if we can tap into this pool of resources and shed the broker's load onto the peers, we can build a micropayment scheme with much better scalability and performance properties than existing ones.

The main challenge in exploiting the above two points is to ensure that the *security properties* of the scheme are not compromised. For example, transferable coins delay the detection of coin fraud, and we now want operations normally done by the trusted broker to be performed by untrusted peers.

In this paper, we present PPay[1], a micropayment scheme that addresses the dual problem of improving performance while maintaining security. PPay only requires broker involvement when peers open or close accounts, for arbitration, and in limited cases, to perform services on behalf of offline peers. We show that under realistic application scenarios, PPay significantly outperforms existing schemes in terms of broker load. At the same time, however, PPay guarantees that all fraud is detectable and traceable. Our contributions are as follows:

- We present (Section 3) the basic PPay protocol for floating, self-managed coins.
- We show (Section 4) how the necessary security properties hold.
- We describe (Section 5) several extensions to the basic protocol that greatly improve system performance.
- We present a detailed performance analysis (Section 6), using simulations to (1) show PPay outperforms existing micropayment schemes in our target application scenario (Section 7.1), (2) determine the special cases in which PPay is *not* better, and (3) show how to tune the performance of PPay by setting the appropriate pa-

[1] "PPay" is short for *PeerPay*

rameter values for our extensions (Section 7.2).

We believe that our performance analysis is unique in this field, where most schemes are proposed and only qualitatively analyzed.

Preliminaries. One very important point to keep in mind is that micropayments are payments of a *small amount*. Therefore, (1) utmost security is not required, and (2) the payment mechanism must be lightweight, otherwise the cost of the scheme will outweigh the value of the payment. A real-world analogy to vending machines is illustrative. Because the goods are low-cost, it is not profitable to hire an employee to attend the machine. As a consequence, candy bars may be stolen, or clients may receive low-quality, stale candy. Likewise, most micropayment schemes (all schemes known to the authors) do not guarantee fair exchange of goods and payment. An escrow service (e.g., [8]) would make transactions more expensive than the value of the goods. Therefore, the main disincentive for cheating is that a bad user will be shunned, and will not be able to carry on business in the future.

Furthermore, because payments are offline, coin fraud (e.g., using a counterfeit coin in a vending machine) may not be discovered until after the fact. However, offline payments are preferred from a practical standpoint because they have lower latency, communication costs and computational costs. Naturally, large-scale fraud needs to be detected and punished (discussed in Section 4). On the whole, however, effective micropayment systems simply need "good enough" security where fraud is detectable, traceable and unprofitable, while maintaining high efficiency.

Notation. The public key of a node N is denoted PK_N, and the secret key SK_N. A message M signed by some key K is denoted as $\{M\}_K$ (e.g., $\{M\}_{SK_N}$).

2 Related Work

Many micropayment schemes have been proposed in the past, including NetBill [13], DigiCash [4], Millicent [5], and the scheme proposed by Rivest et. al. [14, 11]. Each of the above schemes require $O(n)$ broker load, where n is the total number of payments made. In fact, in DigiCash and NetBill, every transaction requires an online broker involvement, which as we noted earlier, has worse performance but better security.

Payword hash chains [14] and electronic lottery tickets [11] greatly reduce the "hidden constant" of this $O(n)$ cost in the work of Rivest et. al., but we note that both these techniques may also be applied directly to our scheme to reduce peer and broker load (see Sections 5.4 and 7.2.3).

The Millicent scheme also has a very low "hidden constant," because the broker does not need to perform any signature generations or verifications. As a result of this low security, however, there is no way of proving fraud. Instead, the broker will eject any vendor for which it receives many complaints. The money spent by the broker to pre-purchase that vendor's scrip will be lost. Furthermore, given the possibility of malicious peers "framing" a particular user, we prefer a system with stronger security properties.

At first glance, *transferable coins* [1, 3] seem to exploit the first characteristic of P2P systems we mentioned in the previous section. The basic idea behind transferable coins is similar to *layered coins* discussed in Section 5.2. When a user uses a coin for payment, he adds some information to the coin, such as his (possibly disguised/hidden) identity. Most transferable coin schemes have been developed in a macropayment context, and are very heavyweight in order to provide additional security properties such as anonymity and untraceability. In general, anonymity and untraceability are not used for micropayments because the cost outweighs the benefits. However, even a lightweight scheme employing the basic idea of transferable coins has the following properties: (1) the coin grows in size as it is transferred, and (2) double-spending detection is significantly delayed. For the above reasons, in practice, transferable coins must be cashed after they have been used a certain number of times. Even with transferable coins, then, broker load will still be $O(n)$ in the number of transactions. Furthermore, even if coins could be transferred indefinitely, eventually when the coins are cashed, the broker will receive $O(n)$ amount of data, because each coin's size is $O(n')$ in the number of times it was transferred.

Elements of the PPay protocol are *optimistic*, in that transactions between peers do not involve the broker unless an exception occurs (e.g., a peer dies). Optimistic protocols for fair exchange have been studied in the past (e.g., [2]). Unlike fair exchange, however, a micropayment protocol will always require some broker involvement in non-exception cases (e.g., when the final payment is made); hence, the focus of this work is to minimize broker involvement in non-exception cases.

Finally, MojoNation [12] developed a micropayment system for P2P applications; however, the company has since switched focus, and has left little documentation of their micropayment scheme. From informal interactions with others knowledgeable about MojoNation, it appears that their system used a technique similar to *soft credit windows* (described in Section 5.4), and had broker load that was $O(n)$ in the number of coin transfers.

In Section 7.1, we provide a performance comparison between our scheme and the micropayment scheme of [11, 14] in conjunction with transferable coins. For a survey over these and other schemes, please refer to [10].

3 PPay Protocol

In PPay, we present the concept of *floating, self-managed currency* to greatly reduce broker involvement. Floating currency allows digital currency to "float" from one node to another without the involvement of a centralized broker, which is only useful when peers act as both buyers and vendors. However, unlike traditional transferable cash, coins in PPay do not grow in size as they are transferred. We say the currency is "self-managed" because all security related to a coin, except for when the coin is first created or cashed, is managed by the users who "own" the coins. In this way, we harness the resources available at individual peers.

Basic Implementation. Self-managed, floating currency is implemented as follows. A user U purchases digital coins from the broker B. A *raw coin* from the broker has the following form:

$$C = \{U, sn\}_{SK_B} \quad (1)$$

where sn is the *serial number* of a coin that uniquely identifies it. User U is now the *owner* of coin C.

Now, when U wishes to buy a service from another user V, it will send V the *assigned coin*:

$$A_{UV} = \{V, seq_1, C\}_{SK_U} \quad (2)$$

where seq_1 represents the *sequence number* of the assignment. Each assignment of a coin must have a sequence number that is greater than the sequence number of the previous assignment, if a previous assignment exists (that is, if this assignment is not the first of this coin). V is now the *holder* of the coin. At any time, a holder may "cash" its assigned coins (or any unused raw coins) at the broker for a flat fee (the fee is meant to deter users from constantly cashing coins, which reduces the benefits of floating currency).

Instead of cashing the assigned coin, V may instead pay another node X using the same coin. In the basic case, V sends the following *reassignment* request to owner U:

$$R_{UVX} = \{X, A_{UV}\}_{SK_V} \quad (3)$$

When U receives R_{UVX}, it keeps a record of the reassignment request in order to later prove that V "relinquished" the that particular assignment of the coin, in the case of a dispute.

After receiving and processing the reassignment request, U then sends to V and X the new assignment:

$$A_{UX} = \{X, seq_2, C\}_{SK_U} \quad (4)$$

Nodes V and X are responsible for verifying that $seq_2 > seq_1$. After the new assignment has been issued, the old assignment, A_{UV}, is no longer valid. In Section 5.2 we discuss the option of *layered* coins, where V reassigns a coin to X without going through owner U. Note that both V and X receive the new assignment. That way, if for some reason X claims to have not received the coin, V can send its copy to X. If neither V nor X receive the coin, then the scenario is identical to the one in which U is simply down.

Downtime Protocol. Peers must be online to manage their coins. Say a node V receives a coin owned by U, and then attempts to spend it at node X. If U is unavailable (e.g., due to network partitions or natural downtime of a computer), or if U is attempting an attack, then it will not reassign the coin.

If the owner of a coin is not online (which the holder should determine via a ping *before* sending the reassignment request), the holder should attempt to make the payment via a coin owned by a different user who is online. In the case where this option is not available, or where the holder already sent the reassignment request but received no response, the holder may have the coin reassigned by the broker. Continuing our example from above, broker B will generate the newly assigned coin:

$$A_{UX} = \{X, seq_2, C\}_{SK_B} \quad (5)$$

which is identical in content to an assignment issued by owner U. The broker will then store the reassignment request from V and send it to owner U when it comes back online. U will be responsible for detecting any misconduct while it was down.

In the off chance where both the broker and the owner reassign the coin (e.g., because the holder timed out on waiting for a response from the owner), no damage is done. Both assignments will have the same sequence number and be assigned to the same user; hence they are essentially the same assignment. If the holder tries to reassign the coin to two different users, the owner can detect this misbehavior through the two reassignment requests. Please refer to Section 4 for a detailed security analysis of our protocol.

Broker B will charge both V and U a percentage of the reassigned amount for this service. Both parties must be charged in order to prevent attacks where U always pretends to be down, or V pretends U is down. Note that this extension provides incentives for nodes to remain online, and to minimize the involvement of the broker. It also makes any attack by the owner unprofitable, since the owner will not only be charged a fee, but it will also be unable to reuse the coin for itself.

An alternative to this downtime protocol is for peers to simply print a new coin whenever it does not hold another coin whose owner is alive. What a peer decides to do will depend on how much it is willing to manage another coin, and on what the respective fees are (which the broker controls). Later in Section 7.1 we present a performance analysis of this alternative method of dealing with offline owners. Even if this alternative method is used, however, we still need the downtime protocol to handle the case when the owner does not respond after the reassignment request has already been sent.

Observations. First, the broker's involvement is now limited to $O(m)$, where m is the number of coins issued. Since each coin can be involved in thousands of payments before being cashed, m may be orders of magnitude lower than n, the number of payments made. Furthermore, in the "ideal" case where all nodes own the same number of coins, and coins are randomly selected to make payments, the cost per node of handling coin reassignments remains constant, regardless of the number of nodes in the system – i.e., it scales perfectly.

Next, we observe that reassigning a coin is relatively expensive (2 signature generations, 4 signature verifications, and 3 extra network messages). However, here we can directly use existing micropayment techniques originally intended to reduce broker involvement. For example, we can use *electronic lottery tickets* [11] to reduce the number of coin reassignments by a factor of s, where in practice s can safely be set to roughly 100. We refer readers to the citation for details.

Third, PPay is offline with respect to the broker, meaning that broker load is conserved. However, because payments must (usually) be online with respect to the coin owner, overall communication cost and latency is higher than with true offline payments. In Section 7.1 we compare the load of peers in PPay with other offline schemes.

Finally, we note that downtimes increase the involvement for the broker, but to a limited degree. As downtimes increase, PPay essentially degenerates into a regular brokered scheme where broker involvement is $O(n)$ in the number of transactions. In Section 7.1 we show how our scheme maintains good performance, even in the face of many offline peers.

4 Security

As with any offline payment system, *coin fraud* is an important issue that must be carefully addressed. Like other micropayment schemes, PPay does not prevent coin fraud at the outset, but instead makes fraud unprofitable. To achieve this goal, we need to ensure that (1) any fraud can be detected and traced back to the misbehaving peer (Section 4.1), and (2) proper punishments are in place (Section 4.2). Note that although a payment scheme can be designed to make fraud impossible (e.g., by making all payments online through the broker), such a scheme would be very inefficient. Recall that the goal of micropayment schemes is to provide an *efficient* payment mechanism with "good enough" security – i.e., where fraud is unprofitable.

Throughout this discussion, we assume that the broker is trustworthy. Broker fraud is unlikely for several reasons, as discussed in [5]. First and foremost, brokers will be businesses that rely on customer satisfaction and reputation for success. Even if a broker can commit fraud without being caught, its customers will realize they are losing money and will cease to use the broker's services. Second, because micropayments are worth very little, the broker will need to cheat a lot to gain any meaningful amount. However, large amounts of cheating will result in greater likelihood of being caught. Finally, by our protocol any broker misconduct can be detected and proven. In implementation, this detection mechanism will require that nodes keep a record of all coin assignments it has ever held; later we discuss how *coin renewals* (Section 5.3) can strictly limits amount of state that must be kept. In practice, we do not feel that this detection mechanism will be necessary, due to the first reason. Hence, the following discussion will only focus on detecting and preventing peer misconduct.

4.1 Detection

To assure that coin owners and holders cannot commit fraud without leaving behind proof of misconduct, the following invariants must hold:

1. If H is the valid holder of coin C owned by U, then U does *not* have proof that H has relinquished its valid assignment. Proof of relinquishment is in the form of a signed reassignment request from H for coin C, for the specific assignment in question (i.e., with the matching sequence number).
2. The sequence number of each assigned coin is greater than the sequence number of the preceding assignment of the coin.
3. If H is *not* the valid holder of coin C, then U can refute any assignment that H claims with proof of relinquishment (as defined earlier).

Note that as a consequence of invariant 3, the owner of a coin must keep a full "audit trail" of the coin.

To show that our system can detect and trace various attacks, we must prove the following theorem:

Theorem 1 *The above three invariants hold at all times, for all coins C.*

For a full proof of the above theorem, we refer readers to Appendix A. The proof covers both the *default protocol*, where owners reassign their own coins, and the *downtime protocol*, where owners are offline and the broker must reassign their coins.

Attack Analysis. Given that Theorem 1 holds, it is now a simple matter for malicious peers to be detected and identified. Here we enumerate the most common attacks on a micropayment scheme, and show how all instances of the attack can be detected and traced back to the misbehaving peer.

First, the holder of a coin may commit fraud by **replicating** an assigned coin and spending it twice. For example, say owner U assigns a coin to V. User V then reassigns the coin to X, and then tries to reassign it again to Y. Given invariant 3, however, owner U will always be able to refute the second reassignment of coin C. Note that if owner U was offline when V replicated its assignment, then the second, invalid assignment will be retroactively refuted when U comes back online (please refer to Appendix A for a discussion of "retroactive" action in the downtime protocol).

The owner of a coin may commit fraud by wrongfully denying validity of an assignment, or by double-spending a coin (generating two coins with the same serial number). As an example of **wrongful denial**, a peer V may hold a valid assignment of coin C owned by peer U. If V tries to reassign C, and U claims that V's assignment is invalid, then V will lose the value of that coin. By our protocol, wrongful denial is prevented by default, because the burden falls upon the owner to prove that a given assignment is invalid. If an assignment is valid, by invariant 1, no such proof exists.

As an example of **double-spending**, an owner U may assign a raw coin C to V, and then assign the coin to X as well. The broker evades the double-spending attack by recording the serial number and owner of each coin that has been cashed, and the holder of the coin who cashed it. If multiple coins with the same owner and serial number are cashed, the broker will ask the owner of the coin to prove the invalidity of one or both of the assignments. If the owner is able to refute an assignment, then the holder of the coin will be punished. If the owner is *not* able to refute either assignment, then the owner will be punished. By invariants 1 and 3, the owner will only be punished if it double-spent the coin.

4.2 Punishment and Risk Management

In real life, if a business can be proven to be corrupt, then legal action can be taken. Likewise, because fraud can be exposed with proof, then the broker will be involved in the arbitration. In general, the broker B will require the misbehaving party to pay for any "fake" coins, as well as a penalty fee for misbehavior. If a node continues to misbehave, the broker may boot the node out of the system (e.g., never allow the node to cash in its coins, and/or inform other users to avoid the node).

Unfortunately, in any offline payment scheme, it is possible that when credit fraud is detected, the cheater is not able or available to make payments. In our system users must provide a valid credit card to the broker when they first join the system, and the broker can charge the penalty fees via this payment channel. However, if a user cheats, immediately closes down his credit card account, and disappears from the system, the broker can no longer charge the penalty fees. In the worst situation, the business owning the broker must settle with the user out-of-system, using the personal information provided when the user first joined the system (e.g., name, address, phone). Fortunately, we believe this type of "cheat and run" attack can be avoided in most or all cases given the following proper disincentives.

First, bear in mind that individual micropayments are worth very little. Hence, if the user cheats with a small number of coins, then the loss to the broker is negligible. To guard against medium to large cheat amounts, the broker may charge a deposit when the user first joins the system. When a user permanently leaves the system, he notifies the broker, who then recalls all coins belonging to that user. After the coins have been recalled and the broker has determined no cases of cheating, the user receives his deposit back, and may cash any coins it is currently holding. If the user cheats and disappears, without going through the formal process of leaving the system, he forfeits his deposit.

It is also possible that a user cheats with an enormous number of coins that a reasonable deposit can not cover. However, such a scenario means that the user must purchase an enormous number of low-value goods (e.g., 100000 web pages), which is not likely or realistic behavior. This observation is in contrast to credit card fraud, where the cheater can select from a huge range of goods that are potentially very expensive. Furthermore, to select and download all these goods will take quite some time. Because the user cheated with an enormous number of coins, there is a greatly increased chance that the user will be caught before he finishes downloading the goods. Finally, the user will always be caught, eventually. Hence, even if the user is able to cheat and disappear before he is caught, he will eventually face a collection agency and bad credit as a consequence. Given these disincentives, then, we believe that large-scale cheating will be very rare or nonexistent.

In terms of a peer's risk, the broker will guarantee refunds to cheated peers in all cases except large-scale fraud. With large-scale fraud, the broker will provide refunds as far as can be covered by the bad peer's deposit. The remainder of the fraud amount will be covered if the broker can extract payment from the bad peer through out-of-system means. Individual peers must therefore manage their risk; e.g., by not accepting too many coins from the same user, or by periodically cashing their coins.

5 Issues and Extensions

Here we consider a few issues in the scheme described, and provide extensions that solve or mitigate the problems. For each extension proposed, we also discuss the associated security considerations and risks.

5.1 Limit Certificates

Printing raw coins can be expensive for the broker, given that the broker's load is otherwise very low. Observe that the purpose of a raw coin is to notify other users that the owner U has purchased this coin; hence, the coin can later be redeemed at the broker. To achieve this same functionality, we can instead have users print their own coins, given proof that the broker has authorized them to do so. When a user U first joins the network, it obtains a *limit certificate* from the broker that specifies how many coins U is authorized to print. The certificate has the following format:

$$L = \{U, lim_l, lim_u\}_{SK_B} \tag{6}$$

Field lim_l denotes the lower bound on the serial number of the coins that U may print, while lim_u denotes the upper bound. User U will typically need to prepay the value of $lim_u - lim_l + 1$ coins. U may now print his own "raw coin" with the following format:

$$C' = \{sn, L\}_{SK_U} \tag{7}$$

where it must be the case that $lim_l \leq sn \leq lim_u$. Any raw coin with a serial number outside of the limits set by the certificate is not valid.

Security Considerations. Note that while user U may print a coin with the same serial number twice, such an action is no different from U copying and double-spending a raw coin issued by the broker. The security properties of self-printed coins with limit certificates is therefore no different from the security properties of broker-printed raw coins.

5.2 Layered Coins

Rather than requiring a node V to go through owner U to reassign coin C to node X, V can instead reassign C itself by directly sending X the following message:

$$P_{UVX} = \{X, V, seq_n, A_{UV}\}_{SK_V} \tag{8}$$

where seq_n is greater than the sequence number of A_{UV}. Similarly, X can then reassign the coin to some node Y by adding another "layer" to P_{UVX}. Each layer of the coin is effectively a reassignment request, and serves as proof of relinquishment. When a node finally does go to U to have the coin reassigned, U can obtain all necessary proofs by "peeling" off the layers.

Layered coins are a lightweight implementation of traditional transferable cash, with no anonymity or untraceability properties. Like transferable cash, layered coins grow in size as they are transferred. Unlike transferable cash, however, layered coins in our scheme will eventually be peeled at a peer, rather than the broker; therefore broker load is significantly lower.

Security Considerations. In Appendix B, we show how Theorem 1 continues to hold in the presence of layers. However, layers may still delay the discovery of fraud. For example, say user V replicates coin C belonging to owner U. If layers are not allowed, V will be caught as soon as it tries to spend C the second time. With layers, however, V can reassign coin C to two users without U's knowledge. Only after both coins are peeled, can U pinpoint V as the misbehaving party. In Section 7.2.1 we study the tradeoff of layered coins between security and efficiency.

5.3 Coin renewal

To limit the amount of state each peer must keep (e.g., audit trails, payment history), we can require coins to be *renewed* during a certain time window (e.g., a 30-day window starting at time t). After the renewal window closes, the owner of the coin may purge the audit trail of the coin and start the audit trail anew. Further, any payment history involving an expired coin may also be purged. Any coin that has a renewal date in the past is considered invalid. If nodes do not renew their coins, then they forfeit the coins after the renewal window closes. Hence, the renewal window must be large enough such that nodes have a fair chance to renew the coins in their possession.

Just as with reassignment, if V tries to renew a coin at U but U is not alive, then V may have the coin renewed at broker B, at a fee for both V and U. Alternatively, V may simply cash the coin if it is about to expire, thereby taking it out of circulation. The decision V makes will depend on the respective fees charged for renewal and cashing.

Assigned coins now have the following form:

$$A_{UV} = \{V, ts, r_b, r_e, C\}_{SK_U} \tag{9}$$

where r_b and r_e designate the begin time and end time of the renewal period, respectively, and ts is the current timestamp. When V wants to renew a coin, it sends the following renewal request to U:

$$N = \{A_{UV}\}_{SK_V} \tag{10}$$

U will then return the assigned coin to V with a new renewal period specified:

$$A_{UV} = \{V, ts, r'_b, r'_e, C\}_{SK_U} \tag{11}$$

Security Considerations. In Appendix C, we discuss what happens if owner U tampers with the renewal window (e.g., sets the window to a time in the past) and how a holder V can easily guard against that attack. We also prove that Theorem 1 continues to hold in the presence of renewals. Here, we briefly discuss how the size of the renewal period affects the worst-case time to detect fraud.

As we mentioned in Section 4.2, we expect large-scale fraud to be rare, and likely to be detected within a short amount of time. However, in the worst case, fraud will not be detected until conflicting coins (i.e., with the same serial number) are cashed, reassigned or renewed at the broker. There are no guarantees as to when a given coin will be cashed or reassigned; however, the coin must be renewed within the renewal period. Hence, the worst-case time to detect any kind of fraud is equal to the renewal period. In practice, we must balance the tradeoff between renewal load and worst-case fraud detection. In Section 7.2.2, we discuss further this tradeoff between security and performance.

5.4 Soft Credit Windows

In some systems, payments are not only made for large items, such as downloads, but also for small items, such as forwarding messages or answering queries. Such systems use economic incentives to ensure the proper operation of the P2P infrastructure. Because these "picopayments" may need to be very fast and will occur very frequently, we cannot require a signature generation and verification per payment. Furthermore, because the value of picopayments are very low (fraction of a micropayment), security surrounding them can also be low.

To make payments fast, we observe that in P2P systems, the economic relationship between nodes is often symmetric. For example, U may provide several services to V, but later, V will then provide services to U.[2] Due to this symmetry, payments between nodes can "cancel" each other out before the exchange of a signed lottery ticket or coin is required.

We take advantage of the above observation by utilizing *soft credit windows*.[3] For each node V that U interacts with, U must keep track of the balance M_{UV} of "picopayments" it owes that node. The balance begins at 0. Each time V performs a service for U, M_{UV} is increased. When U performs a service for V, M_{UV} is decreased. At the same time, V is maintaining M_{VU}. Under correct behavior, $M_{UV} = -M_{VU}$ at all times. Naturally, peers can easily behave incorrectly; we will discuss this problem shortly. When M_{UV} falls out of the range $(-x, x)$, then U must pay V, or vice versa. The balances are then reset to 0.

We make several observation about the performance of soft credit windows. First, if x is large enough to handle the variance in balance, theoretically, two nodes may never need to exchange coins, even if they provide enormous amounts of service for each other. Second, soft credit windows require you keep state per node you interact with, but this per-peer state is very small. Even if each node interacted with a million other peers in the system, total state would be on the order of 6MB of memory (i.e., one integer and one short per node), which is not at all significant on today's computers. Finally, soft credit windows are very efficient, not only because of the symmetric relationship between nodes, but also because no signature generation/verification is needed. Even if the relationship is totally asymmetric, we can still utilize soft credit windows to decrease the number of signature operations required.

Security Considerations. Because soft credit windows do not provide proof that one party is indebted to another, choosing window size x presents a tradeoff between efficiency and risk. As we observed earlier, large x may result in super-linear reduction in the number of coin reassignments. However, the larger x is, the more a peer stands to lose if it is cheated, for example, if the other peer refuses to make a payment when the balance reaches x. Note that it is possible for to U believe the balance has exceeded the window, but V disagrees. If this situation occurs frequently, then U should learn to avoid V. However, if the situation occurs infrequently (e.g., due to honest mistakes, lost messages), then the loss of a few picopayments is a very small deal. In Section 7.2.3 we will explore this tradeoff between efficiency and risk as x varies, and as the number of misbehaving peers in the system varies.

Because potential loss sustained by a peer can be high, we need to investigate ways of bounding this loss. We now present the *NetLossCap* method of managing the risk of credit loss.

Let $c_{ij}(t)$ denote the credit (in picopayments) peer p_j owes peer p_i at time t ($c_{ij}(t) = -c_{ji}(t)$). Potential *credit loss* $l_i(t)$ of peer p_i is defined as $l_i(t) = \sum_{p_j \in network} c_{ij}(t)$. The maximum value of $c_{ij}(t)$ is x (the size of the window), hence maximum possible loss $l_i(t)$ is $x \cdot \|network\|$. Loss is implicitly potential until a peer permanently leaves the system.

In the *NetLossCap* method, peers set a *cap value* v that limits credit loss l_i at any point in time. If $l_i(t) = v$ and a peer p_j wishes to extend its credit at p_i, p_i will refuse. Instead, in order to receive service, p_j will need to pay p_i a coin. In paying this coin, p_j is essentially extending p_i x picopayment credits (since a coin is worth one window's worth of picopayments). Loss $l_i(t)$ is thus decreased by x, and p_j can now receive service at p_i.

Note that when p_j pays p_i the coin, other requirements should not be broken. In particular, if $l_j(t) > v - x$ prior to the coin payment, then p_j cannot pay p_i – otherwise $l_j(t)$ will go over the cap value. In addition, p_j cannot pay p_i if $c_{ij}(t) > 0$ prior to the coin payment, otherwise $c_{ij}(t) + x$ will fall out of the credit window. No exchange of credit and service will occur unless p_j can afford to make the payment without unnecessarily affecting its own risk. In Section 7.2.3, we investigate the impact of *NetLossCap* on system performance and risk.

Hash Chains. An existing, popular way to make fast, small payments are Payword hash chains [14]. For readers not familiar with hash chains, a longer discussion of the statements made in this section can be found in Appendix D.

[2] Symmetric behavior may not be seen for large items, e.g., in download patterns. However, if payments are made for small items such as forwarding messages or processing queries, then relationships become much more symmetric.

[3] The authors have been informed that MojoNation [12] used a similar idea, which they called *bilateral accounting*. However, no documentation is available to confirm or cite. Beyond the introduction of this idea, we also discuss risk management for this technique, as well as a performance analysis in Section 7.2.3

Unlike soft credit windows, hash chains present no credit loss risk to peers, assuming broker functionality is extended to honor partial hash chains. However, maintaining hash chains has high state overhead. In a system with a million peers and hash chains of length 100, over 4GB of state per peer must be maintained. Furthermore, because hash chains can only express an *increasing* amount of credit (as opposed to both increasing and decreasing, like soft credit windows), they can not take advantage of the symmetric relationship between peers. If users A and B exchange an infinite but balanced number of picopayments, the will need to exchange an infinite number of coins, rather than zero coins. In Section 7.2.3 we compare the performance of hash chains and soft credit windows.

6 Experimental Setup

In the previous three sections we have provided a formal description of the PPay micropayment scheme. Now, we wish to use simulations to answer the following important questions: (1) Is PPay feasible in terms of broker load, peer load, and the amount of state needed to implement our security features? (2) How does PPay perform relative to other schemes? What usage scenarios produce best or worst case behavior? (3) How should we tune parameter values (such as credit window size) for best performance?

We evaluate PPay in the context of a file-sharing system using the GUESS protocol [7] to discover files. GUESS is a new protocol from the Gnutella Development Forum (GDF) aimed at addressing performance and security issues in Gnutella [6] – the largest open P2P system in operation today. In brief, under the GUESS protocol each peer keeps a “pong cache”, which is a list of IP addresses of other peers in the system. When a peer has a query, it “probes” the peers in its pong cache with the query message, one by one, until a match is found. There are many policies by which peers can order their probes; for simplicity, we assume that the order is random.

One problem with the current GUESS protocol is that it does not provide the proper incentives for peers to cooperate – peers can easily cheat the system by sending out its probes all at once. This behavior incurs a much higher load on other peers, while drastically improving the response time of the search for the querying peer. If all peers act according to their best interests, the system might fail as if under a denial of service (DoS) attack. In order for GUESS to work, there must be a way to give peers an incentive to adhere to the protocol. One straightforward proposal is to have peers pay for each probe. Peers will then be motivated to probe as few peers as possible to answer their queries. In our evaluation, we use the PPay micropayment system to implement this incentive mechanism. In particular, peers will make one picopayment per probe. Coin payments are then made as credit windows are filled.

Name	Default	Description
Network Size	1000	Number of peers in the network
Cache Size	100	Size of each peer’s pong cache
Query Rate	$9.26 \cdot 10^{-3}$	The expected number of queries per user per second
Lifespan Multiplier	1	Multiplier used to extend the period of time peers remain online
Downtime Multiplier	1	Multiplier used to extend the period of time peers remain offline
Credit Window Size	10	Size of a soft credit window or hash chain
Credit Window Type	Payword	The type of credit window used (Payword, soft credit windows)
Max Layers	2	Maximum number of layers allowed on a coin before it must be peeled
Renewal Wait	30 days	Elapsed time between audit trail purges for a given coin
Scheme	PPay	The micropayment scheme used (PPay, PPay*, or EC)
Net Loss Cap	∞	Maximum allowed net loss per peer due to soft credit windows

Table 1: System and Micropayment parameters

6.1 Metrics

Performance of a micropayment scheme is measured primarily by *load*. Load is defined as the amount of work an entity must do per unit time. Load is measured along three resource types: *incoming bandwidth*, *outgoing bandwidth*, and *processing cost*. Bandwidth is measured in bits per second (bps). Processing cost is conceptually measured in cycles per second (Hz), although we actually measure costs in coarse units before converting them to approximate cycles (described further below). We look at load from two perspectives: the broker, and the peers. Recall that our goal is not overall efficiency, but broker efficiency. At the same time, however, we want to ensure that peer loads are reasonable.

In addition to load, we also look at *state*, and *credit loss*. State is simply the number of “proofs” (i.e., reassignments or renewals) that an entity must store. Because the amount of state varies over time, we will be measuring the average state over time.

6.2 Parameters

We will be comparing the performance of different *configurations* of PPay, where a configuration is defined by a set of system and micropayment parameters, listed in Table 1. Most system parameters will remain unchanged in our experiments, as they do not qualitatively affect the results. We will describe these and other parameters in further detail as they appear later in the section. In our experiments, unless otherwise specified, the default parameter values are used.

Note that `Network Size` = 1000 is a modest number of peers, compared to the scale of an actual P2P application. We had to limit network size because of the length of our simulations. However, most results shown in Section 7 should be a good qualitative representation of any network size, because PPay is scalable. For peer loads, although application-level purchases grows with the number of peers in the system, so does the number of peers that manage coins. In fact, peer load *decreases* as network size increases because more payments are absorbed by the greater total amount of picopayment credit (recall that maximum possible credit is equal to the network size times the size of the credit window). Broker load scales sublinearly with network size for the same reason. The only potentially unscalable result the *NetLossCap* method for bounding credit loss, and we discuss this issue in Section 7.2.3. Please see Appendix E for figures showing the above scalability results.

Action	Bandwidth Cost (Bytes)	Processing Cost (Units)
Send/Recv Query	$70 + querylength$	1
Process Query	0	$14 + 1.1\cdot$ # results
Send/Recv Response	$208 + 76 \cdot$ # results	1
Hash	0	5
Signature generation	0	5240
Signature verification	0	365
Send/Recv Coin	$172 + 44 \cdot$ # layers	$1 + .5\cdot$ # layers
Send/Recv Reassignment	$200 + 44 \cdot$ # layers	$1 + .5\cdot$ # layers
Send/Recv Renewal	$196 + 44 \cdot$ # layers	$1 + .5\cdot$ # layers
Send/Recv Limit Cert.	190	1

Table 2: Costs of atomic actions

6.3 Simulation Framework

In a given simulation run, we simulate `Network Size` peers participating in the GUESS protocol. At the beginning of the simulation, peers begin with a limit certificate but no printed coins (e.g., as if the peers just joined the network). Over time peers print new coins as they are needed. We assume the limit certificate is large enough to handle each peer's needs for the duration of the simulation, which is 40 simulated days. Throughout the simulation, peers may go offline and come back online. The duration of these online and offline periods follow the distribution of lifespans and downtimes measured by [15] over the Gnutella network. We may tune online or offline times via the `Lifespan Multiplier` and `Downtime Multiplier` parameters, respectively. If `Lifespan Multiplier` = x, then all values in the measured distribution of lifetimes are multiplied by x. At the end of the simulation, we assume that all peers leave the system and cash in their coins. Doing so is a worst-case scenario for our scheme since in practice, the existing coins can continue to "float" before they are cashed.

The "arrival" of queries at each peer follows a Poisson process with rate `Query Rate`. When a peer has a query, it will sequentially probe the peers in its pong cache, which is of size `Pong Cache Size`. Peers in the cache may be offline, and the queryer need only pay peers that are online. To determine whether a peer returns a result for a query, we use the query model developed in [16]. Though this query model was developed for hybrid file-sharing systems, it is still applicable to the file-sharing systems we are studying. The probability of a peer returning a result depends partially on the number of files owned by that peer; number of files owned by peers are assigned according to the distribution of files measured by [15] over Gnutella. If a peer returns a result to the queryer, then the query is stopped. Otherwise, the queryer will probe every live node in its cache before stopping the query.

When a peer makes a payment, it will first try to pay with a coin for which the owner is online. If the peer has no such coin, it will attempt to pay with a coin via layers. If all its coins already contain the maximum number of layers, the peer may reassign a coin through the broker, or print its own coin if it has a valid limit certificate (in Section 7.1 we compare these two alternatives). Within each class of coin (e.g., the class of coins whose owners are alive), the peer may often choose among several coins in its possession. The order in which coins are selected may have security and performance implications; for example, a peer V may choose a coin owned by another peer U, if V already holds many coins belonging to U. This policy decreases V's risk if U commits large-scale fraud. While the ordering of coins is an interesting question, in our simulations coins within a class are accessed through a queue.

Action Costs. Our simulations tell us how many actions each entity must execute. In order to calculate load, we now need to calculate to cost of these actions. There are several "macro" actions in our cost model: receiving and processing a query, reassigning a coin, requesting a coin renewal, etc. Each of these actions can be broken down into smaller "atomic actions" whose costs are listed in Table 2. Due to space limitations, we will defer the discussion of how these costs are determined to Appendix G.

7 Results

In this section we present and discuss the results of our simulations. We divide the results into two sections, corresponding to the two main questions that we wish to investigate: (1) The performance of the PPay micropayment scheme relative to other schemes, and (2) the tradeoffs involved in setting micropayment parameter values.[4]

[4]Due to space limitations, we report all load results in terms of processing cost. Bandwidth results can be omitted because in all our experiments, either behavior is comparable to processing cost results, and/or the absolute bandwidth costs are very low, making them insignificant relative to processing costs.

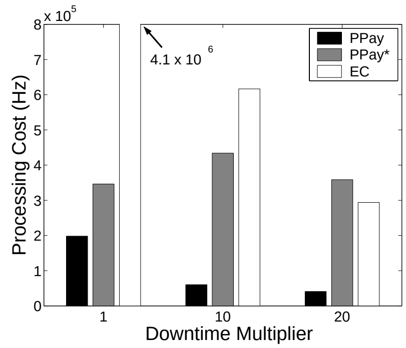

Figure 1: Comparison of broker processing loads in PPay vs. EC as Downtime Multiplier varies

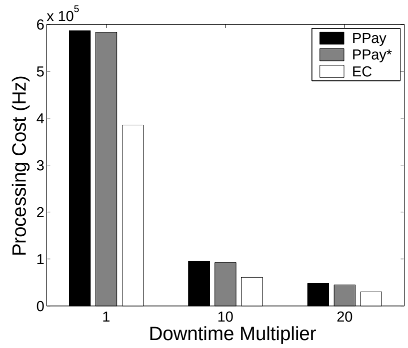

Figure 2: Comparison of peer processing loads in PPay vs. EC as Downtime Multiplier varies

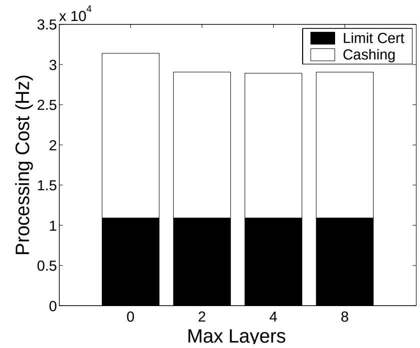

Figure 3: Broker load is barely affected by Max Layers

7.1 Comparison with Existing Schemes

We compare PPay against the *electronic check* micropayment scheme that is used, for example, by Rivest et. al. in [11, 14]. We choose the electronic check (EC) scheme among many possible existing schemes because it is used in some of the most recent research in the area known to the authors, and because it provides a comparable level of security with PPay. In this scheme, users print tokens (like coins) to give to vendors as payment. Vendors then give these coins to the broker to redeem. In fact, the EC scheme is a special case of our scheme in which peers immediately[5] cash every coin they receive. On top of EC, Payword hash chains are used to delay the payment of a coin, as discussed in Section 5.4. We also allow the use of layered, transferable coins. The maximum number of layers allowed before a coin must be cashed is set by `Max Layers`.

In [11], Rivest et. al. also refine the idea of electronic lottery tickets, which reduce the number of coin payments without requiring large amount of state. Like Payword, electronic lottery tickets can be implemented on top of PPay as well. From the broker perspective, electronic lottery tickets have the same effect as Payword in reducing the number of coin payments by a constant factor. Hence, in our performance study we consider Payword, but not electronic lottery tickets.

We compare the EC scheme to two variants of PPay. By default, we assume peers prefer printing a new coin over using the downtime protocol. If a peer must make a payment, and it does not possess any coins whose owners are alive or that can be layered, the peer will print a new coin. In the second variant, which we call "PPay*", peers prefer using the downtime protocol over printing a new coin. Likewise for renewals, in the first variant peers prefer to cash a coin that is about to expire, rather than have it renewed through the broker; in PPay*, the preference is switched. While the first variant conserves broker load in terms of renewals and reassignments, it also causes more coins to be printed, which must eventually be cashed. In our performance studies, we want to see whether this tradeoff is worthwhile.

Figure 1 shows us the broker's load (work per unit time) in the three micropayment schemes in terms of processing cost, as `Downtime Multiplier` is varied. Bars are labeled below according to `Downtime Multiplier` value, and shade indicates the micropayment scheme. For now, let us focus on `Downtime Multiplier = 1`, the default case.

We see from Figure 1 that PPay and PPay* perform significantly better than EC. While the broker must cash $O(n)$ coins in EC, where n is the number of transactions, it must only cash $O(m)$ coins in PPay and PPay*, where m is the number of coins. Therefore, in this simulation EC broker load ($40.6 \cdot 10^5$ Hz) is over 12 times heavier than in PPay* ($3.46 \cdot 10^5$ Hz), and over 20 times heavier than in PPay ($1.98 \cdot 10^5$ Hz).

At the same time, PPay outperforms PPay* because the broker no longer needs to reassign or renew any coins. Instead, for each reassignment/renewal the broker used to do in PPay* (which constituted about 43% of broker load), it now has to cash at most one new coin. Since cashing (1 verification) is substantially less expensive than reassignment or renewal (1 verification + 1 generation), broker load is likewise less expensive. The price of PPay falls onto the peer (discussed later), rather than the broker, which is a desirable tradeoff from our perspective. From Figure 1, then, we can conclude that floating, self-managed coins are crucial to minimizing broker performance in a P2P application.

Recall that the distributions of online and offline periods are measured directly from the Gnutella network. Therefore, these distributions reflect the instability of peers, and their tendency to come online for short periods of time.[6] Despite peer instability, PPay and PPay*

[5]By immediate, we mean that the coin is not reassigned to a different user before it is cashed.

[6]These distributions do *not* reflect the case where some peers may

continue to perform well. One interesting question is how far can we "push" the instability of peers before the broker in PPay becomes overloaded.

Let us look again at Figure 1. In general, overall application-level activity decreases in absolute value as offline periods increase in duration (i.e., `Downtime Multiplier` increases), because there are fewer peers online performing queries. At `Downtime Multiplier` = 20, only 3% of all peers are online at any given time, on average. We believe this scenario is very extreme, yet PPay continues to far outperform EC. At `Downtime Multiplier` = 20, broker load in EC ($2.94 \cdot 10^5$ Hz) is over 7 times heavier than load in PPay ($.41 \cdot 10^5$ Hz). However, broker load in PPay* actually *increases* as application-level activity decreases, because the number of reassignments and renewals that must go through the broker increases. At `Downtime Multiplier` = 20, 89% of broker load in PPay* ($3.59 \cdot 10^5$ MHz) consists of reassignment load. As a result, PPay* has the highest load of all.

While we have achieved our goal of significantly reducing broker load, we still need to make sure that peer load does not suffer. Figure 2 is just like Figure 1, except that it shows average peer load, rather than broker load. In this figure, we see that peers in PPay do have higher load than peers in PPay*, but only slightly. As we mentioned earlier, peers in PPay will need to manage more coins than in PPay*. The cost of managing these coins comes from reassignments and renewals. However, note that since the number of reassignments is not affected by the number of coins in circulation, having more coins does not necessarily mean that reassignment load will be higher. Furthermore, renewals only constitute about 6% of total peer load in PPay, which is higher than it is in PPay*, but still not a significant cost. As a result, PPay has peer load that is comparable to PPay*.

Also in Figure 2, we see that peers in PPay and PPay* have processing load that is roughly 50% higher than peer loads in EC. Peer bandwidth load (not shown) is 25-35% heavier than in EC. While the increase in peer load is non-trivial, we believe it is a desirable tradeoff with broker load for two reasons. First, the absolute value of peer loads are small; users probably would not notice the extra load. Second, although the absolute values of broker load shown in these experiments are not much higher than peer loads, recall from Section 6.2 that peer load *decreases* slightly as the number of peers grows, while broker load *increases* (sublinearly). When network size grows, for example, to 1 million, broker load will be much higher than peer load. Therefore, from a scalability viewpoint, it is most important for broker load to be minimized.

log on, log off, and then never come back to the network again. In our simulations, we assume that even though peers are online for short periods of time, they will always come back at some (possibly distant) time in the future. This assumption is reasonable in a system where peers have an incentive to come back (i.e., to make money).

Because PPay outperforms PPay* in all the scenarios we studied, in terms of broker load, we will only present results for PPay from this point on. PPay has also outperformed EC in the experiments we have looked at thus far, but there are specific scenarios in which EC outperforms PPay. In particular, the worst-case scenario for PPay is the extreme case in which each coin is used exactly once before being cashed. Even in this scenario, however, broker load is comparable in both micropayment schemes: bandwidth load is 35% higher in PPay, and processing load just 6.6% higher. Further details are given in Appendix H.

7.2 Setting Micropayment Parameters

7.2.1 Layers

Figure 3 shows us broker processing load as the maximum number of allowed layers on a coin (`Max Layers`) is varied. In this figure, `Downtime Multiplier` = 20, because the more peers that are offline, the more impact layers will have on performance. As we can see, broker performance varies very little as `Max Layers` increases We initially found this result quite surprising. Because coin owners are often down when a payment must be made (with almost 97% probability, on average), we would have expected layers to be very useful in avoiding the need to print new coins. However, recall that new coins are printed in two payment scenarios: (1) when the peer has no coins, and (2) when a peer has coins but the owners are offline, and layers cannot be used (e.g., because the maximum number of layers has been reached). It turns out that printing more coins in scenario 2 results in printing fewer coins in scenario 1, such that the total number of coins printed is not largely affected by layers. Because layers have so little impact on broker performance (nor peer performance), and since layers do delay the detection of fraud, we recommend that very low values for `Max Layers` (e.g., 0 or 1).

7.2.2 Coin Renewal

The `Renewal Wait` parameter specifies the system-wide renewal period for a coin. Renewals present a tradeoff between performance and security, and state management as well. As discussed in Section 5.3, the renewal period determines the worst-case time to fraud detection. Therefore, the closer together renewals are, the tighter the bound. State also decreases, because audit trails are purged after each renewal period. However, load increases, since each renewal is fairly expensive.

In our implementation, we assume that that coins are renewed every `Renewal Wait` = w days, and that the renewal window is also w days long. That is, if the renewal begin-time of an assignment is r_b, then after a renewal, the new renewal begin-time is $r_b + w$, and the renewal end-time is $r_b + 2 \cdot w$

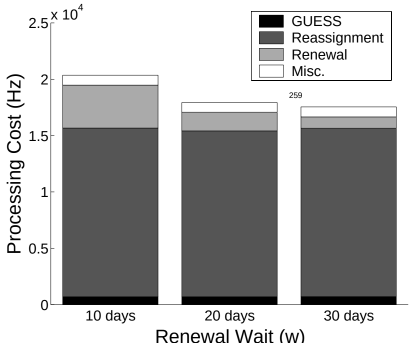

Figure 4: Peer load decreases as renewal period increases

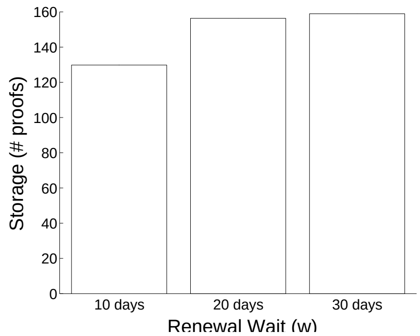

Figure 5: Peer storage (average number of proofs stored over time) increases as renewal period increases

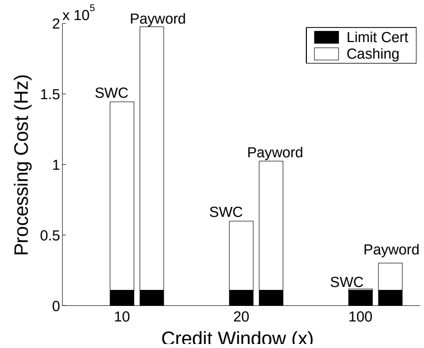

Figure 6: Soft credit windows have better broker load by accounting for symmetric peer relationships

Figures 4 and 5 illustrate the tradeoff between peer load and state as `Renewal Wait` is varied. Figure 4 shows peer processing load divided into four main tasks: GUESS-related work (e.g., processing queries), reassignments, renewals, and miscellaneous (e.g., cashing coins). As we can see, renewals take up a non-trivial fraction of peer load, ranging from 20% at `Renewal Wait = 10 days` to 6% at `Renewal Wait = 30 days`. However, Figure 5 shows us that although `Renewal Wait` does have an impact on the amount of state a peer must store, this impact is limited. Therefore, the more important tradeoff is between performance and security. In terms of broker load, `Renewal Wait` has very little effect, since the number of coins printed due to payment is far higher than the number of coins printed due to expired coins. Therefore, since broker load is unaffected and peer load is scalable, we recommend a relatively low `Renewal Wait`, such as 10 days in our application scenario. Note that the renewal period should not be too low (e.g., 1 day), otherwise peers may not have a chance to renew their coins due to downtimes.

We note that in Figure 4, for peer loads, the overhead of PPay relative to the load of the application (GUESS) is very high - about 18 times higher. However, this comparison is skewed, as GUESS is not processing-intensive. In terms of bandwidth, the overhead of PPay is just 4% of the GUESS load. Furthermore, if we set `Credit Window Size` to 100 (default is 10), then even the processing overhead of PPay is just 13%, and the bandwidth overhead just .03%. In both processing and bandwidth cost, absolute overhead is always very low; therefore we do not believe the overhead of PPay for peers to be a significant issue.

7.2.3 Credit Windows

Figure 6 shows the processing load of the broker as `Credit Window Size` is varied, for both Payword hash chains (Payword) and soft credit windows (SWC). We immediately observe that credit window size greatly affects broker load – large credit window is crucial for good broker performance.

As we discussed in Section 5.4, Payword cannot take advantage of the "back and forth" symmetry of peer relationships. Soft credit windows, on the other hand, are designed to take this symmetry into account. In Figure 6, we see that soft credit windows consistently outperform Payword for this reason: broker load with Payword ranges from being over 40% heavier at `Credit Window Size`=10, to almost 3 times heavier at `Credit Window Size` = 100. Furthermore, peer load shows an even greater performance benefit with soft credit windows. Figure 7 shows peer load as a function of window size, in terms of processing cost. Here, we see how Payword loads range from 17 times heavier (at a window size of 10) to over 38 times heavier (at a window size of 100)! Clearly, then, exploiting the symmetry of peer relationships is very beneficial to performance of both the broker and peers.

NetLossCap. Figures 6 and 7 show us the benefits of soft credit windows when no credit loss cap is applied. However, as we discussed in Section 5.4, in practice we need to bound the risk taken by peers using the *NetLossCap* method. Before discussing *NetLossCap*, let us define how a misbehaving peer might take advantage of soft credit windows.

We assume that the goal of a misbehaving peer is to receive the best quality service possible while spending no money. In other words, rather than being *malicious*, the peer is *lazy*. A lazy peer p_j will never pay a coin for service. Instead, it will only probe a peer p_i if $c_{ij} < x$, where x is the size of the credit window. Because a lazy peer p_j still wants to receive service, it will not lie about its view of c_{ij}, since that will cause peer p_i to shun it. Furthermore, the lazy peer is willing to provide service to other nodes, since it can potentially make money by doing so, and because it will at least receive credits to redeem for service in the future. For this same reason, then, the lazy

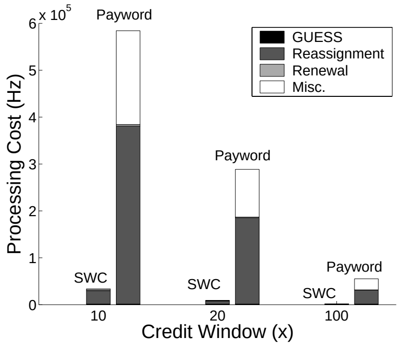

Figure 7: Soft credit windows have better broker load by accounting for symmetric peer relationships

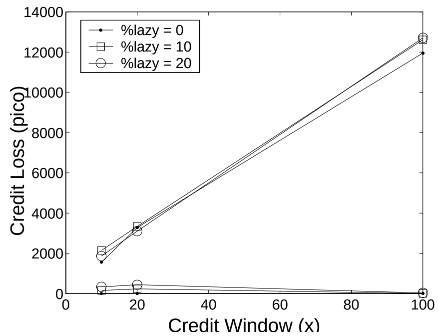

Figure 8: Maximum credit loss increases as credit window size increases

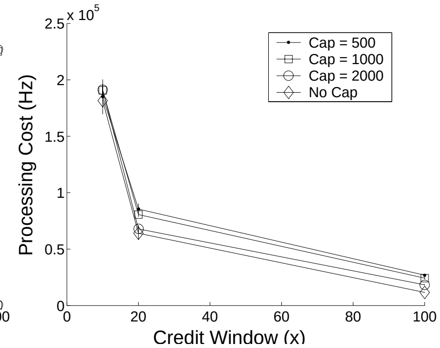

Figure 9: Broker load as the credit loss cap value varies

node will not lie about its cap value (e.g., will not refuse service to another peer by claiming that $l_j = v$).

Now, let us see how effective *NetLossCap* is as `Credit Window Size` and `Percent Lazy Node` (the percentage of peers in the network that are lazy) vary. Recall that we assume all peers permanently leave the system at the end of the simulation. Credit loss for a peer p_i in the following figures therefore reflect $l_i(40 \text{ days})$.

First, Figure 8 shows us the mean and maximum credit losses of non-lazy peers in our simulation, when *NetLossCap* is not used. Along the x-axis we vary credit window size, while the different curves show different values of `Percent Lazy Node`. For example, when `Credit Window Size` = 100 when there are no lazy peers, the largest value of $l_i(40 \text{ days})$ observed for some peer p_i is roughly 12000 picopayments. This worst-case observed value is rare, however; 90% of all $l_i(40 \text{ days})$ values are under 4000, while the average value of $l_i(40 \text{ days})$ is 0.

We make several observations. First, the percentage of lazy peers in the system does not seem to greatly affect the risk of non-lazy peers. The reasons for this are that (1) credit risk is already very high, with no lazy peers, and (2) lazy peers are not cheating, per se – they are simply acting conservatively. Cheating peers will eventually be cut off from the rest of the system; hence, lazy/cheating peers do not pose a significant additional risk to the existing credit risk of an honest peer. Second, soft credit window size greatly affects potential credit loss. The need for a way to bound credit loss is clear, if we are going to reap the benefits of large credit windows.

When we apply the `NetLossCap` method with a cap value of v, the maximum observed credit loss is always v, regardless of credit window size. As an example, when window size is 100 and $v = 500$, maximum credit loss is 500, as opposed to 12000 when there is no cap. However, we need to be sure that using *NetLossCap* does not significantly affect performance – otherwise, hash chains may be a better alternative than soft credit windows.

Figure 9 shows us broker processing load as cap value and credit window size are varied. Window size is varied along the x-axis, and different curves represent different cap values. Although the lines may be difficult to distinguish, the basic shape of the curves tells us what we need to know. First, the choice of `Credit Window Size` is still very important in tuning performance, for all cap values shown. Second, assuming the cap value is not too small, cap values do not significantly degrade performance. Although this result may be surprising at first, the cause goes back to the law of large numbers. Even though credit between pairs of peers may be imbalanced (e.g., $c_{ij} = 100$ and $c_{ik} = -100$), the sum of these individual credits, l_i, will fall closer to the expected value, which is 0. Finally, we found (not shown here) that the quality of service received by each peer is not affected by *NetLossCap*, for the cap values studied. Hence, *in the given simulation scenario*, the *NetLossCap* method is very effective in limiting risk while maintaining good performance.

However, we note that the *NetLossCap* method is not necessarily scalable. As the number of peers in the system increase, if we assume that all users interact with all other peers, then the cap value must also increase proportionally. However, if the number of peers with which a given peer interacts does *not* grow with the size of the network, which may be reasonable in many situations (e.g., peers only interact with neighbors), then *NetLossCap does* scale. In either case, important future work lies in designing a quick payment mechanism that can take into account symmetric peer relationships, but can also provide proof of the credit balance to the broker.

8 Conclusion

Business-oriented P2P applications hold great potential for the future. Before such applications can take off, however, there must be an efficient way for peers to pay each other for services. In this paper, we present PPay, an efficient micropayment scheme designed for P2P applications. By identifying and exploiting characteristics unique to P2P applications, PPay can significantly outperform existing schemes in terms of broker load, while maintaining a reasonable peer load. We present a detailed performance analysis over a sample P2P application to support our claims, and to illustrate how PPay can be tuned for best performance.

In the future, we plan to further investigate methods for risk management and reduction for both peers and brokers, particularly in the area of soft credit windows. In addition, as discussed in Appendix I, PPay is currently most suitable for applications where payments can be made in a few fixed denominations. Therefore, another important area of future research is how to extend PPay to efficiently handle payments of any amount.

References

[1] H. Antwerpen. Electronic cash. Technical report, University of Eindhoven, Masters Thesis, 1990.

[2] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for multi-party fair exchange. Technical Report RZ 2892 (# 90840), 1996.

[3] D. Chaum and T. Pederson. Transferred cash grows in size. In *Advances in Cryptology – EUROCRYPT '92*, 1993.

[4] DigiCash website. http://digicash.com.

[5] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The millicent protocol for inexpensive electronic commerce. In *In Proc. of WWW4*, 1995.

[6] Gnutella website. http://www.gnutella.com.

[7] GUESS protocol specification. http://groups.yahoo.com/group/the_gdf/files/Proposals/GUESS/guess_o1.txt.

[8] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *Proceedings of 3rd ACM Conference on Electronic Commerce*, 2001.

[9] KaZaA website. http://www.kazaa.com.

[10] J. Kytojoki and V. Karpijoki. Micropayments - Requirements and Solutions. http://www.tml.hut.fi/Opinnot/Tik-110.501/1999/papers/micropayments/micropayments.html.

[11] S. Micali and R. Rivest. Micropayments revisted. In *"CT-RSA"*, 2002.

[12] MojoNation website. http://web.archive.org/web/20020122164402/http://mojonation.net/.

[13] NetBill website. http://www.ini.cmu.edu/netbill.

[14] R. Rivest and A. Shamir. Payword and micromint: two simple micropayment schemes. In *Security Protocols Workshop*, 1996.

[15] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of the Multimedia Computing and Networking*, January 2002.

[16] B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *Proc. of the 27th Intl. Conf. on Very Large Databases*, September 2001.

[17] B. Yang and H. Garcia-Molina. Improving efficiency of peer-to-peer search. In *Proc. of the 28th ICDCS*, July 2002.

A Proof of Theorem 1

To show that these invariants hold at all times, we begin with the base case, which begins with the issue of a new raw coin C and ends with the completion of the first reassignment of C. We then use induction to show that our invariants hold, or can be enforced, for each subsequent reassignment. Readers not interested in the details may skip this section.

In the following analysis, we assume that public/private key encryption is secure, such that it is not possible for H' to forge an assignment with the owner's signature. As a result, for invariant 3, an "invalid" holder H' of coin C must have previously held C, and then relinquished it.

Base Case. First, the broker B issues a raw coin C to owner U. Invariants 1, 2 and 3 hold by default, because the coin has not yet been assigned.

Next, owner U assigns the coin to V via the following message:

$$A_{UV} = \{V, seq_1, ts, C\}_{SK_U} \tag{12}$$

Since V has not yet reassigned the coin, invariant 1 holds. Because A_{UV} is the first assignment of coin C, invariant 2 holds. Likewise, because no other node has previously held C, invariant 3 holds.

Say V decides to make a payment to X with coin C. V sends the following message to U:

$$R_{UVX} = \{X, ts, A_{UV}\}_{SK_V} \tag{13}$$

Here, we find one vulnerability of our scheme. At this point, invariant 1 no longer holds. Although V is still the valid holder of coin C until U issues the new assignment, U now has valid proof that V has relinquished the coin. Hence, U might never respond and thereby cheat V out of a coin.

However, note that from the perspective of all parties other than U, this attack is identical to the case where U is simply unavailable (i.e., not connected to the network). Therefore, no special effort is required to handle this attack; it can be addressed via the same exact mechanism used to address downtimes, discussed in the previous section. Furthermore, because U will be charged if it does not respond to a reassignment, and will not subsequently be able to use the coin for itself, U has no incentive to attack in this manner.

For now, let us assume U is available and not malicious. Invariant 2 holds because the current assignment (A_{UV}) is still the valid one. Invariant 3 holds because V is still the only holder in the history of coin C – hence, no invalid holders exist.

Now, say U responds to R_{UVX} with the new assignment:

$$A_{UX} = \{X, seq_2, ts, C\}_{SK_U} \tag{14}$$

At this point, X is the valid holder of C, and U does not have a reassignment from X; thus invariant 1 holds.

Invariant 2 holds iff $seq_2 > seq_1$. If V or X detects that invariant 2 does not hold, then this assignment is not valid. However, V or X can present the invalid assignment, along with R_{UVX}, to the broker as proof of U's misconduct. Because U cannot produce a reassignment request for C to be assigned to X with a sequence number lower than seq_2, U will be punished (as described below). Therefore, U again has no incentive to attack in this manner, because it will be punished in addition to being forced to produce a valid assignment.

For now, let us assume that A_{UX} contains a valid sequence number. Invariant 3 also holds because V is the only previous holder of C, and U has R_{UVX} as proof of relinquishment from V.

Induction. Now, let us assume that the coin is reassigned n times, and that all three invariants hold. We will show that the invariants continue to hold after the $n+1$th reassignment. Say node Y is the nth holder of coin C, and that Y will now reassign coin C to node Z.

Case 1. Z has never held this coin before. Y sends the following request to owner U:

$$R_{UYZ} = \{Z, ts, A_{UY}\}_{SK_Y} \tag{15}$$

After the message is sent Y is still the valid holder until U responds with the newly assigned coin. Again, invariant 1 is temporarily broken, but can be dealt with as described above. At the end of the nth reassignment, invariants 2 and 3 were valid, according to our inductive hypothesis. Because the valid holder of the coin does not change as a result of this message, invariants 2 and 3 continue to hold.

When owner U responds with the new assignment:

$$A_{UZ} = \{Z, seq_n, ts, C\}_{SK_U} \tag{16}$$

then the valid holder of the coin changes from Y to Z. Invariant 1 holds because Z is the holder of the coin, and U has not received a reassignment request from Z. Invariant 2 may or may not hold initially, but given the mechanism described earlier to enforce correct behavior from owner U, eventually invariant 2 will hold. For invariant 3, by the inductive hypothesis, U has proof of relinquishment from the first $n-1$ holders of C. Y was the nth holder, and in the previous step, Y sent a reassignment request R_{UYZ} to U. U therefore has proof of relinquishment from the first n holders of C, which are all the previous holders of C – therefore, invariant 3 holds as well.

Case 2. Z has held this coin before. Y sends the following request to owner U:

$$R_{UYZ} = \{Y, ts, A_{UY}\}_{SK_Y} \tag{17}$$

All three invariants hold (or can be enforced) in the same way as in case 1. When owner U responds with the new assignment:

$$A_{UZ} = \{Z, seq_n, ts, C\}_{SK_U} \tag{18}$$

then again, invariants 2 and 3 hold, or can be enforced, in the same way as in case 1. Invariant 1 is slightly trickier, because U *does* have proof of relinquishment from Z for coin C, but for a *previous assignment* of C to Z. Here, we see the importance of invariant 2: each assignment of a given coin has a unique sequence number, and a reassignment request (e.g., R_{UYZ}) contains the assigned coin (e.g., A_{UY}), which in turn contains the sequence number. Hence, owner U can not use an old reassignment request to refute a new, valid assignment. As a result. invariant 1 holds as long as invariant 2 holds at all times.

Analysis of Downtime Protocol. The third invariant does not necessarily hold if the owner of a coin is down, and a holder goes to the broker for reassignment, because now the broker has proof of relinquishment of an assignment that the owner needs. Instead, we use a weaker (but sufficient) form of the invariant, which states that if H is *not* the valid holder holder of coin C, then an *omniscient party* O can refute any assignment that H claims with proof of relinquishment. By "omniscient party", we are referring to a (theoretical) party with access to all proofs of relinquishment relating to coin C, whether these proofs reside at the broker or owner, or both. If the broker is not storing any such proofs, the owner of the coin is an omniscient party. If the owner is currently down, and the broker reassigns coin C, then in practice, an omniscient party does not exist (but the invariant can still hold). Once the owner of C comes back online, however, it once again becomes an omniscient party, and can retroactively detect any misconduct regarding C.

In terms of state, the *downtime protocol* for reassignment (in which the broker reassigns the coin) is identical to the *default protocol* (in which the owner reassigns the coin) except that the proof of relinquishment is stored at the broker. Hence, at every step in the downtime protocol, an omniscient party O has the same state that owner U has at the analogous step in the default protocol. It is therefore clear that the weaker form of invariant 3 holds, by the same arguments used for the default protocol.

B Security of Layers

Because layers over the coin are now used to encode reassignment, an owner U may no longer have the full audit trail of a coin. Invariant 3 as originally stated no longer holds. For example, say node V holds coin C owned by node U. V may reassign the coin to node X using layers, and then reassign the coin again to node Y using layers. U cannot detect V's misbehavior, because U is no longer involved in the reassignment.

However, we can show that the weaker form of invariant 3 still holds. Let us assume that node V is the nth holder of coin C owned by U, that V is the current valid holder, and that the three invariants hold. Say V now reassigns the coin to X via the layered coin P_{UVX}. Invariant

1 holds by the same argument used for the default protocol. Invariant 2 may not hold if V uses a bad sequence number, but X can detect this case and report misconduct to the broker. The broker will punish V and require it to produce a valid layered coin; hence V has no incentive to attack in this manner. Invariant 3 holds because an omniscient party now has access to the proofs stored as layers over a coin, as well as the proofs stored at the broker and the owner. We are given that the omniscient party can refute the first $n-1$ holders of C, and P_{UVX} now contains the proof of relinquishment by the nth holder. Therefore, the omniscient party can refute all old assignments of coin C.

Owner U becomes omniscient when the layered assignments of C are peeled at U (only one assignment of a coin should exist, but in the case of misconduct, then multiple assignments might exist). Unlike with downtime, owner U has no control over when it becomes omniscient. However, given that all coins must eventually be peeled to be cashed, such a time is guaranteed to exist. Furthermore, we can "hasten" this time by imposing a limit on the number of layers a coin may hold. Alternatively, coin renewals (described below) produce the same effect.

Finally, note that it is possible for the holder of a coin to "illegally" peel off layers of the coin, and proofs may be lost in this manner, thereby violating invariant 3. However, it is simple to detect and prove this type of misbehavior. Let us say V is the current valid holder of coin C, and that the assignment to V was made via a layered assignment P_{UTV} from node T. Note that the first field in an assigned coin (whether or not it is layered) is the recipient of the assignment. If V peels a layer off P_{UTV}, V will end up with a coin that is assigned to T. V can not change the recipient of the coin because we assume V can not forge the signature of whoever assigned the coin to T. Therefore, V can not use the peeled coin. If for some reason, V does peel the coin and tries to use it or cash it, the owner can prove that V misbehaved, and V will be punished. Furthermore, although invariant 3 no longer holds, V will be held financially responsible for any cheating that can be detected but for which no culprit can be identified (i.e., because the proofs were lost by V's peeling).

There is also the possibility that V did previously hold the layered coin, and V can peel off layers until it produces a layered coin assigned to itself. For example, consider the scenario in which the assignment A_{UV} is reassigned via layers as follows: $V \rightarrow X \rightarrow Y \rightarrow V$. The second time V receives the coin, it can peel three layers and reproduce A_{UV}. However, V has gained nothing by doing so – it could have easily reproduced A_{UV} by replicating that assignment. We have already discussed in Section 4.1 how replicating A_{UV} is not helpful, because it can not be reused. Therefore, U has no incentive to attack in this manner.

C Security of Coin Renewals

After a renewal request, the holder V must check that the renewed coin has a renewal period in the future. Furthermore, ideally the renewal period adheres to a system-wide constant set by the broker. Because renewals present a tradeoff between performance, state and security (discussed below and in Section 7.2.2), renewal windows that are unnecessarily small will result in high loads for peer and broker, while renewal windows that are unnecessarily large will present security risks – especially if it is a malicious node who is trying to assign a long renewal period! Hence, in PPay we require that the renewal period be fixed for all coins. V must check that the new renewal period is correct after a renewal, and that the renewal period has not changed after a reassignment. If owner U tampers with the renewal period, V can provide the original and reassigned/renewed coin to broker B as proof of misconduct on U's part.

To prove Theorem 1, we change the definition of a valid assignment to now include the restriction that the assignment has not expired. Furthermore, "proof of relinquishment" of an assignment is extended to include a renewal request for that assignment. Under this definition, we must show that all three invariants still hold under the coin renewal protocol.

Earlier, we showed how if the invariants held before the $n+1$th reassignment, the will hold after the reassignment as well. Here, we show how if the invariants hold before a renewal request, they will hold afterwards as well.

Say node V is the valid holder of a coin C owned by U, and all three invariants hold. Now say V requests to renew its assignment A_{UV} via the renewal request:

$$N = \{A_{UV}\}_{SK_V} \tag{19}$$

At this point V is still the valid holder, but U possesses proof of relinquishment (i.e., the renewal request). This problem is analogous to the case in which V is having the coin reassigned, and submits a reassignment request. We deal with the problem in the same way, by having V go to the broker for renewal. Since U will be charged and can not subsequently use the coin, it has no incentive to attack in this way. Eventually, invariant 1 will hold.

Invariant 2 holds because A_{UV} is still a valid assignment, and the serial number has not changed. Likewise, invariant 3 holds because the valid holder has not changed.

Now, owner U responds with the renewed coin:

$$A_{UV} = \{V, seq_n, ts, r'_b, r'_e, C\}_{SK_U} \tag{20}$$

Invariant 1 holds because owner U does not have proof that A_{UV} relinquished the renewed coin (distinguished from the old coin by the new renewal periods). Invariant 2 may not hold if U has changed the sequence number; however, just as with the new renewal period, V is responsible for checking the correctness of the sequence number and going to the broker if the value is incorrect. Eventually, then, invariant 2 will hold. Invariant 3 holds because the valid holder of the coin has not changed as a result of the renewal.

To show the correctness of the renewal policy, we also need to show that any proofs of relinquishment for expired assignments are not required for the three invariants to hold. Because destroying proofs can not create evidence against a valid holder, invariant 1 holds. Furthermore, destroying proofs does not change the serial number of the current assignment, so invariant 2 holds. For invariant 3, any assignment that has expired is invalid, and the proof of the invalidity lies within the coin itself (the renewal end time, r_e). Hence, an omniscient party can automatically refute any assignment that has already expired. For any assignment that has not expired, but that has been relinquished (either reassigned or renewed), the omniscient party has not destroyed the proof of relinquishment. Hence, invariant 3 holds as well.

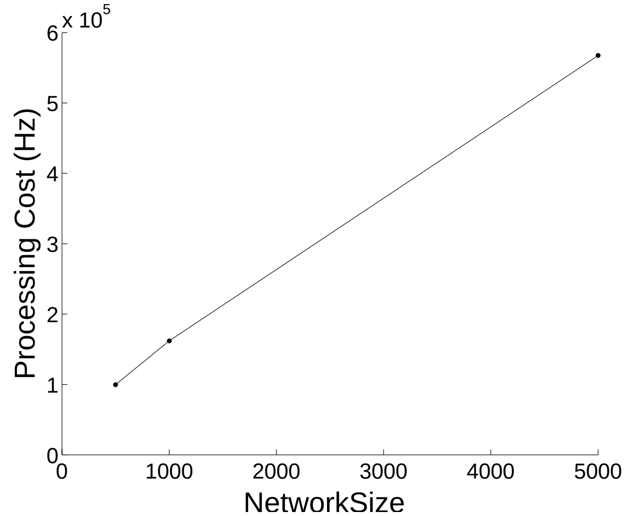

Figure 10: Broker load scales sublinearly as the number of peers in the system grows

D Hash Chains

Hash chains seem perfect for picopayments because after an initial setup cost, each payment requires only a one-way hash. Two users A and B can set up a hash chain to denote A's payment to B, which requires a signature generation and verification. Each time A wishes to make a picopayment to B, it sends B the next "link" in the chain. A can verify this payment via a single hash. The chain has a pre-determined length, and once all links have been sent, A makes a real coin payment to B. Further details of hash chains can be found in [14].

Unlike soft credit windows, if the broker functionality is extended to honor partial Payword chains, then Payword presents no risk. Payments can be proven by the links the hash chain, and partial chains can be redeemed when a peer permanently leaves the system.

However, using hash chains for a P2P application presents many practical issues. First, maintaining hash chains has high state overhead. Like soft credit windows, hash chains require peers to keep state per node they interact with. Unlike soft credit windows, however, this per-peer state is very large. Again let us assume that a peer interacts with a million other peers. If each hash "link" is 160 bits (the size of a SHA-1 digest), each chain consists of 100 links, and each peer has two hash chains per other peer (one to denote credit given, another to denote credit received), then total state will be an average of 4GB, which is *very* significant. Rather than storing full hash chains, peers can instead store only the base of the hash chain, and perform i hashes to reconstruct the ith link in the chain. However, doing so results in a processing load equivalent to one signature verification per picopayment, if a chain is 100 links long, which largely neutralizes the benefits of hash chains (see Section 6.3 for a discussion on calculating action costs). Although shorter hash chains can proportionally reduce state or processing cost, we will see later (Section 7.2.3) that they also result in significantly worse performance.

Therefore, hash chains are only really appropriate in applications where each peer only interacts with a small number of other peers. Even in this case, however, soft-credit windows can be much more efficient, due to the balanced relationships often observed between peers. Hash chains are a one-way payment mechanism that can only express an *increasing* amount of credit, rather than increasing or decreasing. Hence if users A and B exchange an infinite but balanced number of picopayments, they will need to exchange an infinite number of coins, rather than zero coins. In Section 7.2.3 we will investigate more closely the effect of this observation on the performance of hash chains relative to soft credit windows.

E Scalability

Figure Figure 10 shows broker load as the number of peers (parameter `Network Size`) is varied along the x-axis. Load is shown in terms of processing cost, but similar behavior is observed for bandwidth load as well. Similarly, Figure 11 shows peer processing load as `Network Size` is varied. All parameters in these experiments are set to default values, as given in Table 1.

F Extracting Online/Offline Data

In reference [15], a set of IP addresses was monitored over a 60 hour period. Reference [15] noted when a peer at a given IP address was online (and reachable from the

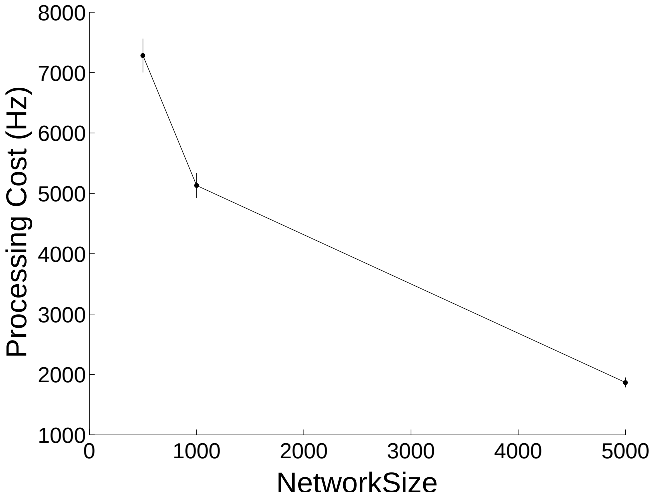

Figure 11: Peer load decreases as the number of peers in the system grows

Description	Value
Expected length of query string	12 B
Average size of result record	76 B
Average size of metadata for a single file	72 B
Average number of queries per user per second	$9.26 \cdot 10^{-3}$

Table 3: General Statistics

measurement machine), and had an open socket on the Gnutella port. Given this setup, it is impossible to determine the length of online or offline periods that began before the observation window began, or ended after the observation window ended. For example, if a peer is detected to be online when the observation begins, there is know way of knowing how long the peer was on prior to that moment.

For our simulations, we only used those online and offline periods that had both begin and end points inside of the observation window. Although this automatically precludes any intervals larger than 60 hours, hopefully the *ratio* of online to offline times is preserved – the ratio is the most important factor in the performance results, as opposed to the absolute values. Furthermore, we can tweak this ratio by setting the `Lifespan Multiplier` and `Downtime Multiplier` parameters appropriately.

G Atomic Action Costs

In terms of bandwidth, the cost of an action is the number of bytes being transferred. We define the size of GUESS-related messages according to their analogs in the Gnutella protocol, since no formal GUESS message protocol has yet been defined. For example, query messages in Gnutella include a 22 bytes header, a 2 byte field for flags, and null-terminated query string. Total size of a query message, including UDP/IP and Ethernet headers, is 76 + query string length. Some values, such as the size of a metadata record, are not specified by the protocol, but are a function of the type of data being shared. These values, listed in Table 3, were gathered through observation of the Gnutella network over a 1-month period, as described in [17].

We define the size of micropayment-related messages according to the protocol defined in this paper. For example, a limit certificate consists of an 8-byte peer identifier, a 2-byte limit, and a 128-byte signature certificate. Total size of the message transferring this limit certificate, including headers, is 190 bytes.

The processing costs are given in coarse units, and were determined by measurements taken on a Pentium III 930 MHz processor running Linux kernel version 2.2. Processing costs will vary between machines and implementations, so the values seen in Table 2 are meant to be representative, rather than exact. A unit is defined to be the cost of sending a receiving a Gnutella message with no payload over TCP, which was measured to be roughly 7200 cycles on the measurement machine. When displaying figures in the following sections, we will convert these coarse units to cycles using this conversion ratio.

H EC Outperforming PPay

The relative performance of PPay degrades as coins are used fewer times before being cashed. Coins are used fewer times if owners are often offline, or the rate of buying and selling for a given peer is unbalanced (e.g., the peer spends more than it buys). In our simulations, a reasonable degree of imbalance and instability is already modeled: some peers have many more files than others (from 0 to over 70 thousand files), and some peers are online for much longer periods of time than others (from a few minutes to over 1000 hours). However, for argument's sake, let us consider the case in which coins are used exactly once before being cashed. Technically, in this scenario PPay degenerates exactly into the EC scheme. However, because coins in PPay include limit certificates (which EC does not need) they are larger.

Table 4 shows the cost of the two schemes when coins are used exactly once before being cashed. As we can see, broker load in terms of incoming and outgoing bandwidth is now 35% higher in PPay than in the EC scheme. However, in terms of processing cost, broker load in PPay is only 6.6% higher. Hence, even in the most extreme case, PPay performance is still comparable to the performance of EC. In the experiment described by Table 4, we note that electronic lottery tickets (an important aspect of the EC scheme of [11]), are not being used in either case, but can be applied to equally reduce the broker load in both.

However, in a complete comparison between EC and PPay, we also note that additional features are possible in EC due to the foreknowledge that coins will be "im-

Scheme	Bandwidth Cost (bps)	Processing Cost (Hz)
PPay(*)	$5.59 \cdot 10^3$	$3.82 \cdot 10^6$
EC	$3.75 \cdot 10^3$	$3.59 \cdot 10^6$

Table 4: Broker loads in EC and PPay, in the worst-case scenario for PPay

mediately" cashed. In particular, while electronic lottery tickets can be used in both schemes to equally reduce broker load, in the EC scheme, the broker can also implement the MR2 extension described in [11] to eliminate a buyer's risk of paying more than he should (for a discussion of client risk using electronic lottery tickets, we refer readers to the citation). This result is consistent with the fact that clients in PPay must manage more risk than in a completely centralized system.

I Multiple Denominations

Because PPay coins are designed to float, they will typically be issued in fixed denominations. We note that users in PPay *can* issue coins of any integral denomination, simply by associating multiple serial numbers with a single coin. However, in applications where the value of goods vary widely, it is unlikely that a user can reuse a single existing coin to exactly pay for another good; instead, the user would have to pay using several coins. If users instead issue new coins every time a payment needs to made, then PPay degenerates to the default case where the broker load is linear in the number of transactions. Therefore, PPay has the greatest advantage in applications where the value of goods are aligned with a few fixed denominations.

Important future work lies in how we can support goods with widely varying values, while simultaneously allowing coins to float and be reused. One part of the solution may be to allow coins to be "split" into smaller coins, or multiple coins from the same owner to be "coalesced" into one large coin. Such functionality can be easily added to PPay by treating each serial number as a single unit in the audit trails, although many serial numbers can be lumped together in a single coin. It remains to be seen whether or not this kind of flexibility will allow most purchases to be made using existing coins, and new coins can be issued only for the remaining purchases.

J Member Certificates

Each peer is assigned a certificate upon opening an account with the broker. Like membership certificates described in [14], these certificates must have the following information:

$$cert = \{U, address_U, PK_U, expiry\}_{SK_B}$$

We also assume that all peers in the system have a way in which to obtain the public key of the broker over some authenticated channel.

Whenever a peer V pays U by reassigning a coin owned by O, U must check that O is in the system under any one of the two following conditions:

1. U has never interacted with O before.
2. U has interacted with O in the past, but the last known certificate for O has expired.

Note that we require U to keep track of peers it has interacted with (along with the expiration date of the certificate). If we have a stable user base, then we can greatly reduce the number of times U must check O's certificate.

Whenever a peer V pays U by reassigning a coin via layers, U must check that V is in the system under the same two conditions. If U ever accepts a coin from a peer V that is not in the system, U must sustain the loss. If U reassigns such a coin, U is responsible for fulfilling the value of the coin when the error is discovered.

Now, if V is not in the system, but somehow obtains a coin, he will either have to have it reassigned through owner O, or via layers. If the coin is reassigned through layers, U will not accept the layered coin from V, because U will check V's certificate and find it to be invalid (or non-existent). If the coin is reassigned through owner O, it can only be spent once, and no harm is done.

However, we missed a step in the non-layered case: in order learn the public key of V (the peer making the payment), in order to verify the signature on the reassignment request, owner O must also check V's certificate. Otherwise, V could generate a reassignment request with a false public/private key pair, and then another reassignment request with the real public/private key pair, and O could not prove that he was "tricked" by the first request. Therefore, assuming no cached knowledge of past interactions, 2 certificates (as opposed to 1) must be checked in a non-layered coin reassignment.

We make three observations about the additional cost of certificate-checking. First, in any offline electronic check scheme, a party receiving a payment needs to check the certificate of the party making the payment, or otherwise risk not getting paid in the end. Hence, for every payment in a generic lottery ticket scheme (such as the EC scheme), 1 certificate verification is needed under the same two conditions listed earlier. Second, our focus is on broker load, and certificate-checking is purely client load. Finally, a stable user base will greatly reduce the number of certificate checks needed in PPay or EC, or any other scheme using digital signatures. Given that peers have an incentive to stay with a system if they can make a profit, we do expect a reasonable amount of a stibility in a system using micropayments.