Distributed Aggregation of Micropayments*

Jan Chomicki †
Dept. of Computer Science
Monmouth University
Howard Hall
W. Long Branch, NJ 07764
chomicki@moncol.monmouth.edu

Shamim Naqvi
Bell Labs
Lucent Technologies
600 Mountain Ave.
Murray Hill, NJ 07974
shamim@lucent.com

July 31, 1998

Abstract

We propose a novel protocol for aggregating micropayments in a networked environment. The protocol is based on the idea of debt consolidation and is fully distributed. We propose client-server and serverless versions of the protocol. We also analyze the mathematical properties of the protocol. Finally, we show how basic cryptographic techniques can be used to support the operation of the protocol in an untrusted environment.

Keywords: electronic payments, billing, aggregation, protocol

1 Introduction

The rapid growth of computer networks and the popularity of network-based applications like email or WWW have dramatically increased the number of interactions between network users. Many of those interactions are commercial in character and result in purchases of goods of various kinds. The problem of billing for such purchases is one of the central problems of the emerging area of *electronic commerce* [12]. Especially difficult is the issue of billing for *micropayments:* transactions of a very low monetary value (several cents or less). Micropayments are, however, potentially ubiquitous: A WWW page hit at a

*A preliminary version of this paper appeared in [4].

†Contact author. Phone: (732) 571-4457. Fax: (732) 263-5202. Work done while visiting Lucent Bell Labs and Bellcore.

commercial provider, the routing of a packet, the execution of a custom program (e.g., a Latex to HTML translator), or the collection of electronic royalties may be viewed as such. Currently, micropayments are not billed individually because of the lack of appropriate billing mechanisms. The cost of billing itself exceeds the monetary amount involved in such a transaction[1]. Recent approaches [1, 8, 9, 11, 13], mostly based on cryptography, attempt to rectify this situation by making the billing operations very lightweight.

Example 1.1 Consider the case of a copyrighted dictionary. If it is put online, how much could a customer be charged for a word lookup? Assume that the dictionary contains 100,000 entries. Then charging .001c for a word lookup gives the user the ability to look up every word in the dictionary ten times for the total price of $10, still below the price of a hardbound volume.

In this paper we propose a new protocol for billing for micropayments which, instead of making billing lightweight, minimizes the number of billing operations. Our approach is unique in that it does not require any intermediaries, e.g., banks or brokers, between customers and vendors. It is enough that customers and vendors use a reliable identification service and a common arbitrator. In any payment schema our approach reduces or completely eliminates the need for a separate settlement phase, so we believe that it is applicable not only to micropayments.

We discuss two variants of the protocol: trusted and untrusted. In the first we assume that the vendors can be trusted. This is a reasonable assumption as long as we are dealing with a single, although possibly very large, organization (a service provider like a telephone carrier, a bank, an association of merchants etc.). In such an organization trust can be enforced using the standard administrative procedures. In the untrusted variant basic cryptographic techniques are used to protect the integrity of the protocol.

The central idea of our approach is the decoupling of purchases and billing. Let's call the vendor with whom a customer has made a purchase the *primary* vendor of the purchase. There are clearly various degrees in which purchasing and billing can be coupled:

- a customer is always billed for a purchase by the primary vendor of this purchase, possibly through an intermediary (this will be called *tight* coupling);
- a customer is always billed for a purchase by the primary vendor of this *or some other* purchase of the same customer (*conservative* coupling),
- any vendor can bill any customer, provided of course every customer is accurately billed for his purchases and every vendor receives the expected remunerations for the goods sold (*liberal* coupling).

[1]This is true even for electronic billing as every billing operation has some fixed cost associated with generating the appropriate paperflow. As an example, consider the cost of posting a credit card charge.

All payment systems proposed to date are based on tight coupling. In contrast, our approach supports both conservative and liberal coupling. Some of the desirable properties of tight coupling can still be recovered. E.g., in the cryptographic version of our protocol it is possible for a customer to obtain, in addition to a summary bill, a detailed record of his purchases. This may be useful, for example, in a situation where a customer has made purchases with the ACM and a music store. He may accept being billed only by the music store, as long as he has a record of his ACM purchases to support his tax deduction for educational expenses.

We propose that a customer's purchase be represented as his debt. The debts of a customer are aggregated before he is billed. The aggregation is done, however, without any central debt information repository (bank, broker etc.) but just by swapping customers' debts between vendors until the debts become large enough to be collectable. As long as the swapped monetary amounts are the same, the accuracy of billing is preserved.

Example 1.2 Suppose that customers C_1 and C_2 each owe 1 monetary unit to each of the vendors V_1 and V_2. In this situation, 4 billing operations are necessary. However, if V_1 swaps with V_2 the debt of C_2 for that of C_1, V_1 will only bill C_1 for 2 units, the same for V_2 and C_2. The number of billing operations is thus reduced to 2.

We argue that the aggregation of debts by individual vendors is not sufficient because one customer's charges with a specific vendor might not be large enough for billing. However, one customer's charges across many different vendors should be sufficiently large. Therefore, aggregation across many different vendors seems essential and this is what our approach achieves. For example, in the WWW world a customer may have made a hundred or so page hits at a specific vendor's site. If we assume .01c per hit, this amounts to the total charge of 1 cent. The customer may have, however, visited hundreds of sites, so his total charge for Web surfing may well be several dollars – a billable amount. (Aggregation is more difficult in an untrusted environment because the identity of individual purchases needs to be preserved and safeguarded.)

The summary of the paper is as follows. In section 2 we introduce the basic ideas of our protocol, discuss the implementation issues, and analyze its mathematical properties. In section 3 we show in detail how the protocol can be adapted to an untrusted environment, using basic cryptographic techniques. We discuss related work in section 4 and draw conclusions in section 5.

2 Basic swapping protocol

We first describe the version of our protocol in which vendors can be trusted.

An *agent* is an autonomous entity capable of interacting with other agents over the network. A *customer* is an agent interested in purchasing goods. A *vendor* is an agent offering goods for sale. (The nature of the goods is of no concern to us.) We assume that there is a universe of agents U, a set of active agents $Act \subseteq U$, a set of customers $C \subseteq Act$, and a set of vendors $V \subseteq Act$. For simplicity, we assume that $C \cap V = \emptyset$, $C \cup V = Act$, $|C| = m$ and $|V| = n$. Customers are numbered from 1 to m and vendors from 1 to n. Money changes hands in multiples of a single monetary unit, thus money amounts are integer values. (The money value of a monetary unit can in principle be made arbitrarily small. In practice, however, this value should be determined by the cost associated with a specific version of the protocol. In this respect the cryptographic version is considerably more expensive.)

When involved in purchases and sales, agents follow the same protocol which assigns different roles to customers and vendors. Customers make purchases with specific vendors. Vendors bill specific customers, giving them only a summary bill where individual purchases are not itemized. Additionally, vendors can swap customer debts between themselves.

Each vendor keeps an array of m elements indexed by customer numbers, which represents the current debts of individual customers. This array may be viewed as a column in an $m \times n$ *debt matrix* D representing the global state of the system. The meaning of $D_{ij} = k$ is the following: "the customer i owes the vendor j exactly k units." The elements of the debt matrix are nonnegative integers.

We describe now the basic operations of the protocol:

- $purchase(i, j, k)$: customer i purchases from vendor j the goods worth k monetary units,
- $bill(i, j, k)$: customer i is billed by vendor j for k monetary units,
- $swap(i_1, j_1, i_2, j_2, k)$: vendor j_1 swaps with vendor j_2 k units of debt of customer i_2 for k units of debt of customer i_1.

The basic operations are formally specified below using the appropriate pre- and post-conditions. (The values after an operation are primed.)

$purchase(i, j, k)$
pre $k > 0$
post $D'_{ij} = D_{ij} + k$

$bill(i, j, k)$
pre $k > 0 \wedge D_{ij} \geq k$
post $D'_{ij} = D_{ij} - k$

$swap(i_1, j_1, i_2, j_2, k)$
pre $i_1 \neq i_2 \wedge j_1 \neq j_2 \wedge k > 0 \wedge D_{i_2,j_1} \geq k \wedge D_{i_1,j_2} \geq k$
post $D'_{i_1,j_1} = D_{i_1,j_1} + k \wedge D'_{i_2,j_2} = D_{i_2,j_2} + k$
$\wedge D'_{i_2,j_1} = D_{i_2,j_1} - k \wedge D'_{i_1,j_2} = D_{i_1,j_2} - k$

The protocol is fully distributed and does not require any central authority. The purchase and billing operations are executed by vendors at their sites and update their local debt information. We assume that there is underlying software that performs those operations reliably. While it is important that no billing or swap operations are lost, perfect bookkeeping of purchases is not necessary as long as the monetary amounts involved are minuscule. (How the purchased goods are delivered and the billed amount collected are issues that are outside of the scope of our protocol.) Swapping can be implemented in several different ways. In this paper we discuss two basic approaches: one that uses a swapping server and a serverless one.

All the operations of the protocol can be performed offline, provided a simple online data gatherer, e.g., a logger, is available. Each vendor separates the current purchases of a customer from his past debt used for billing and swapping. Only the former is updated online, the latter is kept offline and occasionally refreshed. The information about current purchases may be kept in an append-only log of all purchases at the vendor's site (similar logs are routinely maintained by WWW servers). Only the offline information is used in billing and swap operations.

2.1 Correctness

Definition 2.1 *(Sum-equivalence). Two $m \times n$ debt matrices D_1 and D_2 are* sum-equivalent *if for every column in D_1 its sum is equal to the sum of the corresponding column in D_2 and the same for the rows.*

Clearly, any swap operation transforms a debt matrix D to a matrix which is sum-equivalent to D.

We assume here that the global state is changed only by the operations listed above. However, any interleaving of operations is valid. If this is the case and initially $D_{ij} = 0$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, then the system satisfies two invariants that are essential for the overall correctness:

Truth in billing: No customer can be billed for more than his total outstanding debt.

Recovery of money: Every vendor can bill for the full amount of purchases made with him (although not necessarily the original purchasers).

These two invariants are sufficient to support liberal coupling of purchases and billing. If conservative coupling is desired, an additional invariant needs to be maintained:

No surprises: The debt of a customer with a vendor can become nonzero only through a purchase.

This invariant means that no customer will be billed by unknown vendors.

2.2 Debt consolidation

While purchase and billing operations are fairly routine, debt swapping constitutes the major new idea of our approach. We show how swapping can lead to *debt consolidation*. Purchase and billing operations modify the debt matrix and therefore require repeated debt consolidation. We leave this issue for future research.

We assume that every customer incurred a debt with at least one vendor and each vendor had at least one sale. This does not affect the generality of our results as customers without debts and vendors without sales do not participate in debt consolidation.

Definition 2.2 *(Consolidation Ordering). For two $m \times n$ sum-equivalent debt matrices D_1 and D_2, D_1 is* more consolidated *than D_2, in symbols $D_1 \prec D_2$, if the number of nonzero elements in D_1 is less than in D_2.*

We will call debt matrices that are minimal in $\prec$ simply *minimal*. Minimal matrices are desirable from the point of view of debt consolidation. A minimal matrix represents the maximum debt consolidation possible given a specific debt matrix. Billing operations can thus be fewer and deal with larger monetary amounts.

Example 2.1 Given a matrix

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

there are two minimal matrices that are sum-equivalent to it:

$$\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Any 2×2 matrix consisting of nonzero elements only is not minimal (in such a matrix it is always possible to introduce a zero through a swap). However, there are 2×2 matrices with only one zero that are minimal like

$$\begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}$$

We present now a number of approaches to debt consolidation. The first is based on a global consolidation of the debt matrix. We show how it can be implemented using a *swapping server*. In the second *serverless* approach debt consolidation is achieved in a fully distributed way.

2.3 Global consolidation

Cycle Breaking Reduction

The debt matrix D may be viewed as a undirected bipartite graph G_D with weights. Every nonzero element D_{ij} of the matrix is represented as the edge (i, j) with a positive weight equal to D_{ij}.

Definition 2.3 *(Cycle Breaking Reduction (CBR)). If there is a cycle W of length at least 4 in G_D, select an edge in W with a minimum weight w. Remove this edge, and alternately add w to and subtract it from the weights of the edges in the cycle W. (If the weight of an edge becomes 0, the edge is removed.) The resulting graph G' is called a* cycle breaking reduct *of G_D. G' corresponds to a matrix D' which is called a* cycle breaking reduct *of D.*

Notice that if there are two edges with the same minimum weight in a cycle, or more than one cycle, different cycle breaking reducts may be obtained. Notice also that CBR guarantees **No surprises**.

Example 2.2 An example of CBR is shown in Figures 1, 2 and 3. Figure 1 shows the debt matrices before and after Cycle Breaking Reduction. The first matrix corresponds to the graph in Figure 2, the second to the graph in Figure 3. The reduction removes the edges (c_1, v_1) and (c_2, v_3).

$$\begin{bmatrix} 5 & 8 & 0 \\ 10 & 0 & 5 \\ 0 & 7 & 5 \end{bmatrix} \qquad \begin{bmatrix} 0 & 13 & 0 \\ 15 & 0 & 0 \\ 0 & 2 & 10 \end{bmatrix}$$

Figure 1: Debt matrices: before and after CBR

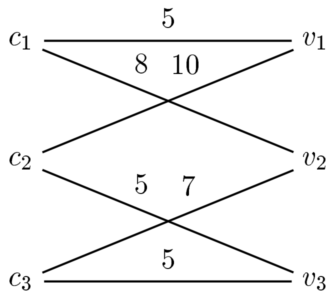

Figure 2: Before CBR

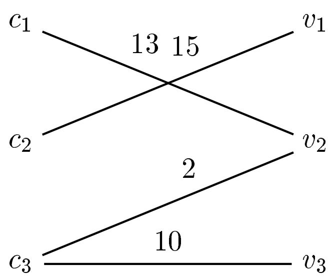

Figure 3: After CBR

Theorem 2.1 *If D' is a cycle breaking reduct of D, then D' is sum-equivalent to D and $D' \prec D$.*

Proof: G_D is a bipartite graph and thus every cycle has even length. Thus for every column and row in D, every weight decrease during cycle breaking is compensated by a corresponding increase and vice versa. Also, by the choice of the edge to be removed no elements of the debt matrix become negative. ∎

Theorem 2.2 *For any $m \times n$ debt matrix D, by iterating CBR, one can obtain a debt matrix with at most $m + n - 1$ nonzero elements.*

Proof: If the reduction is not applicable to D, the graph G_D does not have a cycle and is a spanning forest. Thus it has at most $m + n - 1$ edges. ∎

Theorem 2.2 means that the savings obtained by CBR can be quite spectacular. To appreciate that consider $m = 100000$, $n = 10000$ and assume that the debt matrix is 10% full with all the elements equal to 1 unit. In this case, the matrix constructed by the above algorithm has less than 110000 nonzero elements and the *average debt* (the average of all the nonzero elements) is around 900 units. Thus consolidation results in almost a thousandfold increase in the value of average debt! So if one unit corresponds to 1c, the average consolidated debt is around $9.

Theorem 2.2 provides also an upper bound on the number of nonzero elements in any minimal $m \times n$ debt matrix.

Theorem 2.3 *Iterating CBR to obtain a cycle-free debt graph G_D takes $O(e^2)$ time where e is the number of edges in G_D.*

Proof: To remove a single edge one has to find a cycle in G_D and on it an edge with minimum weight, and then traverse this cycle updating all the edge weights. The length of the cycle is at most $m + n$. Finding a cycle can be done using depth-first search in $O(max(m + n, e))$ time where e is the number of nonzero elements in the original matrix D. However, because we assume that every customer and vendor is involved in at least one purchase, we have that $e \geq max(m, n)$. Thus CBR is performed in $O(e)$ time and it takes $O(e^2)$ time to obtain a matrix that cannot be further reduced. ∎

The number of nonzero elements in a minimal matrix is at least $max(m, n)$. So the upper bound from Theorem 2.2 is always at most *twice* the number of nonzero elements in a minimal matrix. Thus iterated CBR is guaranteed to produce a matrix close to a minimal one.

Because there are only finitely many matrices sum-equivalent to a given debt matrix (here the assumption that matrix elements are nonnegative integers is essential), a minimal matrix can in principle be found by exhaustive search, but this is clearly not practical.

Swapping server

While in a distributed enviroment no site is capable of keeping the whole debt matrix, we propose that specialized sites called *swapping servers* keep fragments of the debt matrix and perform global consolidation on those fragments. A fragment of an $m \times n$ debt matrix

D is a matrix whose rows are indexed by a subset of $\{1, \ldots, m\}$ and columns by a subset of $\{1, \ldots, n\}$. Moreover, the fragment elements have to represent real debt and no debt can be represented more than once. (In the trusted case, this condition is enforced if all the agents follow the basic swapping protocol. In the untrusted case, cheating may violate it but will be detected.)

A vendor periodically (once every billing period) sends to a swapping server a column of the aggregated charges of many customers. A server, after having received such columns from many vendors, has now a fragment of the debt matrix and performs global consolidation on it. After the consolidation, the columns are returned to the vendors who have now the option of billing the customers directly, referring the bills to a specialized billing server, or further postponing the billing - in the case the amounts to be collected are too low. It is the vendors' responsibility to make sure a column goes only to a single swapping server.

Swapping servers are not brokers or banks required by other protocols for micropayments [1, 8, 9, 11, 13] because they only perform on-demand services for vendors, without entering with them into any long-term relationships. In particular, the servers do not maintain vendor accounts.

We have implemented the debt consolidation module of a simple swapping server and tested it on a variety of inputs. The module uses a hybrid approach that combines consolidating swaps (Definition 2.4) with CBR. For a given debt matrix it executes in three stages. In the first stage, all consolidating swaps are performed; in the second, all cycles of length 6 are broken; in the third, the remaining cycles are broken.

The debt consolidation module is implemented in Java. It was tested under JDK 1.1 and Linux, on a 200MHz Pentium Pro with 64MB of memory. In selecting test data (Tables 1 and 2), we were guided by several basic assumptions. In each table the ratio of customers to vendors is constant. Also the number of nonzero entries in each row of the debt matrix is constant in each table (this is supposed to reflect the fact that every customer has a fixed budget which does not depend on the number of vendors). Consequently, the *density* of nonzero entries (the number of nonzero entries in the matrix divided by mn) is inversely proportional to the number of customers and the reduction in the number of nonzero entries in the resulting matrix is fixed for each table. In Table 1 the reduction is 7 times (7X), while in Table 2 it is 8 times (8X).

The experimental results indicate that for sparse debt matrices the execution of the swapping server takes time that grows *quadratically* with the number of customers. (This can also be confirmed analytically.) They also suggest a very simple way to speed up the server using parallel processors. If k processors are available, then an $m \times n$ debt matrix can be split into k matrices of dimension $(m/k) \times n$ and each smaller matrix shipped to a different processor. For the data in Table 2 using $k = 10$ processors will give an order of magnitude speedup (compare the corresponding rows in Table 1) and reduce the memory requirements of a single processor, at the price of obtaining a slightly inferior 7X reduction instead of an 8X one.

Number of customers ($m = 10n$)	Number of vendors (n)	Density	Elapsed time (in seconds)
1000	100	8%	2
2000	200	4%	10
4000	400	2%	39
8000	800	1%	175

Table 1: Ratio 10:1, 7X reduction

Number of customers ($m = 10n$)	Number of vendors (n)	Density	Elapsed time (in seconds)
10000	100	8%	27
20000	200	4%	114
40000	400	2%	459
80000	800	1%	out of memory

Table 2: Ratio 100:1, 8X reduction

2.4 Distributed consolidation

Debt consolidation can also be achieved using a fully distributed approach. In this case, no sites are designated as servers but rather vendors swap debts between themselves. No coordination is necessary which means that (pairs of) vendors can swap independently. The challenging issues are: *progress* in debt consolidation and *termination*.

Definition 2.4 *(Consolidating Swap) An operation $swap(i_1, j_1, i_2, j_2, k)$ is called a* consolidating swap *if it satisfies an additional precondition:*

$$|D_{i_2,j_1} - D_{i_1,j_2}| < D_{i_1,j_1} + D_{i_2,j_2} \text{ and } k = min(D_{i_2,j_1}, D_{i_1,j_2}).$$

The additional precondition on consolidating swaps prevents back-and-forth and in-place swapping.

Example 2.3 Given a debt matrix D

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

the following debt matrix D' can be obtained through the consolidating $swap(1,1,2,2,1)$:

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

However, the reverse $swap(1,2,2,1,1)$ that produces D is not consolidating.

Similarly, no consolidating swap is applicable to the minimal matrix

$$\begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}$$

Example 2.4 To see that the requirement $k = min(D_{i_2,j_1}, D_{i_1,j_2})$ is necessary, consider the following matrix D

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

Applying a $swap(1,1,2,2,1)$ to this matrix produces the matrix D':

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

from which the matrix D can be obtained by $swap(1,2,2,1,1)$. Both swaps satisfy the first part of the additional precondition of a consolidating swap. Only the second part prevents back-and-forth swapping in this case.

Theorem 2.4 *If only consolidating swaps are allowed, every sequence of swaps terminates and the maximum number of swaps in such a sequence is bounded from the above by the square of the sum of all the elements in the matrix.*

Proof: First, we show that the sum of squares of matrix elements increases (by at least 1 because debts are integers) in a consolidating swap. It is enough to consider the four matrix elements touched by the swap. Denote $d_{p,q} = D_{i_p,j_q}$, $p, q = 1, 2$. Assume the precondition of the consolidating swap holds. Thus

$$|d_{21} - d_{12}| < d_{11} + d_{22}.$$

and $k = min(d_{12}, d_{21})$. W.l.o.g. assume $k = d_{12}$. Thus we also have

$$d_{21} - d_{12} < d_{11} + d_{22}.$$

Then

$$2k - d_{12} - d_{21} + d_{11} + d_{22} > 0$$

which is equivalent to

$$d_{11}^2 + d_{12}^2 + d_{21}^2 + d_{22}^2 < (d_{11}+k)^2 + (d_{12}-k)^2 + (d_{21}-k)^2 + (d_{22}+k)^2.$$

The sum of squares of all matrix elements is bounded from the above by the square of the sum of all matrix elements. The latter remains invariant in any swap. ∎

To achieve faster termination at the price of obtaining a "less consolidated" debt matrix, one can consider stronger preconditions for swap.

Definition 2.5 *(Locally Reducing Swap) An operation $swap(i_1, j_1, i_2, j_2, k)$ is called a* locally reducing *swap if it satisfies an additional precondition:*

> *the number of nonzero elements among $D_{i_1,j_1}, D_{i_2,j_2}, D_{i_2,j_1}$, and D_{i_1,j_2} is greater than among $D_{i_1,j_1}+k, D_{i_2,j_2}+k, D_{i_2,j_1}-k$, and $D_{i_1,j_2}-k$.*

Theorem 2.5 *If only locally reducing swaps are allowed, every sequence of swaps terminates and the maximum number of swaps in such a sequence is bounded from the above by mn.*

2.5 Comparison

We compare now CBR with consolidating and locally reducing swaps.

Invariants

Clearly, CBR, locally reducing and consolidating swaps of the above approaches guarantee **Truth in billing** and **Recovery of money**. However, only CBR guarantees **No surprises**. In the distributed approaches the latter property has to be imposed as an additional precondition of swaps.

Debt consolidation

How do various approaches fare as far as debt consolidation is concerned?

We show first that consolidating swaps are strictly more powerful than locally reducing ones.

Theorem 2.6 *If a locally reducing swap is applicable to a debt matrix, so is a consolidating one. There is a matrix to which a consolidating swap is applicable but a locally reducing one is not.*

Proof: We use the notation of Theorem 2.4. Assume a locally reducing $swap(1,1,2,2,k)$ for some $k > 0$. Because this swap is locally reducing, we have $d_{12} - k = 0$ or $d_{21} - k = 0$. W.l.o.g. assume $d_{21} \geq d_{12}$ and thus $d_{12} = k > 0$.

If $d_{21} - d_{12} < d_{11} + d_{22}$, then $swap(1,1,2,2,k)$ is consolidating. Otherwise, we have

$$d_{21} - d_{12} \geq d_{11} + d_{22}.$$

Assume $d_{22} \geq d_{11}$. Consider now $swap(1,2,2,1,d_{11})$. If $d_{11} > 0$, we have

$$d_{22} - d_{11} \leq d_{21} - d_{12} - 2d_{11} < d_{21} < d_{21} + d_{12}.$$

Thus $swap(1,2,2,1,d_{11})$ is consolidating.

We show now that assuming $d_{11} = 0$ leads to a contradiction. Indeed, in this case $swap(1,1,2,2,k)$ eliminates one zero element (d_{11}) and thus, being locally reducing, has to introduce two zero elements. Therefore, not only $d_{12} = k$, but also $d_{21} = k$ and $d_{12} = d_{21}$. But then not only $d_{11} = 0$, but also $d_{22} = 0$, which means that $swap(1,1,2,2,k)$ is not locally reducing.

To see that the second part of the theorem is true as well, consider the sequence of matrices each of which is obtained by a swap from the preceding one:

$$\begin{bmatrix} 2 & 0 & 2 \\ 1 & 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 3 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 3 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

No locally reducing swap is applicable to the first matrix. However, two consolidating swaps yield the third matrix with a smaller number of nonzero elements. ∎

Consolidating swaps are also more powerful than CBR (Cycle Breaking Reduction). However, to prove that we need a graph-theoretic lemma.

Lemma 2.1 *If G is an undirected graph with nonzero weights associated with edges by a function l, then in every cycle of length at least 4 in G there are three consecutive edges e_1, e_2, and e_3 such that $l(e_1) \leq l(e_2)$ and $l(e_3) \leq l(e_2)$.*

Proof: We fix the direction of the cycle. Assume the thesis is false, i.e., for every selection of three consecutive edges e_1, e_2, and e_3 in the cycle, $l(e_1) > l(e_2)$ or $l(e_3) > l(e_2)$.

If there is a pair of consecutive edges (e, e') in the cycle such that $l(e') > l(e)$, then for the edge e'' that follows e' in the cycle, $l(e'') > l(e')$. Repeating this step of reasoning through every pair of edges in the cycle, we get finally $l(e) > l(e_0)$ for the edge e_0 preceding e in the cycle. But $l(e_0) > l(e')$ and $l(e') > l(e)$ - a contradiction.

Otherwise, for all pairs (e, e') of consecutive edges in the cycle, $l(e') \leq l(e)$. This implies that for every pair (e, e') of not necessarily consecutive edges in the cycle, $l(e') = l(e)$ - a contradiction. ∎

Theorem 2.7 *If there is a cycle of length at least 4 in the debt graph G_D, then there is also a consolidating swap in D whose precondition is satisfied. There is a debt matrix to which a consolidating swap is applicable but whose corresponding debt graph does not have a cycle.*

Proof: We find the appropriate consolidating $swap(i_1, j_1, i_2, j_2, k)$ as follows. We fix the direction of the cycle. By Lemma 2.1 we can find in the graph G_D three consecutive edges e_1, e_2, and e_3 in the cycle such that $l(e_1) \leq l(e_2)$ and $l(e_3) \leq l(e_2)$. Thus

$$|l(e_1) - l(e_3)| \leq max(l(e_1), l(e_3)) - 1 < l(e_2).$$

Now if the edge $e_1 = (i, j)$ leads from a customer i to a vendor j, then $e_2 = (j, i')$ and $e_3 = (i', j')$ for some customer i' and vendor j'. In this case, we pick $i_1 = i'$, $j_1 = j$, $i_2 = i$, $j_2 = j'$, and $k = min(D_{i_2,j_1}, D_{i_1,j_2})$. Clearly:

$$|D_{i_2,j_1} - D_{i_1,j_2}| < D_{i_1,j_1} \leq D_{i_1,j_1} + D_{i_2,j_2}.$$

On the other hand, if the edge $e_1 = (j, i)$ leads from a vendor j to a customer i, then $e_2 = (i, j')$ and $e_3 = (j', i')$ for some customer i' and vendor j'. In that case, we pick $i_1 = i$, $j_1 = j'$, $i_2 = i'$, $j_2 = j$, and $k = min(D_{i_2,j_1}, D_{i_1,j_2})$. Again:

$$|D_{i_2,j_1} - D_{i_1,j_2}| < D_{i_1,j_1} \leq D_{i_1,j_1} + D_{i_2,j_2}.$$

To see that the second part of the theorem is true as well, consider the example in the proof of Theorem 2.6. The graph corresponding to the first matrix does not have a cycle, although there is an applicable consolidating swap. ∎

Corollary 2.1 *For any $m \times n$ debt matrix D, by iterating consolidating swaps, one can obtain a debt matrix with at most $m + n - 1$ nonzero elements.*

Proof: This follows from Theorems 2.2 and 2.7. ∎

However, we were not able to establish for the iteration of consolidating swaps similar time bounds as for CBR (Theorem 2.3), so if the whole debt matrix is available, CBR is the method of choice.

To complete the picture, we compare CBR and locally reducing swaps. A locally reducing swap may be applicable to a matrix, while Cycle Breaking Reduction is not and vice versa.

Example 2.5 To see an example of the first, consider the matrix

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

and for the second - the matrix

$$\begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 2 \\ 0 & 3 & 3 \end{bmatrix}$$

Finally, we should remark that a minimal matrix may fail to be found by any of the above approaches.

Example 2.6 Consider the following sequence of matrices each of which is obtained by a swap from the preceding one:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The swap leading from the first (nonminimal) matrix to the second one is neither consolidating not locally reducing. However, the swap is necessary to make the next swap, which produces a minimal matrix, possible. Notice Cycle Breaking Reduction will also stop at the first matrix because this matrix already represents a minimal spanning forest.

It is an open question whether there is a way to impose a precondition on swaps that guarantees that a minimal matrix is reached in a terminating sequence of swaps. Similarly, it is open whether one can improve the complexity bounds of Theorem 2.3.

Transaction processing

We compare now the swapping server and serverless approaches from the point of view of transaction processing. In the first approach, the transactions follow the *client-server* model. Vendors need not be aware of other vendors and be online for the swaps to be effected. The transactions are, however, of relatively *long duration*, as a significant amount of time can elapse between the instant when a vendor submits a column to a swapping server and the instant when he receives it back after consolidation. The transactions are also *large-scale* (involving the whole column of debts), so the reliability requirements are considerably more stringent than in the serverless approach. In the second approach, there is a need to support transactions between an arbitrary pair of vendors but the transactions are much smaller. On the other hand, the vendors need to know about each other and be on-line at the same time. To achieve faster progress in this case, swaps may be batched.

2.6 Accounts

Although one of the points of using debt swapping is to eliminate the need for customer accounts, one can incorporate account functionality in our framework, if desired. To do so, every customer i is assigned a single vendor $j = home(i)$, called the *home vendor* of i. The home vendor (and no other vendor) acquires the debt of the customer with other vendors through swapping. This effect is obtained by adding the following precondition to any swap operation $swap(i_1, j_1, i_2, j_2, k)$:

$$home(i_1) = j_1 \wedge home(i_2) = j_2.$$

However, there may be consolidating swaps that do not satisfy this restriction. Thus in this case we cannot guarantee an upper bound on the number of nonzero elements in a reduced matrix, as in Theorem 2.1. On the other, it is easy to see that each debt can now be swapped at most once and thus iterating restricted swaps can be done in $O(e)$ time where e is the number of nonzero elements in the debt matrix.

There is, however, another problem with the last approach. The home vendor of a customer may simple not have enough debt of other vendors' customers to swap with them. Thus swapping has in this case to be supplemented by a separate mechanism for *settlement* between vendors.

3 Debt consolidation using digitally signed notes

We assume that customers and vendors can be identified in a reliable and stable way. Reliability means that transactions are concluded between real parties, not impostors. Stability means that the parties are guaranteed to be around long enough for billing and possible arbitration. Also, we assume that party identification provides reliable information about the party's public key and the expiration date of the credentials. Identification is separate from payment. All these requirements are satisfied by, e.g., X.509 certificates [6] and apply also to most electronic payment protocols. In the description below we assume for simplicity that both customers and vendors are identified by their *public keys*.

3.1 Basic notions

Customer debts are represented as sets of *digitally signed promissory notes*. The notes are issued by individual customers and vendors. For simplicity we assume that all the notes have the same denomination, equal to the single monetary unit. The notes are either *original* (issued by a customer and exchanged in a purchase transaction) or *derived* (issued by a vendor and exchanged in a swap transaction). Every derived note is based on a single note from which it is derived (called its *source*) and a single original note (called its *root*). Each original note of a customer has a unique serial number.

In the following we use the notation $S_x(M)$ to denote the message M signed by x (who is a customer or a vendor). The value $S_x(M)$ can for example be obtained by computing

$E_x(h(M))$ where E_x is the encryption using the private key of x and h is a one-way hash function. It is verified by applying D_x (the decryption using the public key of x) to the encrypted message and checking whether the result is equal to $h(M)$.

Original notes. An original note P is of the form

$$P = M \mid\mid S_C(M)$$

where C is a customer – the issuer of the note. If the signature of C verifies correctly and

$$M = N \mid\mid C \mid\mid V \mid\mid D$$

then we say that P is *well formed* and the following *attributes* of P are well defined:

$$\begin{array}{lcl} \mathit{Issuer}(P) & = & C \\ \mathit{Bearer}(P) & = & V \\ \mathit{Number}(P) & = & N \\ \mathit{ExpDate}(P) & = & D. \end{array}$$

Derived notes. A derived note P is of the form

$$P = M \mid\mid S_{V_1}(M)$$

where V_1 is a vendor - the issuer of the note. If the signature of V_1 verifies correctly and

$$M = N \mid\mid C \mid\mid V_1 \mid\mid V_2 \mid\mid D \mid\mid P'.$$

then we say that P is *well formed* and the following *attributes* of P are well defined:

$$\begin{array}{lcl} \mathit{Issuer}(P) & = & V_1 \\ \mathit{Bearer}(P) & = & V_2 \\ \mathit{Number}(P) & = & N \\ \mathit{Customer}(P) & = & C \\ \mathit{ExpDate}(P) & = & D \\ \mathit{Source}(P) & = & P'. \end{array}$$

Note validity and redemption. Every well-formed original note is *valid.* It can be redeemed from the issuer of the note by its bearer (provided it is not a duplicate). On the other hand, there may be well-formed derived notes that are not valid and cannot be redeemed. For example, the source note may be not well formed or inconsistent with the derived note. We say that a derived note P is *valid* if

1. P is well formed, and

2. $P' = Source(P)$ is well-formed, and

3. the following *consistency conditions* hold:

   - $Customer(P) = Customer(P')$,
   - $Number(P) = Number(P')$,
   - $Issuer(P) = Bearer(P')$,
   - $ExpDate(P) \leq ExpDate(P')$, and

4. P' is valid[2].

A derived note cannot be redeemed from the issuer who is a vendor, not a customer. However, a derived note is ultimately based on an original note, called its *root* note. The root note of an original note is the note itself. The root note of a valid derived note P is recursively defined as the root note of $Source(P)$. The redemption of a derived note consists thus of the redemption of its root note.

Billing and arbitration. A customer bill is a set of notes issued by the customer (or derived from such) and presented to him for redemption by a vendor. Customers keep detailed records of all the notes issued. There is also an independent arbitrator available to both customers and vendors that becomes active when there is a dispute over a payment.

Anchored notes. The last consistency condition for derived notes guarantees that a valid note has not expired. However, it may be the case that the credentials of the issuer have expired before the note is redeemed. Also, bill processing and arbitration requires some time and it is necessary that the parties that participated in the derivation of a note be still around at that time. To achieve that it is enough to test on receiving of a note whether its expiration date is less than the expiration date of the credentials of the issuer by at least the amount of time necesary for bill processing and arbitration. Such notes will be called *anchored.*

3.2 Operations

We describe now how different operations of the protocol are handled in this approach.

For simplicity, we assume that purchases and swaps involve amounts equal to a single unit. We first describe the serverless version of the protocol.

[2] Condition 4 implies Condition 2but writing the conditions in this order leads to an obvious terminating tail-recursive definition.

Purchase. A customer C issues to a vendor V an original promissory note P. The vendor checks whether he is the bearer of the note P, and whether P is well formed and anchored. He also checks whether the expiration date of the note has not passed and whether he has never received an original note with the same number attribute from the same customer C. If the checks succeed, the note is accepted. Both the customer and the vendor store the note.

Swap. A vendor V_1 issues to a vendor V_2 a derived promissory note P. The vendor checks whether he is the bearer of the note, and whether P is well formed[3] and anchored. He also checks whether the expiration date of the note has not passed and whether he has never received an original note with the same number and customer attributes from the same vendor V_1. If the checks succeed, the note is accepted. V_2 stores the note, and V_1 marks the note P' as *"spent"* (V_1 still needs to keep a record of the note, to prevent multiple submission of the same note.) Symmetrically, V_2 issues a note to V_1 using the same approach. If anything goes wrong at any point, the whole transaction is aborted.

Billing. A vendor V sends to a customer C a set of unexpired notes issued by C or derived from notes issued by C. These notes are timestamped and signed by V. C should acknowledge the receipt immediately. If he doesn't the arbitrator is notified because V wants to be on the record that he submitted the bill on time. The customer (or, rather a program working on his behalf) analyzes all the notes submitted for redemption. The customer checks whether the vendor is the bearer of each submitted note. Moreover, each note is unfolded level by level and its validity completely checked. Any invalid notes are referred to the arbitrator. Valid notes (presumably a vast majority) are honored and a payment (in real money) is made to V. A proof of payment and a record of the redeemed notes are kept. (This record documents the purchases made by the customer.) If no payment is made, the vendor V contacts the arbitrator.

Arbitration. For each invalid or duplicate note submitted for redemption by a vendor V the arbitrator identifies the party guilty of duplication or forging and requests that this party remits the money to V (unless V himself is that party).

To prevent specific attacks the above protocol needs to be slightly extended: see the discussion in the remainder of this section.

Swapping server. In this approach the course of action depends on whether the vendor can trust the server or not. If so, no extension of the above protocol is necessary. Otherwise, exchanging the notes with the server requires an extra level of note derivation.

[3]Note that V_2 does not check the validity of the note but only its well-formedness. The validity check is expensive, as it requires the unfolding of all levels of the note, and is done only when the note is redeemed.

3.3 Possible attacks (and defenses)

Vendor fabricates or modifies a note. Assuming a vendor has only access to his own private key, he can only sign notes issued by himself. Thus if a fabricated (or modified) note is used in a derivation, the resulting note is not well formed. The cheating vendor will be identified during the validity check as the issuer of a note that is not well-formed.

Vendor derives a note which is inconsistent (e.g., numbers don't match) with the note from which it is derived. Similar to the above, except that the validity check discovers the violation of the consistency conditions.

Vendor derives a note from an object which is not a valid note. Like above.

Vendor pushes the expiration date of a derived note beyond that of the original note. If the bearer of the derived note always checks whether the note is anchored, such an attempt will be discovered when the note is redeemed (and the issuer will still be around).

Vendor duplicates a note. The customer will detect multiple notes with the same serial number. Moreover, each note carries its own history, so the duplicating vendor can be identified by the arbitrator. The arbitrator will unfold the notes in parallel until the same source note is found. The vendor who derived multiple notes from this note is guilty of duplication.

Vendor neglects to check the note. If the vendor tries then to derive a note from an invalid or duplicate note, he may be later accused of cheating because this situation is indistinguishable from one in which the vendor forges or duplicates the note himself. Notice that if a vendor correctly derives another note from a note which was itself derived from a duplicate (or forged) note, he cannot be accused of cheating.

Vendor erases part of the note. Any party can extract from a derived note P one of the notes P_1 from which P was, directly or indirectly, derived. However, this may be useful to a vendor V_1 only if he is the bearer of P_1. This is the situation when there is a cycle in the note P (through derivation the note "comes back" to V_1). Now V_1 can redeem (or circulate again) the note P_1, instead of P. There is no advantage to V_1 for doing that. In fact, V_1 exposes himself to additional risks. Suppose that another vendor V_2 that was present in the cycle duplicates the note. The information witnessing about the duplication is contained in P but not in P_1. Thus V_2's cheating eludes detection. In fact, the resulting situation is indistinguishable from the one in which V_1 performed the duplication.

Customer generates two notes with the same serial number. If they are sent to the same vendor, the vendor should detect that. If they are sent to different vendors, they can't have been produced by a cheating vendor and the arbitrator can demand that both be redeemed.

Customer refuses to honor a valid note. The bill is timestamped, so the customer cannot claim that it was submitted too late. Note validity can be checked by the arbitrator. The customer can also evade payment by claiming that the note has already been redeemed by someone else. To do this he needs to procure another valid note with the same serial number and a proof that it has already been redeemed and paid. The arbitrator can then compare the two notes and determine who performed the duplication.

Eavesdropping. A note can only be redeemed by the bearer who is identified in it. A vendor can derive another note from an illicitly obtained note but this can be detected when the derived note is analyzed.

Framing. A vendor cannot be framed by a customer or another vendor, unless the latter has access to the private key of the former.

Refusal to pay when requested by the arbitrator. This can be prevented by requiring that both customers and vendors post an initial security deposit.

Customer/vendor conspiracy. A customer C can conspire with a vendor V to let V redeem duplicate notes of C and refund C their cost. The customer C could then reject redemption requests from other vendors claiming the submitted notes are duplicates. However, to produce valid notes C or V needs to do the duplication and this can be detected by the arbitrator.

In general the individual notes have such a low value that duplicating or forging them in small quantities is simply not worthwhile. The parties may agree to absorb small losses (e.g., using a fund created from a small percentage of all security deposits) and pursue only massive cheaters.

3.4 Properties

Under the assumptions that no transactions are aborted and all the disputes are correctly resolved, our approach satisfies the following properties:

> **Preservation of money:** Each valid note can be redeemed and is redeemed at most once.
>
> **Cheater identification:** Cheating parties can be unambiguously identified and each dispute resolved. Also, for every invalid note at least one party that invalidated it is still around at the time of arbitration [4].

Note that in exceptional cases note redemption may be a complex process, involving the arbitrator. Aborting a multistep transaction is a risk because one party may have already acquired a valid note from the issuer. If this happens during a purchase, it is like

[4]It may be impossible that all the guilty parties be around: A conspiracy may involve accepting a note from someone whose credentials have expired.

nondelivery of purchased goods and is beyond the scope of our protocol. In batched swaps, the risk of this event occurring may be reduced by alternating the roles of the issuer and the bearer of issued notes.

Our protocol does not directly support the notion of an account open with a financial institution, so imposing spending limits is not easy. A customer can issue as many notes as he desires, implicitly allowing them to be redeemed with him. If he refuses to do so and his security deposit is used up, the only remaining step is to revoke his credentials.

3.5 Cost

We ignore the cost of reliable identification and arbitration.

Computational requirements. Purchase transaction requires a single signature check and database lookup for the vendor, and a single signature generation for the customer. Swap transactions require a database lookup, two database updates (not necessarily online), a signature generation and a signature verification (per vendor). When a derived note is redeemed it is checked in its entirety, i.e., including all the notes from which it was derived and which should be included in it. This requires as many signature verifications as there are derivation levels in the note.

Bill size. Each derived note includes the note from which it is derived, so swapping increases the note size by an additive factor. Suppose the size of an original note is r bytes. Then after k swaps it will grow to $(k+1)r$ bytes. This is an important concern, as bills consist of large numbers of notes. For example, assume every note is worth 1c and the size of an original note is 100 bytes. To get a billable amount, say $10, one needs then 1000 notes. To keep the bill size under 1 megabyte, one has to limit the number of swaps applied to a note to 9. One can reduce the note size by using shorter keys and consequently weaker cryptography, or compression. It is interesting to see how imposing restrictions on the number of swaps could affect the rate of progress of debt consolidation.

Storage cost. Assume the average note has 500 bytes. Storing it on disk costs .005c, assuming the current prices of $100 for 1 gigabyte.

4 Related work

Our approach is unique in that it does not require any intermediaries, e.g., banks or brokers, between customers and vendors. It is enough that customers and vendors use a reliable identification mechanism and a common arbitrator. The approach supports fully distributed payment aggregation.

The existing approaches to electronic payments can be classified along several different dimensions. (The book [7] provides a good description of some of those systems.) First, they can be *token-based* (cash-like) or *notational* (account-based) [2]. Notational approaches can further be subdivided into *debit-based* (check-like) and *credit-based* (credit-card-like) ones. Second, they can be *centralized* (using a single common bank, broker, or payment server) or *distributed* (using multiple, different banks or brokers). Third, the interaction of vendors with brokers may be *online* or *offline*. Finally, various systems differ in the degree of support for payment aggregation, essential for micropayments. In this brief survey, we do not address many other important issues like anynymity, non-repudiation etc.

Digicash [3], one of the earliest electronic payment systems, represents a token-based, centralized, online approach. To pay in this system a customer has to obtain a digital token from a bank which is subsequently passed to a vendor. The vendor verifies the token with the issuing bank. The bank has to check for *double spending*, as it is trivial to copy a digital token. Digicash was designed to support customer anonymity, not payment efficiency. The online interaction with a central bank is a bottleneck here and its cost makes Digicash unsuitable for micropayments. Also, the only payment aggregation is through the use of a single customer bank account for many purchases.

NetBill [5] is a notational, centralized, online debit-based system. The parties write authorizations to exchange money, in which banks and a special NetBill server are involved. NetBill was designed to support all the stages of a payment, including the initial price offer and the delivery of goods. The online interaction with a bank (and a NetBill server) is again a bottleneck here and makes NetBill transactions relatively costly. Payment aggregation is supported through the use of a single bank account for many purchases.

The Secure Electronic Transaction (SET) protocol of Visa and MasterCard [7] is a notational, centralized, online credit-based system. It can be viewed as a protocol providing secure account-transfer system over the Internet. Thus the cost of a SET transaction will be comparable to that of a traditional credit-card transaction, and is thus too high for micropayments. Payment aggregation is account-based.

Millicent [9] is a notational, centralized, debit-based system. Vendors issue "scrip" (encrypted account information) to individual customers, which can then be used for purchases with the issuing vendors. Vendor scrip can also be obtained from a broker with whom customers maintain accounts. The only form of aggregation is through the use of those accounts. In fact, Millicent may be viewed as based on a principle opposite to aggregation: Customers purchase first some amount of scrip which is then broken up into individual payments. The presence of a broker, common to customers and vendors, makes Millicent centralized. The broker has to be online to provide vendor scrip but does not have to stay online for the individual purchases to be effected. SubScrip [7] is a version of Millicent in which vendors exchange electronic cash for scrip and thus the need for brokers is eliminated.

PayWord [13] is a notational, distributed, offline, credit-based system. Brokers issue signed certificates that can then be used by customers to generate payment tokens with arbitrary vendors. (Many tokens with the same vendor can be generated on the basis of a single signature check through the use of one-way hash functions. A similar technique can be applied to other approaches to micropayments, including ours.) Vendors periodically redeem aggregated payment tokens from brokers, so the aggregation is account-based. A customer and a vendor have to share a broker for a transaction to be possible. This protocol forms the basis of the Micro Payment Transfer Protocol [10]. A similar debit-based system, NetCard, was independently proposed in [1].

MicroMint [13] is a token-based, distributed, offline system. Brokers mint and sell digital coins that are represented by hash function collisions. Vendors periodically redeem obtained coins from brokers. Aggregation is again account-based.

Agora [8] is a notational, distributed, offline, credit-based system. Customers and vendors exchange digitally signed purchase offers and requests. Such a request forms a proof of purchase and can be redeemed from the vendor's bank which obtains the funds from the customer's bank through a settlement process. The aggregation is account-based but it is the vendors' accounts that are used which makes it more effective (the overall payments received by a single vendor are typically larger than those incurred by a single customer). A similar protocol was independently proposed in [11]. We have borrowed from [8] the idea of using digital signatures and an arbitrator.

5 Conclusions and further work

We have presented a novel approach to billing for micropayments. The approach is based on the idea of *distributed debt consolidation.* We have shown that our protocol is applicable to both trusted and untrusted environments.

Our current work involves a complete implementation of the proposed approach and its deployment in a test environment. We also envision further extensions of the protocol to handle multiple currencies and denominations. We believe that our approach opens up a number of interesting research topics, including algorithms and heuristics for debt swapping, various scenarios for debt consolidation, interoperability of our protocol with other electronic payment protocols, and formal security analysis of the protocol. Also, the societal and legal issues raised by our approach need to be carefully examined. Will customers accept the decoupling of purchases and billing? Will practically unlimited credit be acceptable, even if restricted to micropayments? How could arbitrators be selected? What constitutes the cost of a microtransaction?

6 Acknowledgments

We are grateful to Paul Seymour for suggesting the proof of Theorem 2.2. Thanks go also to Gene Tsudik for suggesting some of the attacks on the cryptographic version of our protocol and to Marc Pucci and Ron Underwood for working with us on an early version of this paper.

References

[1] R. Anderson, C. Manifavas, and C. Sutherland. NetCard – A Practical Electronic Cash System. In *Cambridge Workshop on Security Protocols*, 1996.

[2] L. Jean Camp, M. Sirbu, and J.D. Tygar. Token and Notational Money in Electronic Commerce. In *First USENIX Workshop on Electronic Commerce*, 1995.

[3] D. Chaum. Achieving Electronic Privacy. *Scientific American*, pages 96–101, 1992.

[4] J. Chomicki, S. Naqvi, M. F. Pucci, and R. W. Underwood. Decentralized micropayment consolidation. In *International Conference on Distributed Computing Systems*, Amsterdam, The Netherlands, May 1998.

[5] B. Cox, J.D. Tygar, and M. Sirbu. NetBill Security and Transaction Protocol. In *First USENIX Workshop on Electronic Commerce*, 1995.

[6] W. Ford and M.S. Baum. *Secure Electronic Commerce.* Prentice Hall, 1997.

[7] A. Furche and G. Wrightson. *Computer Money.* dpunkt, 1996.

[8] E. Gabber and A. Silberschatz. Agora: A Minimal Distributed Protocol for Electronic Commerce. In *Second USENIX Workshop on Electronic Commerce*, 1996.

[9] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The Millicent Protocol for Inexpensive Electronic Commerce. *World Wide Web Journal*, 1(1):603–618, 1995.

[10] P.M. Hallam-Baker. Micro Payment Transfer Protocol (MPTP) Version 0.1, November 1995. http://www.w3.org/TR/WD-mptp-951122.

[11] A. Herzberg and H. Yochai. Mini-Pay: Charging per Click on the Web. In *Sixth International World Wide Web Conference*, 1997.

[12] R. Kalakota and A. Whinston. *Frontiers of Electronic Commerce.* Addison-Wesley, 1996.

[13] R. L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. In *RSA'96 Conference.*