Decentralized Privacy-Preserving Netting Protocol on Blockchain for Payment Systems

Shengjiao Cao[1], Yuan Yuan[1], Angelo De Caro[2], Karthik Nandakumar[3], Kaoutar Elkhiyaoui[2], and Yanyan Hu[3]

[1] Ant Financial Services Group
{`s.cao, ida.yuan`}`@antfin.com`
[2] IBM Research Zurich
{`adc,kao`}`@zurich.ibm.com`
[3] IBM Research Singapore {`nkarthik, yanyanhu`}`@sg.ibm.com`

Abstract. This paper proposes a decentralized netting protocol that guarantees the privacy of the participants. Namely, it leverages the blockchain and its security properties to relax the trust assumptions and get rid of trusted central parties. We prove the protocol to be optimal and we analyze its performance using a proof-of-concept implemented on top of Hyperledger Fabric.

Keywords: Decentralized Netting · Payment Systems · Blockchain · Zero-knowledge Proofs.

1 Introduction

Currently, banks settle their liabilities to each other through *inter-bank payment systems* generally managed by their country's central bank. The central bank opens an account for the local banks in its jurisdiction and enforces that each account maintains a certain level of liquidity to accommodate future settlements. Upon receiving a payment instruction, say bank A transfers x home-currency to bank B, the central bank deducts x from A's account while adding x to B's account.

Historically, inter-bank payments were settled via (end of day) netting systems, but as the volume and the value of transactions increased central banks became wary of the risks involved in deferred net settlement systems. Now, central banks favor real-time gross settlement (RTGS) systems. In RTGS, payment instructions are settled individually and immediately at their full amount. However, the benefit of immediate finality incurs high liquidity costs on the banks. The liquidity demands in RTGS systems are enormous; in fact, the daily transfer volume in typical inter-bank payment systems could be as large as a substantial fraction of the annual GDP [1]. Fig. 1 on the left illustrates a simple scenario in which participating banks are not able to settle their payments individually due to insufficient liquidity, bringing the system to a halt known as *gridlock*. To resolve gridlocks, banks combine RTGS with liquidity saving mechanisms (LSM),

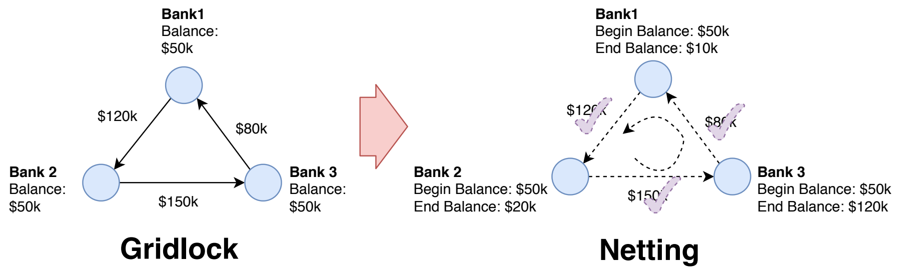

Fig. 1. Illustration of a gridlock scenario and netting procedure

of which *netting* is the most effective one. Fig. 1 on the right illustrates how triple-lateral netting helps resolve system gridlock.

Traditionally, central banks are in charge of resolving gridlocks and guaranteeing that the system runs smoothly without interruptions. Essentially, they operate a centralized payment queue to which every participating bank submits its payment instructions, and perform *multilateral netting*. That is, the process of offsetting the value of multiple payments due to be exchanged between the banks. After netting, the central banks settle the net liability of the participants without overdraft. In addition to performing netting correctly, central banks are trusted to preserve the confidentiality of payment instructions coming from each bank. While placing such trust in central banks may be justified, it comes with great liability risk for them. Furthermore, in the case of cross-border multi-currency transfer, it is challenging to find a *trusted mediator* to settle payments. This is why central banks are actively looking for alternatives to centralized netting and settlement.

Thanks to the emergence of Bitcoin [2] and the ensuing interest in blockchain technology, financial institutions have been investigating avenues to make *decentralized inter-bank payment systems* a reality, e.g., Project Jasper [3], Project Ubin [4]. However, while these two projects succeed in removing the single point of failure and achieving immediate and final settlement without the need for transaction reconciliation, their prototype systems are missing the crucial functionality of decentralized multilateral netting, making them less practical.

We recall here that a multilateral netting process is viable, if it is correct and fair. Correctness ensures that **(i)** when the sender's account is debited x dollars, the receiver's account is credited x dollars; and **(ii)** participants will not pay more than their current balance plus their allowed credit. These two properties guarantee that the total liquidity in the system remains the same before and after settlement. Fairness on the other hand, captures the requirement that netting should not favor any participant in terms of payment settlement priority, rather it should reach an overall optimal netting strategy, i.e. either the maximum number of instructions settled, or the maximum amount of payments settled.

In the case of centralized netting [5–7], correctness and fairness are easy to satisfy as the central party sees all payment instructions and can perform the netting and update the banks' accounts accordingly. In contrast, meeting these requirements in a decentralized payment system is a real challenge. Without a trusted central party, participants might be reluctant to advertise their payment instructions for everyone to see. This means that in a decentralized netting protocol, participants should see only their own payment instructions and based on those decide which payments to settle first. An ill-designed protocol however could allow a malicious participant to choose to settle only the payments that increase her current liquidity balance to the detriment of others'. Therefore, the best approach to design decentralized netting solutions is to enable each participant to solve their own net settlement locally, and have in place a mechanism to verify that the local settlements are both fair and correct.

In this paper, we propose a solution that leverages the blockchain to implement decentralized netting without sacrificing the privacy of the participants. It should be noted that we are not the first to propose leveraging the blockchain for netting. Recently, Wang et al. [8] introduced a blockchain-based netting solution for gridlock, but it differs from ours in two aspects: **(i)** it relies on a central party to check the netting result and make sure that the total liquidity is preserved; and **(ii)** while it hides the individual payment amount of the involved banks, it reveals the net amounts that should be paid.

Contributions. The contributions of this paper are two-fold:

– A first-of-its-kind decentralized netting protocol that does not require any central party but still guarantees correctness and fairness. The proposed protocol collects the local settlements of the participants and feeds them to a smart contract running on the blockchain to reach a *globally optimal*, *correct* and *fair* settlement.
– An enhanced privacy-preserving extension that further protects the confidentiality of the payment amounts. In this extension, payment amounts are encoded as *homomorphic Pedersen commitments* and *zero-knowledge proofs* are provided to the smart contract to verify the correctness of the local settlement in a privacy-preserving manner.

The remainder of the paper is organized as follows. In Section 2, we formulate our problem followed by the proposed decentralized netting protocol without privacy. In Section 3, we give a detailed description of how to enhance our protocol with privacy. In Section 4, we construct a blockchain-based payment system and analyze its security properties. In Section 5, we present an implementation of the protocol on Hyperledger Fabric and discuss our evaluation results. We conclude our paper in Section 6 with possible future work.

2 Decentralized Netting Protocol

The general netting problem is NP complete and can be solved only approximately using the algorithms in [6, 7]. These algorithms often yield multiple possible solutions instead of the optimal one, hence *sacrificing fairness*. In practice, central banks sort payments according to an order determined either by settlement deadlines or by priorities defined by the participants themselves. Payments with high priority are settled before payments with low priority. These allow central banks to find the optimal solution and achieve fairness.

In the remainder of the paper, we restrict ourselves to the problem formulation in [5] that focuses on netting for payments with priority constraints.

Notation. For $n \in N$, let $[n] = \{1, \ldots, n\}$. For $\mathbf{x} = [x_1, ..., x_n], \mathbf{y} = [y_1, ..., y_n]$, let $\mathbf{x} \geq \mathbf{y}$ denote $x_k \geq y_k, \forall k \in [n]$. Let $I(\cdot)$ denote the indicator function, i.e., $I(b) = 1$ if b is true, otherwise $I(b) = 0$.

2.1 The Netting Problem

Let n be the number of participants in the payment ecosystem and P_i refer to the ith participant. Let d_i denote the credit limit of (or the amount of cover money deposited by) P_i. Let $\mathbf{PayOutQ}_i$ denote the queue containing the outgoing payment instructions of P_i (i.e., outstanding payments where P_i is the sender):

$$\mathbf{PayOutQ}_i = [\text{PayOut}_{i,1}, ..., \text{PayOut}_{i,m_i}] \tag{1}$$

$$\text{PayOut}_{i,k} = (\text{Rec}_{i,k}, \text{Amt}_{i,k}) \tag{2}$$

$$\text{Rec}_{i,k} \in \{P_j\}_{j=1, j \neq i}^{n} \tag{3}$$

$$\text{Amt}_{i,k} > 0 \tag{4}$$

where m_i is the number of payment instructions in $\mathbf{PayOutQ}_i$, $\text{Rec}_{i,k}$ and $\text{Amt}_{i,k}$ are the receiver and amount in payment instruction $\text{PayOut}_{i,k}$, respectively. Let $x_{i,k} \in \{0, 1\}$ be the indicator of whether $\text{PayOut}_{i,k}$ will be settled after netting:

$$x_{i,k} = \begin{cases} 1 \text{ if} & \text{PayOut}_{i,k} \text{ will be settled} \\ 0 \text{ o/w} \end{cases} \tag{5}$$

Let $\mathbf{x}_i \overset{\text{def}}{=} [x_{i,1}, ...x_{i,m_i}]$ and $\mathbf{x} \overset{\text{def}}{=} [\mathbf{x}_1, ...\mathbf{x}_n]$. Given $\mathbf{x}$, we define:

$$T_i(\mathbf{x}) = \sum_{k=1}^{m_i} x_{i,k} \tag{6}$$

$$S_i(\mathbf{x}) = \sum_{k=1}^{m_i} x_{i,k} \text{Amt}_{i,k} \tag{7}$$

$$R_i(\mathbf{x}) = \sum_{j=1}^{n} \sum_{k=1}^{m_j} x_{j,k} \text{Amt}_{j,k} I(\text{Rec}_{j,k} = P_i) \tag{8}$$

$T_i(\mathbf{x})$ denotes the number of outgoing payment instructions from P_i that will be settled after netting, $S_i(\mathbf{x})$ denotes the total outgoing amount from P_i and $R_i(\mathbf{x})$ denotes the total incoming amount to P_i. Let $\hat{B}_i$ and $\tilde{B}_i$ be the ex-ante (before netting) and ex-post (after netting) balances of P_i, respectively. The relationship between $\hat{B}_i$ and $\tilde{B}_i$ is given by:

$$\tilde{B}_i = \hat{B}_i - S_i(\mathbf{x}) + R_i(\mathbf{x}) \tag{9}$$

The **netting problem** corresponds to finding the optimal solution that satisfies the following equations:

$$\max_{\mathbf{x}} \sum_{i=1}^{n} f_i(\mathbf{x}) \tag{10}$$

$$\text{s.t. } \tilde{B}_i \geq -d_i, \forall i \in [n] \tag{11}$$

where the **liquidity constraint** (11) stipulates that if the payments are simultaneously settled according to $\mathbf{x}$ then the ex-post balances plus credit limit of each bank has to be non-negative. Here $f_i(\mathbf{x})$ can be either $T_i(\mathbf{x})$ for the number of payments, or $S_i(\mathbf{x})$ for the total monetary value settled. We impose the constraint that the payment instructions in the outgoing queue can only be settled in the given priority order:

$$x_{i,k+1} \leq x_{i,k}, \forall i \in [n], \forall k \in [m_i - 1] \tag{12}$$

For example, for any $j > k$, $\text{PayOut}_{i,j}$ can not be settled if $\text{PayOut}_{i,k}$ is not settled, implying that $x_{i,j}$ must be 0 if $x_{i,k} = 0$. Under these constraints, either choice of $f_i(\mathbf{x})$ leads to the same optimal solution, as proved in [5].

Let $h(\mathbf{x}_i)$ denote the index of the lowest priority instruction in $\mathbf{PayOutQ}_i$ that can be settled:

$$h(\mathbf{x}_i) = \begin{cases} 0 & \text{if} \quad x_{i,k} = 0, \forall k \in [m_i] \\ \max_k kI(x_{i,k} = 1) & \text{o/w} \end{cases} \tag{13}$$

2.2 Blockchain-based Decentralized Netting

In the following, we describe our solution for blockchain-based decentralized netting without privacy.

Each participant is endowed with a system-wide public key and an account associated with that public key. Each payment instruction submitted to the blockchain comes with a signature of its sender and a priority order. The payment smart contract verifies the signature and settles the payment instruction immediately if there is enough balance in the sender's account and no other higher priority payment instruction in the outgoing queue. Otherwise, the payment instruction is added to the sender's outgoing queue to be settled later.

Netting of payment instructions is triggered either periodically or the moment the total queue size reaches a certain threshold defined by the system administrator. The netting process iterates through multiple rounds until convergence. Each round consists of two operations:

Algorithm 1 Blockchain-based Decentralized Netting

```
 1: **Inputs**: $\hat{B}_i, d_i, \mathbf{PayOutQ}_i, \forall i \in [n]$        // $\hat{B}_i$ is ex-ante balance
 2: **Outputs**: $\mathbf{x}$        // Indicator of payments that can be settled
 3: **Initialization**: $t \leftarrow 1$, $x^0_{i,k} \leftarrow 1, \forall i \in [n], k \in [m_i]$, $l^0 = \sum_{i=1}^{n} m_i$
 4: **repeat**
 5:   **At each** $P_i$: calculate the local proposal $\mathbf{x}^t$ for round $t$:
 6:     Set $\mathbf{x}^t \leftarrow \mathbf{x}^{t-1}$, $z_i \leftarrow h(\mathbf{x}^t_i)$        // Include all payments from last round
 7:     **While** $z_i \geq 0$        // Iterate until liquidity constraint is satisfied
 8:       $\tilde{B}^*_i = \hat{B}_i - S_i(\mathbf{x}^t) + R_i(\mathbf{x}^t)$        // Calculate ex-post balance
 9:       **If** $\tilde{B}^*_i \geq -d_i$        // Liquidity constraint is satisfied
10:         **break**
11:       **Else**
12:         $x^t_{i,z_i} \leftarrow 0$, $z_i \leftarrow z_i - 1$        // Remove lowest priority payment
13:     **If** $z_i = m_i$
14:       Submit the proposal $\{\mathbf{x}^t_i, \tilde{B}^*_i, \emptyset\}$ to the ledger
15:     **Else**
16:       $\tilde{B}'_i \leftarrow \tilde{B}^*_i - \mathrm{Amt}_{i,z_i+1}$        // Calculate hypothetical balance
17:       Submit the proposal $\{\mathbf{x}^t_i, \tilde{B}^*_i, \tilde{B}'_i\}$ to the ledger
18:   **Smart Contract:** Upon receiving proposals from $N$ participants for round $t$
19:     **for** i=1,...,n
20:       **Verify Priority Constraint**: $x^t_{i,k+1} \leq x^t_{i,k}, \forall k \in [m_i - 1]$
21:       **Verify Convergence Constraint**: $x^t_{i,k} \leq x^{t-1}_{i,k}, \forall k \in [m_i]$
22:       **Verify Liquidity Constraint**: $\tilde{B}^*_i \equiv \hat{B}_i - S_i(\mathbf{x}^t) + R_i(\mathbf{x}^t)$ and $\tilde{B}^*_i \geq -d_i$
23:       **If** $z_i < m_i$
24:         **Verify Optimality**: $\tilde{B}'_i \equiv \tilde{B}^*_i - \mathrm{Amt}_{i,z_i+1}$ where $z_i = h(\mathbf{x}^t_i)$ and $\tilde{B}'_i < -d_i$
25:     **end**
26:     Calculate $l^t = \sum_{i=1}^{n} \sum_{k=1}^{m_i} x^t_{i,k}$        // total number of payments to be settled
27:     **If** $l^t \equiv l^{t-1}$,
28:       Exit $\mathbf{x} \leftarrow \mathbf{x}^t$
29:     **Else**
30:       $t \leftarrow t + 1$, continue to next round.
31: **until** converged
32: return $\mathbf{x}$
```

1.) Participant Proposal: Each participant P_i calculates her nettable set, which corresponds to the maximum number of payments that can be settled from P_i's outgoing queue without violating the liquidity and the priority constraints, see Algorithm 1 lines 5 to 12. The calculation also takes into consideration the incoming payments from the aggregate nettable set of the previous round. Note that in the first round, all incoming payments are included. P_i then submits her nettable set and to-be-post-balance (i.e. new balance once the payments in the nettable set are finalized). Furthermore, P_i proves the *optimality* of her nettable set by including the highest priority payment that cannot be resolved in the current proposal and showing that the corresponding hypothetical to-be-post-balance is less than $-d_i$, cf. line 16.

2.) Smart Contract Verification and Net Payment Aggregation: Upon receiving proposals from all participants, the smart contract verifies whether each proposal **(i)** satisfies the **priority and liquidity constraints** and **(ii)** is optimal, check lines 20 to 24. If the verification succeeds, then the individual nettable sets of all participants are aggregated to obtain the *aggregate nettable set* for this round.

The smart contract next checks whether the new aggregate nettable set is the same as the one in the previous round. If so, then netting process has converged and the smart contract concludes its execution by returning the aggregate nettable set, cf. lines 27 and 28. If the aggregate nettable set is empty, we call it a DEADLOCK as no payments can be settled on a net basis. In the case of a DEADLOCK, the participants are unable to settle any of the queued payments unless new liquidity is injected into the system.

If all participants are honest, then this decentralized netting protocol achieves the global optimal solution for the netting problem.

Theorem 1. *Algorithm 1 always finds a unique and optimal solution for problem defined by equations* (10) *to* (12)*. In addition, the solution is independent of the choice of the objective function as either maximum total value settled or maximum number of payments settled.*

The proof of Theorem 1 is deferred to Appendix A.

3 Decentralized Privacy-Preserving Netting

Although algorithm 1 is optimal, it does not protect the privacy of system participants. All payment instructions are posted to the ledger in the clear. To preserve the privacy of the participants, we enhance our decentralized netting protocol with Pedersen commitments [11] and zero-knowledge range proofs [12].

3.1 Pedersen Commitments for Privacy-preserving Ledger

Instead of posting account balances and payment amounts to the ledger in the clear, we obfuscate them using Pedersen commitments. These commitments are *hiding* and *binding*: meaning that they do not reveal any information about the committed values and that they cannot be opened to different values later.

Let G be a cyclic group of large prime order p and let g and h be two random generators of G. A Pedersen commitment to a value $v \in F_p$ is computed as $\mathsf{com}(v, r) = g^v h^r$, where $r \in F_p$ is a randomly-chosen blinding factor. By construction, Pedersen commitments are *additively homomorphic*:

$$\mathsf{com}(v_1, r_1)\mathsf{com}(v_2, r_2) = \mathsf{com}(v_1 + v_2, r_1 + r_2)$$

On the ledger, the account balance $\hat{B}_i$ of P_i is stored as $\mathsf{com}(\hat{B}_i, \hat{r}_i)$ while the amount $\text{Amt}_{i,k}$ of the k-th payment message in P_i's outgoing queue is stored as $\mathsf{com}(\text{Amt}_{i,k}, r_{i,k})$, where $\hat{r}_i$ and $r_{i,k}, \forall i \in [n], k \in [m_i]$ are randomly-sampled

Algorithm 2 Decentralized Privacy-Preserving Netting

```
1: **Public Inputs**: $\mathsf{com}(\hat{B}_i, \hat{r}_i), d_i, \mathbf{PayOutQ}_i, \forall i \in [n]$
2: **Outputs**: $\mathbf{x}$                                   // Indicator of payments that can be settled
3: **Initialization**: $t \leftarrow 1$, $x^0_{i,k} \leftarrow 1, \forall i \in [n], k \in [m_i]$, $l^0 = \sum_{i=1}^{n} m_i$
4: **repeat**
5:    **At each** $P_i$: calculate the local proposal $\mathbf{x}^t$ for round $t$:
6:       $\mathbf{x}^t \leftarrow \mathbf{x}^{t-1}$, $z_i \leftarrow h(\mathbf{x}^t_i)$            // Include all payments from last round
7:       **While** $z_i \geq 0$                    // Iterate until liquidity constraint is satisfied
8:          $\tilde{B}^*_i = \hat{B}_i - S_i(\mathbf{x}^t) + R_i(\mathbf{x}^t)$                // Calculate ex-post balance
9:          **If** $\tilde{B}^*_i \geq -d_i$                               // Liquidity constraint is satisfied
10:            **break**
11:         **Else**
12:            $x^t_{i,z_i} \leftarrow 0$, $z_i \leftarrow z_i - 1$                      // Remove lowest priority payment
13:      Calculate $\mathsf{com}(\tilde{B}^*_i, \tilde{r}^*_i) \leftarrow \mathsf{com}(\hat{B}_i, \hat{r}_i)\mathsf{com}(S_i(\mathbf{x}^t), r'_i)^{-1}\mathsf{com}(R_i(\mathbf{x}^t), r''_i)$
14:      Construct zkrp$_{\mathsf{II}}$ proof $\pi(\tilde{B}^*_i)$                      // zero-knowledge range proof
15:      **If** $z_i = m_i$
16:         Submit proposal $\left\{\mathbf{x}^t_i, \mathsf{com}(\tilde{B}^*_i, \tilde{r}^*_i), \pi(\tilde{B}^*_i), \emptyset, \emptyset\right\}$ to ledger
17:      **Else**
18:         $\mathsf{com}(\tilde{B}'_i, \tilde{r}'_i) \leftarrow \mathsf{com}(\tilde{B}^*_i, \tilde{r}^*_i)\mathsf{com}(\mathrm{Amt}_{i,z_i+1}, r_{i,z_i+1})^{-1}$
19:         Construct zkrp$_{\mathsf{III}}$ proof $\pi(\tilde{B}'_i)$                     // zero-knowledge range proof
20:         Submit proposal $\left\{\mathbf{x}^t_i, \mathsf{com}(\tilde{B}^*_i, \tilde{r}^*_i), \pi(\tilde{B}^*_i), \mathsf{com}(\tilde{B}'_i, \tilde{r}'_i), \pi(\tilde{B}'_i)\right\}$ to ledger
21:   **Smart Contract:** Upon receiving proposals from $N$ participants for round $t$
22:      **for** i=1,...,n
23:         **Verify Priority Constraint**: $x^t_{i,k+1} \leq x^t_{i,k}, \forall k \in [m_i - 1]$
24:         **Verify Convergence Constraint**: $x^t_{i,k} \leq x^{t-1}_{i,k}, \forall k \in [m_i]$
25:         **Verify Liquidity Constraint**:
26:            $\mathsf{com}(\tilde{B}^*_i, \tilde{r}^*_i) \equiv \mathsf{com}(\hat{B}_i, \hat{r}_i)\mathsf{com}(S_i(\mathbf{x}^t), r'_i)^{-1}\mathsf{com}(R_i(\mathbf{x}^t), r''_i)$
27:            and $\mathsf{Verify}_{\mathsf{II}}(\pi(\tilde{B}^*_i), \mathsf{com}(\tilde{B}^*_i, \tilde{r}^*_i)) \equiv 1$
28:         **If** $z_i < m_i$
29:            **Verify Optimality**:
30:               $\mathsf{com}(\tilde{B}'_i, \tilde{r}'_i) \equiv \mathsf{com}(\tilde{B}^*_i, \tilde{r}^*_i)\mathsf{com}(\mathrm{Amt}_{i,z_i+1}, r_{i,z_i+1})^{-1}$ where $z_i = h(\mathbf{x}^t_i)$
31:               and $\mathsf{Verify}_{\mathsf{III}}(\pi(\tilde{B}'_i), \mathsf{com}(\tilde{B}'_i, \tilde{r}'_i)) \equiv 1$
32:      Calculate $l^t = \sum_{i=1}^{n}\sum_{k=1}^{m_i} x^t_{i,k}$        // Total number of payments to be settled
33:      **If** $l^t \equiv l^{t-1}$,
34:         Exit $\mathbf{x} \leftarrow \mathbf{x}^t$
35:      **Else**
36:         $t = t + 1$, continue to next round.
37: **until** converged
38: return $\mathbf{x}$
```

in F_p. We assume that the sender transmits the payment amount ($\mathrm{Amt}_{i,k}$) and randomness ($r_{i,k}$) to the receiver privately. Thanks to the homomorphic property of Pedersen commitments, we are able to translate the plain-text balance calculation in Eq. (7) to (9) into an obfuscated calculation as follows:

$$\mathsf{com}(S_i(\mathbf{x}), r_i') \overset{\text{def}}{=} \prod_{k=1}^{m_i} \mathsf{com}(\mathrm{Amt}_{i,k}, r_{i,k})^{x_{i,k}} \tag{14}$$

$$\mathsf{com}(R_i(\mathbf{x}), r_i'') \overset{\text{def}}{=} \prod_{j=1}^{n}\prod_{k=1}^{m_j} \mathsf{com}(\mathrm{Amt}_{j,k}, r_{j,k})^{x_{j,k} I(\mathrm{Rec}_{j,k}=P_i)} \tag{15}$$

$$\mathsf{com}(\tilde{B}_i, \tilde{r}_i) \overset{\text{def}}{=} \mathsf{com}(\hat{B}_i, \hat{r}_i)\mathsf{com}(S_i(\mathbf{x}), r_i')^{-1}\mathsf{com}(R_i(\mathbf{x}), r_i'') \tag{16}$$

Note that Pedersen commitments support positive integers only. However, account balances in our solution may become negative in the case of an overdraft. If we assume that the total liquidity in the system is less than some integer U and that $2U < p$, then Pedersen commitments can be used to handle all integers in $(-U, U)$ by simply mapping negative numbers $w \in (-U, 0)$ to $w + p > p/2$. With this mapping, we ensure that for all $v \in (0, U)$, $\mathsf{com}(v, r_1)\mathsf{com}(-v, r_2) = \mathsf{com}(v, r_1)\mathsf{com}(p - v, r_2) = \mathsf{com}(0, r_1 + r_2)$. For ease of notation, we just write negative integers as they are, i.e., $w \in (-U, 0)$ instead of $w + p$.

3.2 Zero Knowledge Range Proofs

Since both balances and payment amounts are hidden, smart contracts cannot rely only on the information in the ledger to verify the correctness and optimality of the participant proposals. We therefore require that each payment instruction comes with a zero knowledge proof that the payment amount is less than U and the participant proposals carry zero knowledge proofs that shows that the liquidity and optimality constraints are not violated. More precisely, participants are asked to produce three types of zero-knowledge range proofs (zkrp for short).

Definition 1. *A type I zero knowledge range proof* $\mathrm{zkrp}_{\mathsf{I}}$ *is defined as* $\pi(\mathrm{Amt}_{i,k})$, *with verification function* $\mathsf{Verify}_{\mathsf{I}}\big(\pi(\mathrm{Amt}_{i,k}), \mathsf{com}(\mathrm{Amt}_{i,k}, r_{i,k})\big) = 1$ *if* $0 \le \mathrm{Amt}_{i,k} < U$; $\mathsf{Verify}_{\mathsf{I}}\big(\pi(\mathrm{Amt}_{i,k}), \mathsf{com}(\mathrm{Amt}_{i,k}, r_{i,k})\big) = 0$ *otherwise.*

This proof ensures that the amount in each payment instruction is non-negative and less than the total liquidity U. This circumvents attacks in which a participant submits a payment instruction with a negative amount in the aim of stealing liquidity from the prospective receiver. Any payment with negative amount should be rejected and considered an attack.

Definition 2. *Let* $\tilde{B}_i^*$ *denotes* P_i*'s to-be-post-balance in her proposal, at the* t^{th} *iteration, defined in Eq* (9). *A type II zero knowledge range proof* $\mathrm{zkrp}_{\mathsf{II}}$ *is defined as* $\pi(\tilde{B}_i^*)$, *with verification function* $\mathsf{Verify}_{\mathsf{II}}\big(\pi(\tilde{B}_i^*), \mathsf{com}(\tilde{B}_i^*, \tilde{r}_i^*)\big) = 1$ *if* $-d_i \le \tilde{B}_i^* < U$; $\mathsf{Verify}_{\mathsf{II}}\big(\pi(\tilde{B}_i^*), \mathsf{com}(\tilde{B}_i^*, \tilde{r}_i^*)\big) = 0$ *otherwise.*

This proof allows anyone to check that the P_i's proposal does not violate the liquidity constraint.

Definition 3. *Let $\tilde{B}'_i$ denotes P_i's hypothetical to-be-post-balance in her proposal, at tth iteration, defined in line 16 of Algorithm 1. A type III zero knowledge range proof* $\mathrm{zkrp}_{\mathsf{III}}$ *is defined as $\pi(\tilde{B}'_i)$, with verification function* $\mathsf{Verify}_{\mathsf{III}}\big(\pi(\tilde{B}'_i), \mathsf{com}(\tilde{B}'_i, \tilde{r}'_i)\big) = 1$ *if $-U < \tilde{B}'_i < -d_i$;* $\mathsf{Verify}_{\mathsf{III}}\big(\pi(\tilde{B}'_i), \mathsf{com}(\tilde{B}'_i, \tilde{r}'_i)\big) = 0$ *otherwise.*

This proof essentially checks whether P_i's proposal in round t is optimal; i.e. P_i is not holding back payments that can be settled.

We note that all of these zero-knowledge range proofs can be implemented using the schemes in either [12] or [13]. For space limitations, details are omitted.

3.3 Solution Description

Algorithm 2 describes our privacy-preserving decentralized netting.

A payment transaction in Algorithm 2 includes the priority of the payment and the identity of the receiver in the clear, the amount however is obfuscated using a Pedersen commitment. The transaction also contains a zero-knowledge range proof that shows that the amount in Pedersen commitment does not exceed the total liquidity U. The sender of a payment signs her transaction revealing thus, her identity and submits it to the ledger. Moreover, the sender transmits the opening of the Pedersen commitment to the intended recipient through a secure channel.

Before the netting session starts, each participant P_i constructs her outgoing payment message queue $\mathbf{PayOutQ}_i = [\mathrm{PayOut}_{i,1}, ..., \mathrm{PayOut}_{i,m_i}]$ with $\mathrm{PayOut}_{i,k} = (\mathrm{Rec}_{i,k}, \mathsf{com}(\mathrm{Amt}_{i,k}, r_{i,k})), \forall k \in [m_i]$. We assume that P_i is associated with an ex-ante balance stored in the ledger as $\mathsf{com}(\hat{B}_i, \hat{r}_i)$.

In each round t, P_i submits a nettable set $\mathbf{x}_i^t$ to the network. She also provides a zero-knowledge range proof that her to-be-post-balance $\tilde{B}^*_i$ encoded in commitment $\mathsf{com}(\tilde{B}^*_i, \tilde{r}^*_i)$ is in the range $[-d_i, U)$. Additionally, P_i provides a zero-knowledge range proof that the hypothetical to-be-post-balance $\tilde{B}'_i$ hidden in commitment $\mathsf{com}(\tilde{B}'_i, \tilde{r}'_i)$ is less than $-d_i$. The hypothetical to-be-post-balance is calculated by including the payment message with the highest priority that cannot be settled (i.e. message with index $h(x_i^t) + 1$ in $\mathbf{PayOutQ}_i$).

It is easy to see that this protocol hides the payment amounts and account balances and guarantees the correctness of nettable set selection. Furthermore, according to Theorem 1, the proposed protocol also achieves fairness.

3.4 Hiding Senders and Receivers

In the current design, only the account balances and payment amounts are hidden using commitments, while the identities of senders and receivers are disclosed. One way to hide these is to express a payment instruction as an n-sized commitment vector, which commits to the payment amount (positive number) for the receiver, and the negative of payment amount for the sender and zero for

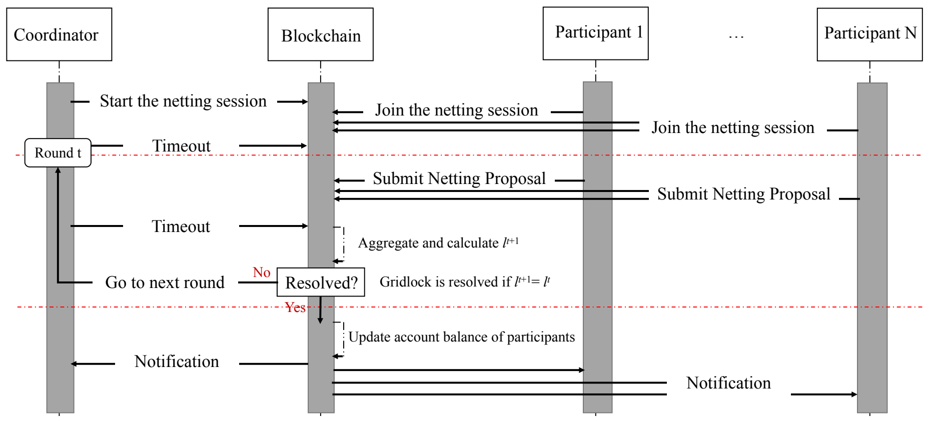

Fig. 2. The detailed decentralized netting protocol illustration

other participants. This is the approach adopted by [9]. Although this approach successfully hides the identities of sender and receiver, it is not scalable: the transaction size, zero-knowledge proof generation and verification times are all proportional to the size of the commitment vector (i.e. the number of participants). Alternatively, participants can inject zero-valued payment instructions to random receivers in the network. These fake payment instructions help disguise the actual instructions. The frequency of these fake payments can be decided based on a trade-off between performance and the desired privacy level.

4 Payment System Construction and Security Analysis

4.1 Blockchain-based Payment System

System Participants We assume that all n participants $P_1, ..., P_n$ have peers running on the blockchain network. We conflate the participants with their peers. We also assume that there is a *system administrator* that initializes the participants' accounts at setup time.

Participant Accounts Each participant has an account stored in the ledger. The account is addressed with the participant's public key and its balance is encoded in a Pedersen commitment. When the system is first bootstrapped, the administrator initializes each account by computing a Pedersen commitment reflecting the current balance. The administrator communicates the opening of each Pedersen commitment to the corresponding participant, so that the latter can submit payment instructions. To counter *front-running attacks*[4], the account of each participant is locked the moment she joins the netting session and then unlocked once the netting session ends.

[4] These attacks send a payment instruction to change account balances while netting is taking place to invalidate the range proofs computed prior to the update.

Payment Instructions and Gross Settlement Each payment instruction identifies the sender and the receiver and includes a Pedersen commitment to the payment amount and a zero-knowledge proof that the amount is non-negative. If the sender has enough liquidity to settle the payment, then she finalizes her transaction (via gross settlement) immediately by submitting to the network a zero-knowledge range proof that her updated account balance, by deducting the payment amount from the current account balance, is non-negative. The network verifies the zero-knowledge proof, updates the sender and receiver's account balance using the homomorphic property of Pedersen commitments and marks the payment instruction as settled. Otherwise, the payment instruction is stored in the sender's outgoing queue based on priority (and time). We recall that the sender is required to send to the receiver the opening of the Pedersen commitment. Our protocol assumes that the participants leverage secure channels to communicate with each other.

Gridlock Resolution and Net Settlement Gridlock [5] is a situation where no participant can proceed to settle her outgoing queue using gross settlement, however, all participants collaboratively may settle their payments simultaneously using net settlement. To that end, all participants engage in the decentralized netting protocol depicted in Algorithm 2. To facilitate netting, our algorithm uses a *coordinator* that acts as a timing service, which initiates netting sessions and keeps track of timeouts in each round, see Fig. 2.

At first, the coordinator submits a request to the the network to start a netting session. Interested participants respond by sending a request to join the netting session. A timeout transaction triggered by the coordinator starts the first round of the netting protocol. In each round, the participants submit proposals as defined in Algorithm 2 line 16 or 20. At the end of each round (triggered by the coordinator's timeout), the blockchain verifies the proposals and aggregates them to calculate the nettable set for the current round. If the nettable set does not change from the previous round, then the algorithm *has converged*. In the absence of a DEADLOCK, net settlement takes place automatically. Namely, the blockchain updates all the involved accounts by adding payment amounts to the receivers' accounts and subtracting the same payment amounts from the senders' accounts. After settlement, the network marks the payment instructions as settled by removing them from the outgoing queue. If the algorithm has not converged yet, a new round starts automatically.

4.2 Trust Model

Participants We assume malicious participants. They may attempt to steal liquidity from other participants, hide liquidity, manipulate liquidity balances, provide false proofs and break the privacy of other participants. However, we assume that if a participant sends a valid payment instruction, then she would provide the corresponding recipient with the correct Pedersen decommitment.

Shared Ledger We assume the ledger to be **live**: *valid* payment transactions will eventually be stored in the ledger. It is also assumed to be **immutable**: once a transaction is stored in the ledger it cannot be removed or modified. These two properties ensure that the ledger will always reflect an up-to-date version of the account balances of the participants. The ledger is available at all times to all participants in the system to submit their transactions and can be constructed on a platform using either a crash-tolerant consensus protocol like Hyperledger Fabric [14], or Nakamoto-like consensus protocol [2] like Ethereum.

Robustness The network waits for all participants to submit a proposal in each round. If the proposal of some participant suffers delays, then our solution simply times out and excludes all incoming and outgoing payment instructions of that party. This guarantees that the efforts of honest parties are not wasted.

Security The security of the underlying blockchain consensus algorithm guarantees that the smart contract verifying the payment instructions and the proposals be executed correctly. We recall that the smart contract logic consists of checking a set of zero-knowledge range proofs that ensure that the system predefined rules and invariant are not violated. Thanks to the soundness property of zero-knowledge proofs, no participant can make the smart contract accept an *invalid* payment transaction or proposal.

5 Implementation

5.1 Proof of Concept Implementation

For evaluation purposes, we implemented an experimental decentralized blockchain payment system on Hyperledger Fabric 1.2. Our zero-knowledge range proofs are based on Boneh-Boyen signature [12]. We use the Go's official BN256 curve [15], a bilinear group with 128-bit security level, to compute Pedersen commitments and range proofs. Our implementation consists of a number of Fabric chaincode (smart contract) interfaces that combined deliver the functionalities of payment, netting and settlement. Appendix B lists the main functions we have implemented and the code could be found at [16]. We set the credit limit to 0 for all participants and we enforce the priority constraint as defined in Eq. (12). We ran our experiments on an Ubuntu 16.04 with Linux kernel version of 4.4.0-133 virtual machine, with 32 VCPUs of 2.0 GHZ and 48 GB memory.

5.2 Protocol Evaluation

In our evaluation, a payment message is settled immediately if the sender has sufficient funds and no other higher priority payment messages are in her outgoing queue. Otherwise, the payment message is added to the the outgoing queue according to its priority. Once a participant receives an incoming payment, she tries to locally settle as many outgoing payments as possible while respecting

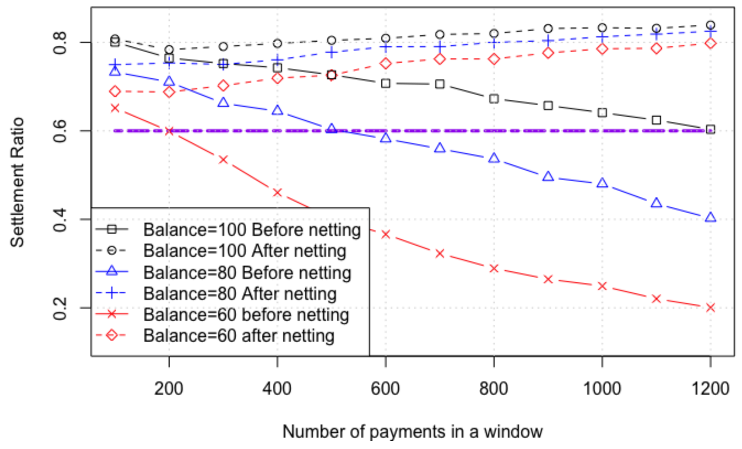

Fig. 3. The settlement ratio before and after netting versus number of payment messages in a window for different initial account balances = 80, 60, 100. Pareto parameter: $X_m = 20, \alpha = 2$

their priorities. At some predefined time, all participants engage in the decentralized netting protocol and conduct a multi-lateral net settlement based on the netting result. We note that periodical netting improves the settlement ratio of the payment system, which is defined as the ratio of the settled payment messages to the total number of messages. In our experiment, we randomly pick a sender and a receiver for each payment message and we draw the value of the payment from a Pareto distribution. We call a *window* the time between the start of two consecutive netting sessions.

Our experiment involves 10 participants and injects different number of payment messages per window into the system. Fig 3 plots the settlement ratio before and after running the netting protocol against the number of payment messages, for different initial account balances. The more payment messages we inject, the more netting improves the settlement ratio. It is intuitive that as the account balance increases, the probability of a gridlock decreases and the settlement ratio before netting improves. In a practical setting, e.g. we want to keep the settlement ratio always above 60% (horizontal line), netting must take place for every 200, 500 and 1200 messages for initial account balance of 60, 80 and 100 respectively. As the initial account balance decreases, netting must take place more frequently to keep the settlement ratio high. Fig 4 shows the average number of rounds to reach convergence versus the number of payment messages for different initial account balances. As the number of payment messages increases in a window, it takes more rounds to reach convergence (the longer is the window); as the account balance decreases, it takes slightly longer to reach convergence. However, the total number of rounds increases only logarithmically.

Besides the Boneh-Boyen signature (BBS) based range proof, we also implemented another variety: Borromean ring signature (BRS) based proof [10]

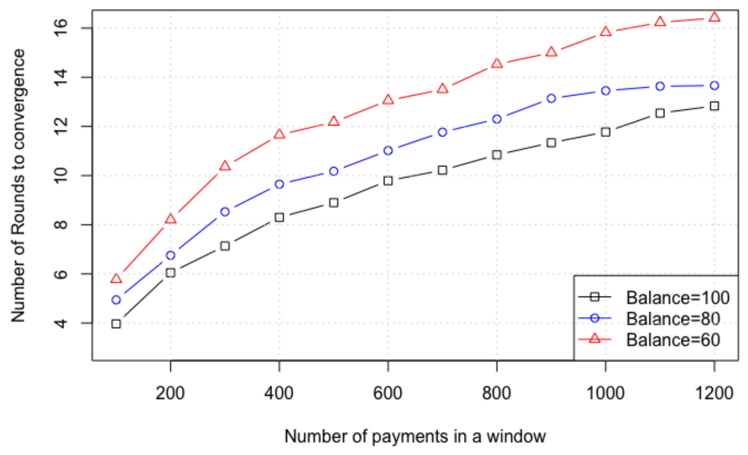

Fig. 4. The total number of rounds to reach convergence versus number of payment messages per window for different initial account balances = 80, 60, 100. Pareto parameter: $X_m = 20, \alpha = 2$

Table 1. Creation and verification time and proof size of various components

Component	Prove	Verify	Size
BBS $[0, 10^4]$	249ms	363ms	2624B
BBS $[0, 10^8]$	485ms	724ms	4928B
BBS $[0, 10^{16}]$	963ms	1420ms	9536B
BRS $[0, 10^4]$	17 ms	22ms	1664B
BRS $[0, 10^8]$	30ms	40ms	3360B
BRS $[0, 10^{16}]$	58ms	83ms	6688B

and compared their performances. We used the elliptic curve secp256k1[18] for BRS-based range proof. Table 1 compares the time to generate and verify the zero-knowledge range proofs for different ranges using different proof methods. Table 1 also compares the size of various proofs, which is proportional to $\log(L)$, with L being the size of the range. Our implementation shows that BRS-based range proofs have better performances and slightly smaller proof size than BBS-based range proofs.

6 Conclusion

This paper presents a possible approach to design a truly-decentralized netting algorithm without compromising any security or privacy requirement. The proposed solution is optimal while still being relatively efficient as the evaluation results show. As a future work, we plan to 1.) evaluate our protocol using more efficient zero-knowledge range proofs e.g. Bulletproofs [13] and 2.) explore decentralized solutions for more general netting problems.

References

1. A. Furgal, R.Garratt, Z. Guo, and D. Hudson: A Proposal for a decentralized liquidity savings mechanism with side payments. R3 Report 2018.
2. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf
3. J. Chapman, R. Garratt, S. Hendry, A. MacCormack and W. McMahon: Project Jasper: Are distributed wholesale payment systems feasible yet. https://www.bankofcanada.ca/wp-content/uploads/2017/05/fsr-june-2017-chapman.pdf.
4. Project Ubin Phase 2: https://www.mas.gov.sg/schemes-and-initiatives/Project-Ubin
5. M. Bech and K. Soramki: Gridlock Resolution in Interbank Payment Systems. Discussion Paper 9/2001, Bank of Finland.
6. Gntzer, M., Jungnickel, D., and Leclerc, M.: Efficient algorithms for the clearing of interbank payments. European Journal of Operational Research 106.1 (1998): 212-219.
7. Shafransky, Y., and Doudkin, A.: An optimization algorithm for the clearing of interbank payments. European Journal of Operational Research 171.3 (2006): 743-749.
8. X. Wang, X. Xu, L. Feagan, S. Huang, L. Jiao, W. Zhao: Inter-Bank Payment System on Enterprise Blockchain Platform. IEEE CLOUD 2018 Cloud and Blockchain Workshop.
9. N Narula, W. Vasquez and M. Virza: zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association.
10. A. Poelstra, A. Back, M. Friedenbach, G. Maxwell and P. Wuille: Confidential assets. In Financial Cryptography Bitcoin Workshop. https://blockstream.com/bitcoin17-final41.pdf
11. T. P. Pedersen: Non-interactive and information-theoretic secure verifiable secret sharing. In Proceedings of the 11th Annual International Cryptology Conference, pp. 129-140.
12. J. Camenisch, R. Chaabouni, A. Shelat: Efficient protocols for set membership and range proofs. ASIACRYPT 2008, LNCS, vol. 5350, pp. 234-252.
13. B. Bunz, J. Bootle, D. Boneh, A. Peolstra, P. Wuille and G. Maxwell: Bulletproofs: Short proofs for Confidential Transactions and more. In Security and Privacy (SP), 2018 IEEE Symposium on (2018), IEEE.
14. Hyperledger Fabric 1.2: https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatis.html
15. bn256. [Online] Available at: https://golang.org/x/crypto/bn256
16. Blockchain based payment system and netting protocol implementation. [online] Available at: http://github.com/blockchain-research/gridlock
17. Borromean Ring signature based zero-knowledge range proof implementation. [online] Available at: http://github.com/blockchain-research/crypto
18. Package btcec implements support for the elliptic curves needed for Bitcoin., July 2017.https://godoc.org/github.com/btcsuite/btcd/btcec

A Proof Of Optimality

We first prove for $f_i = T_i(\mathbf{x})$, then we show the result is invariant to the choice of f_i. Without loss of generality, we also assume $d_i = 0, \forall i$ to simplify proof notation. Let $\mathbf{m}=[m_1, m_2, \ldots, m_n]^\top$ and $\mathbf{T}=[T_1, T_2, \ldots, T_n]^\top \in R^n$. The problem defined by equations (7) to (12) can be rewritten as

$$\max_{\mathbf{0}\leq\mathbf{T}\leq\mathbf{m}} \sum_{i=1}^{n} T_i \tag{17}$$

$$\text{s.t. } \tilde{B}_i \overset{\text{def}}{=} \hat{B}_i - S_i(T_i) + R_i(\mathbf{T}) \geq 0,\ \forall i \in [n] \tag{18}$$

$$S_i(T_i) \overset{\text{def}}{=} \sum_{k=1}^{T_i} \text{Amt}_{i,k} \tag{19}$$

$$R_i(\mathbf{T}) \overset{\text{def}}{=} \sum_{j=1}^{n} \sum_{k=1}^{T_i} \text{Amt}_{j,k} I(\text{Rec}_{j,k} = i) \tag{20}$$

where $\mathbf{0} \leq \mathbf{T} \leq \mathbf{m}$ stands for $0 \leq T_i \leq m_i, \forall 1 \leq i \leq n$. Note that the definitions of $S_i(T_i)$ and $R_i(\mathbf{T})$ above implicitly model the constraints defined in (12) for each participant i. In other words, if there are T_i payments settled for participant i, $S_i(T_i)$ imply that they must be the first T_i payments in $\mathcal{Q}_i$. Let $\mathbf{T}^t$ denote the value of $\mathbf{T}$ at the tth iteration of Algorithm 1. In addition, $\mathbf{T}^0$ is set to $\mathbf{m}$ for initialization. Then Algorithm 1 essentially becomes

- Initialization: $\mathbf{T}^0 \overset{\text{def}}{=} \mathbf{m}$
- Repeat following steps
  - Calculate $R_i(\mathbf{T}^t), \forall i \in [n]$
  - $\forall i \in [n]$ find

$$T_i^{t+1} = \text{argmax}_T \Big\{ T \in [m_i] \Big\} \tag{21}$$

such that

$$\hat{B}_i - S_i(T_i^{t+1}) + R_i(\mathbf{T}^t) \geq 0 \tag{22}$$

$$x_{i,k+1} \leq x_{i,k}, \forall k \in [m_i - 1] \tag{23}$$

  - If $\mathbf{T}^{t+1} = \mathbf{T}^t$, stop. Otherwise, continue the loop.

The decentralized netting protocol is guaranteed to find the optimal solution. To prove this, we first prove that line 6-12 in Algorithm 1 is equivalent to Eqs. 21-23 above.

By the exit condition, we have $\tilde{B}_i^{t+1} \geq 0$. Therefore we could construct the following case, where

$$\tilde{B}_i^{t+1} = 0 \implies \hat{B}_i - S_i(T_i^{t+1}) + R_i(\mathbf{T}^t) = 0 \tag{24}$$

Suppose there exists another optimal solution $\mathcal{T}_i > T_i^{t+1}$ and

$$\tilde{B}_i^{\mathcal{T}} = \hat{B}_i - S_i(\mathcal{T}_i) + R_i(\mathbf{T}^t) \tag{25}$$

Since

$$S_i(\mathcal{T}_i) = \sum_{k=1}^{\mathcal{T}_i} \text{Amt}_{i,k} > S_i(T_i^{t+1}) = \sum_{k=1}^{T_i^{t+1}} \text{Amt}_{i,k} \tag{26}$$

it impies that

$$\tilde{B}_i^{\mathcal{T}} < \tilde{B}_i^{t+1} = 0 \tag{27}$$

Eq.27 clearly violates the non-overdraft condition. Therefore such $\mathcal{T}$ does not exist and T_i^{t+1} is the maximum value that can be achieved at $t+1$th iteration. Furthermore, we have

$$h(\mathbf{x}_i^{t+1}) = \max_k (I(x_{i,k}^{t+1}) = 1) \tag{28}$$

$$h(\mathbf{x}_i^{t}) = \max_k (I(x_{i,k}^{t}) = 1) \tag{29}$$

In view of line 12 in Algorithm 1, the above two equations imply that

$$h(\mathbf{x}_i^{t+1}) < h(\mathbf{x}_i^{t}) \implies x_{i,k}^{t+1} < x_{i,k}^{t} \tag{30}$$

$$\implies T_i^{t+1} < T_i^{t} \tag{31}$$

We note that the decentralized netting protocol starts with all the payment in queue and removes current invalid payments for each deficient participant. The optimality of T_i at each itertion plus the monotonicity of T_i over iterations guarantee that the first feasible solution will also be the optimal solution and it is unique.

Next, we show its invariance. If there is only one feasible solution, then it also achieves the maximum total value and number of payments. If there are two or more feasible solutions, the monotone decreasing of T_i imply that any other feasible solution after the first one contains same or fewer payments for each participant and thus less value. This completes the proof.

B Functions of the smart contract

In Table 2, we describe the detailed functions of our implemented smart contract.

Table 2. Functions and logic of smart contract

Functions	Logics
mintAccount	Smart contract initializes each participant's account with commitments ($cm_b = g^b h^r$) to their balances (b) and verifying zkrp ($U > b \geq 0$)
addMessage	Smart contract adds a payment message to the system with commitment ($cm_a = g^a h^r$) to payment amount (a) and verifying zkrp ($U > a \geq 0$)
grossSettlement	Smart contract settle the first payment in the outgoing queue, update account balance ($cm'_b = cm_b - cm_a$) and verify zkrp ($U > b' = b - a \geq 0$)
proposeNettableSet	Smart contract update a participant's gridlock proposal, verifying two zkrps (refer to the protocol)
tallyGridlockProposal	Smart contract calculate and check the new global nettable set. If it is the same as previous round, the gridlock resolution protocol converges
netSettlement	Smart contract settle all payment messages in the nettable set and update all parties' account balances for a successful gridlock resolution