# Wiser: Increasing Throughput in Payment Channel Networks with Transaction Aggregation

Samarth Tiwari
samarth.tiwari@cwi.nl
Centrum Wiskunde & Informatica
The Netherlands

Michelle Yeo
michelle.yeo@ist.ac.at
Institute of Science and Technology
Austria

Zeta Avarikioti
georgia.avarikioti@tuwien.ac.at
TU Wien, Austria

Iosif Salem
iosif.salem@inet.tu-berlin.de
TU Berlin, Germany

Krzysztof Pietrzak
krzysztof.pietrzak@ist.ac.at
Institute of Science and Technology
Austria

Stefan Schmid
stefan.schmid@tu-berlin.de
TU Berlin, Germany & University
of Vienna, Austria

## ABSTRACT

Payment channel networks (PCNs) are one of the most prominent solutions to the limited transaction throughput of blockchains. Nevertheless, PCNs suffer themselves from a throughput limitation due to the capital constraints of their channels. A similar dependence on high capital is also found in inter-bank payment settlements, where the so-called netting technique is used to mitigate liquidity demands.

In this work, we alleviate this limitation by introducing the notion of transaction aggregation: instead of executing transactions sequentially through a PCN, we enable senders to aggregate multiple transactions and execute them simultaneously to benefit from several amounts that may "cancel out". Two direct advantages of our proposal is the decrease in intermediary fees paid by senders as well as the obfuscation of the transaction data from the intermediaries.

We formulate the transaction aggregation as a computational problem, a generalization of the Bank Clearing Problem. We present a generic framework for the transaction aggregation execution, and thereafter we propose WISER as an implementation of this framework in a specific hub-based setting. To overcome the NP-hardness of the transaction aggregation problem, in WISER we propose a fixed-parameter linear algorithm for a special case of transaction aggregation as well as the Bank Clearing Problem. WISER can also be seen as a modern variant of the *Hawala* money transfer system, as well as a decentralized implementation of the overseas remittance service of Wise.

## KEYWORDS

Payment Channel Networks, Transaction Aggregation, Netting, Fixed Parameter Tractability, Privacy, Throughput

## 1 INTRODUCTION

### 1.1 Motivation

Payment channel networks (PCNs) [12, 13, 29, 33] are a promising technology to significantly increase the transaction throughput of blockchain-based cryptocurrencies such as Bitcoin [30]. Two parties can open a *payment channel* by locking coins in a joint account on-chain. Thereafter, the parties can transact off-chain via updating the distribution of the coins in the channel to depict the new transaction. Only if the parties want to close the channel or allocate extra budget, do they need to go on-chain. Therefore, payment channels can facilitate arbitrary many transactions off-chain with a constant number of on-chain transactions, significantly increasing the transaction throughput. These payment channels collectively constitute a *payment channel network* (PCN) which allows two parties to transact off-chain even if they do not share a payment channel by using other parties as intermediaries in routing their transactions [13, 33].

In order to provide high and reliable transaction throughput, PCNs require many well connected high-capacity payment channels. The reason is that channels may quickly deplete if several transactions are forwarded along a channel or path in the same direction, thus preventing any further use of the channel in that direction. However, locking large amount of capital for a long period of time comes with a cost. To mitigate this cost, the users of PCNs ask for a service fee for relaying transactions of others through their channels. As a result, a business opportunity arises: wealthy individuals can act as *(transaction) hubs* by establishing well-connected high-capacity nodes, offering routing of transactions as a service. In this way, capital reserves can be used to generate passive income.

While this business model benefits the PCN's users, it also introduces several challenges: (a) the liquidity of the hubs still poses a constraint to the PCN's overall liquidity and hence limits the transaction throughput, (b) the transaction fees that are typically proportional to the transaction value may be high, and (c) the hubs may learn a significant amount of transactions, thus compromising the privacy of users.

To address these challenges, we introduce the notion of *transaction aggregation* (Figure 1). In high level, we enable users

to aggregate multiple transactions and execute them simultaneously in the PCN instead of sequentially, i.e., one-by-one. This way multiple transactions may "cancel out" effectively reducing the transferred amounts and the corresponding transaction fees, thus increasing the PCN's liquidity and consequently the transaction throughput. Moreover, the transaction aggregation obfuscates the individual transactions routed through a channel, as only the aggregated amount is executed.

A closely related notion to transaction aggregation in finance is *netting* that deals with the aggregation of various financial obligations, say, to mitigate risk. In particular, the application of netting in inter-bank payment systems essentially tackles the first aforementioned challenge. The computation problem of optimal netting, i.e., optimal aggregation of payments across traditional banks, is known as the Bank Clearing Problem or BCP. The transaction aggregation problem we introduce in this work is a more generic formulation of the BCP problem, hence demonstrating the potential impact of our work in the financial landscape even beyond blockchains.

Transaction aggregation can also be seen as a decentralized and privacy-enhancing variant of the payment system Wise (formerly TransferWise) [46] built on payment channel networks. Wise is a financial technology allowing convenient online transactions, often used to cheaply transfer money abroad. Another similar financial system is *Hawala* [35], a centuries-old system of fund transfer across long distances and overseas. A system that implements transaction aggregation can be interpreted as a modern and improved digital version of the *Hawala* money transfer system, which is still in use in parts of the Middle East and South Asia.

## 1.2 Contribution

The goal of this work is to address the three challenges of PCNs in the hub-based business model, i.e., (a) improve the throughput of payment channel networks while (b) minimizing fees in the process and also (c) accounting for privacy concerns. To do so, we propose a system which allows hubs to act cooperatively, combining their capital to serve all their clients. Any two users connected to any of the hubs can thereafter transact thought the hubs. The transactions collected over a fixed time interval are then aggregated into a single monetary flow through the network. This flow is consequently executed atomically through the PCN, meaning that either all the selected transactions are executed or none. We term the computational problem of finding a subset of transactions that can be aggregated into a feasible resultant flow as the *transaction aggregation problem* (Figure 1).

Specifically, we present a generic framework which combines three building blocks: (a) a protocol to enable optimal liquidity usage among hubs, (b) an efficient algorithm for the transaction aggregation problem, and (c) a protocol for atomic multi-channel state updates, i.e., the atomic execution of the selected transaction in the PCN. We present a skeleton protocol, that is an abstraction of the composed protocols, to illustrate its benefits as well as technical challenges. We further determine the necessary properties for such a system and show that our abstraction satisfies them.
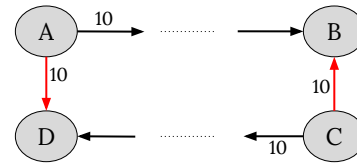


**Figure 1: Transaction aggregation example. User A wants to make a payment of 10 coins to user B and user C wants to make a payment of 10 coins to user D. The A-to-B and C-to-D payment paths might render the transaction infeasible and incur routing fees linear to the path length. Instead, if A pays D and C pays B, each 10 coins, the transactions success depends on the capacity of fewer channels and the fees are reduced.**

We then present WISER, a concrete implementation of the generic framework. WISER uses channel factories [9] and Thora [1] as substantiations of the aforementioned building blocks ((a) and (c) respectively). As the final building block (b), we present a fixed-parameter linear[1] algorithm to solve the transaction aggregation problem, which we first show to be NP-hard. For the execution of our algorithm, we leverage secure multi-party computation (MPC) to conceal channel balances and transaction data.

We further derive a connection between transaction aggregation and netting, and exploit this connection to enrich both notions. On one hand, our fixed-parameter linear algorithm developed for WISER is directly applicable to BCP. Our algorithm exhibits the best known parameterized complexity, improving dependence on number of transactions from pseudo-polynomial to linear. On the other hand, algorithms for the Bank Clearing Problem with inferior asymptotic complexity may still be faster in certain parameter regimes. WISER can be implemented with any of these optimization algorithms as a building block, so that work on BCP also bolsters the versatility of our protocol.

## 1.3 Organisation

The remainder of this paper is organized as follows. In Section 2, we present the model and additional assumptions, as well as the proposed skeleton protocol for solving the transaction aggregation problem and the desired properties it should fulfil. In Section 3, we present our implementation of the skeleton protocol, namely WISER, and in Section 4 we prove that our implementation provides the desired guarantees. In Section 5 we demonstrate the connection to the netting problem and discuss further extensions of our protocol. Finally, in Section 6 we compare to related work and we conclude with Section 7.

## 2 MODEL AND PROTOCOL OVERVIEW

In this section, we first describe payment channel networks, and then introduce the assumptions of our work. Later, we formulate transaction aggregation as an optimization problem, then provide a high-level overview of our solution (skeleton protocol), and conclude the section by defining the desired properties for our system.

---

[1]FPL is a subclass of the fixed-parameter tractable (FPT) complexity class [15].

## 2.1 Payment Channel Networks

In this subsection, we provide a brief introduction to payment channel networks with standard fees as they currently operate.

We assume that a payment channel is characterized by the public keys of the two users opening it, and the funds and forwarding fees that each user adds to each direction of the channel upon its creation [33]. Let $u, v$ be two users that have opened a payment channel $\{(u,v), (v,u)\}$ and $c(e), bf(e), pf(e), e \in \{(u,v), (v,u)\}$ be the current capacity, the base forwarding fee, and the proportional forwarding fee of each channel direction, respectively. Users $u$ and $v$ can do an arbitrary number of payments to each other by subtracting any non-negative amount $x \leq c(e), e \in \{(u,v), (v,u)\}$, and increasing $c(e')$ by $x$, where $e' \in \{(u,v), (v,u)\} \setminus \{e\}$. Note that the forwarding fees and $c(u,v) + c(v,u)$ is public information, while each summand (current balance) is not.

A payment channel network (PCN) is the network built over all payment channels. Users that do not share payment channels can perform payments via paths in the network, by covering a forwarding cost to the intermediate nodes of the path. For example, if payment channels exist between $u, v$ and $v, w$, but not between $u, w$, then $u$ can transfer an amount $x$ to $w$ in two steps: (i) $u$ transfers $x + (bf(v,w) + pf(v,w) \cdot x)$ coins to $v$ and (ii) $v$ transfers $x$ coins to $w$, i.e. $bf(v,w) + pf(v,w) \cdot x$ are the forwarding fees charged by $v$. The transaction is feasible only if $x + (bf(v,w) + pf(v,w) \cdot x) \leq c(u,v)$ and $x \leq c(v,w)$.

In general, suppose that user $s$ wants to pay $x$ coins to user $r$, via a payment path $\{(s, u_1), ..., (u_k, r)\}$. Then $u_k$ receives $rcv_k = x + (bf(u_k, r) + pf(u_k, r) \cdot x)$ and forwards $x$ coins to $r$, $u_{k-1}$ receives $rcv_{k-1} = x + (bf(u_{k-1}, u_k) + pf(u_{k-1}, u_k) \cdot rcv_k)$ and forwards $rcv_k$ coins to $u_k$, and so on. Moreover, the transaction is feasible only if every node in the path has sufficient capacity to route the forwarded amount. This is not always the case, since current balances are private and senders might have to try different payment paths towards a successful transaction.

A transaction along a payment path is successful when all intermediate transactions go through, and otherwise none of them is performed. That is, PCNs must ensure atomicity in transaction routing. In the Bitcoin Lightning Network, this is ensured by Hash Timelock Contracts (HTLCs) [33].

## 2.2 Assumptions

In this subsection, we introduce the cryptographic and network assumptions as well as the considered PCN topology.

*Cryptographic assumptions.* We assume the existence of secure communication channels between users.

*Blockchain and network model.* We assume the underlying block-chain satisfies persistence and liveness as defined in [19]. We also assume a synchronous network model. That is, there is a known network delay which bounds the time needed for any user to receive any incoming message.

*Studied PCN topology.* We restrict the PCN topology to allow for a computationally tractable implementation of transaction aggregation. Let $V = H \cup C$ be the set of all users, which
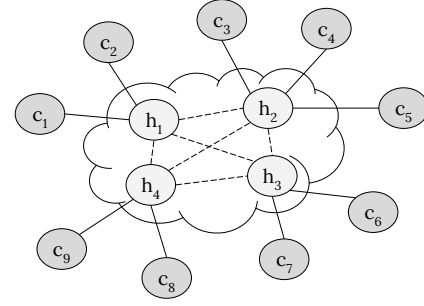


**Figure 2: Example of the restricted PCN topology. Each client (nodes $\{c_1, \ldots, c_9\}$) is connected to one of hub nodes (nodes $\{h_1, \ldots, h_4\}$) and the hub nodes are well interconnected (potential edges are dashed). We propose a solution for hub connectivity in our protocol description.**

we split to a relatively small set of hub nodes $H$ and the remainder set of client nodes $C$. The hubs are well interconnected, and the client nodes are those connected to only one hub (Figure 2).

*Users and transactions.* We assume the users of the protocol follow the protocol specification unless they can monetarily gain from deviating from it.

Over time, users accumulate transactions before initiating the protocol. The list $\mathcal{T}_u$ of transactions that user $u$ submits must not exceed the capacity of the channel $u$ has with their hub $h_u$. In other words, users must not authorize a total payment of funds exceeding their balance with the hub. Moreover, the same must also hold for the set of incoming transactions where $u$ is the recipient.

*Fee function.* We assume that the forwarding fees charged by hubs for each channel are publicly known before the protocol execution. We also assume that the forwarding fee is a non-negative function $F(a, b)$ of the initial and final states $a, b$ of the channel and that the triangle inequality holds: for all states $a, b, c$: $F(a, c) \leq F(a, b) + F(b, c)$. As described in Section 2.1 above, the standard fee functions are simply a base fee plus a proportional fee, which do satisfy our more general assumptions. Intuitively, it should not be cheaper to forward a given amount along a channel in multiple parts compared to one.

Note that our proposed solution does not aim to minimize the fees, but rather to maximize the volume of transactions that are cleared. We nevertheless show that transaction aggregation implicitly reduces incurred fees for the above class of fee functions. We discuss a more sophisticated consideration of fees in Section 5.

## 2.3 (Computational) Problem Definition

Let $G = (V, E)$ be a directed graph that models a PCN with integral edge capacities $c(e)$. Let $V = \{v_1, \ldots v_n\}, |E| = m$. A payment channel between $v_i$ and $v_j$ is modelled as two unidirectional edges $(v_i, v_j)$ and $(v_j, v_i)$.

A flow through this network is an $m$-dimensional vector $\mathbf{f} = (\mathbf{f}(e))_{e \in E}$ that satisfies the capacity constraints:

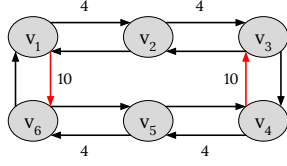$$0 \leq \mathbf{f}(e) \leq c(e) \quad \forall e \in E.$$

**Figure 3: A transaction aggregation example. The non-depleted edges are annotated with their capacity (e.g. $c(v_2, v_1) = 0$, while $c(v_1, v_2) = 4$). Let $\mathcal{T} = [(10, 0, -10, 0, 0, 0), (10, 0, -10, 0, 0, 0), (0, 0, 0, 10, 0, -10)]$, i.e., two transactions of 10 coins from $v_1$ to $v_3$ and one transaction of 10 coins from $v_4$ to $v_6$. These transactions are non feasible when done sequentially. However, $\mathcal{T}^* = [(10, 0, -10, 0, 0, 0), (0, 0, 0, 10, 0, -10)]$ is feasible by the flow noted with red edges ($(v_1, v_6)$ and $(v_4, v_3)$). That is, the flow $\mathbf{f}$ such that $\mathbf{f}(v_1, v_6) = \mathbf{f}(v_4, v_3) = 10$ and $\mathbf{f}(e) = 0$ otherwise, realizes the demand vector $\mathbf{d} = (10, 0, -10, 10, 0, -10)$, which is the sum of the transactions in $\mathcal{T}^*$.**

A demand vector $\mathbf{d} = (d_1, \dots d_n) \in \mathbb{Z}^n$ is a vector representing the monetary flow through each vertex. For instance, a transaction of 1 unit from $v_1$ to $v_2$ can be represented as the demand vector $(1, -1, 0, \dots 0)$. A flow $\mathbf{f}$ is said to route a demand vector $\mathbf{d}$ if for every vertex $v$:

$$\sum_{(v,u) \in E} \mathbf{f}(v, u) - \sum_{(u,v) \in E} \mathbf{f}(u, v) = \mathbf{d}(v)$$

A demand vector is feasible if there is a flow that routes it. For example, the demand vector $(1, -1, 0, \dots 0)$ representing a transaction of amount 1 from $v_1$ to $v_2$ is feasible if, say, there is an edge (payment channel) from $v_1$ to $v_2$ of capacity at least 1.

Consider a list of transactions $\mathcal{T} = [\mathbf{t}_1, \dots \mathbf{t}_k]$ each represented as a demand vector, i.e., $\mathbf{t}_i$ is a demand vector with $w_i$ in the position of the sender, $-w_i$ in the position of the receiver, and zero otherwise, where $w_i$ is the transaction amount of $\mathbf{t}_i$. We denote by $|\mathbf{t}_i| := w_i$ the amount of transaction $\mathbf{t}_i$. We define *transaction aggregation* as the process of:
(i) algebraically adding up all the demand vectors in $\mathcal{T}$, to obtain a demand vector $\mathbf{d} = \sum_{\mathbf{t} \in \mathcal{T}} \mathbf{t}$,
(ii) computing a flow $\mathbf{f}$ that routes $\mathbf{d}$, and then
(iii) executing this flow on the payment channel network as a means of executing all the transactions in $\mathcal{T}$.

The computational challenge of transaction aggregation lies in finding a subset $\mathcal{T}'$ whose aggregate demand vector can feasibly be routed, which we define below.

*Definition 2.1.* The Transaction Aggregation Problem on a directed graph $G$ concerns computing a sublist $\mathcal{T}^* \subseteq \mathcal{T} = [\mathbf{t}_1, \dots \mathbf{t}_k]$ that can feasibly be aggregated, and which is optimal in terms of total throughput $\sum_{\mathbf{t}_i \in \mathcal{T}^*} |\mathbf{t}_i|$.

Next, we reformulate the Bank Clearing Problem [21, 38] in our notation, noting it to be a special case of transaction aggregation.

*Definition 2.2.* The Bank Clearing Problem with $n$ participants $\{v_1, \dots, v_n\}$, each with current balance (also known as cover money) $\mathbf{b} = (b_1, \dots b_n)$ concerns computing a sublist
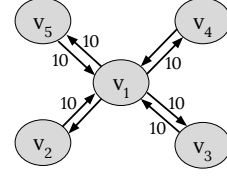


**Figure 4: Examples of cancelling out of transactions. Capacities are annotated only on the edges relevant to the examples. We consider two examples: $\mathcal{T}_1 = [(0, 5, 0, -5, 0), (0, 0, 5, 0, -5)]$ and $\mathcal{T}_2 = [(0, 0, 5, 0, -5), (0, 0, -2, 0, 2)]$. Transaction aggregation benefits $\mathcal{T}_2$, but not $\mathcal{T}_1$. That is, aggregating $\mathcal{T}_2$ produces $(0, 0, 3, 0, -3)$ (reduced flow, one transaction instead of two), while aggregating $\mathcal{T}_1$ outputs $(0, 5, 5, -5, -5)$ (equivalent to sequential transaction execution).**

$\mathcal{T}^* \subseteq \mathcal{T} = [\mathbf{t}_1, \dots \mathbf{t}_k]$ that can be aggregated into a demand vector $\mathbf{d}^*$ such that $\mathbf{d}^* \le \mathbf{b}$, and which is optimal in terms of total throughput (also known as clearing volume) $\sum_{\mathbf{t}_i \in \mathcal{T}^*} |\mathbf{t}_i|$.

Let us briefly elaborate on the throughput benefits of transaction aggregation. First, it may be that the complete list $\mathcal{T}$ cannot be aggregated into a feasible demand vector, but sublists of transactions can (Figure 3). Figure 3 also illustrates that routing an aggregate may require less flow along channels, or use shorter paths, and thus incur lower fees. Secondly, there is a varying degree of cancelling out of the demand vector $\mathbf{d}$, depending on transactions in $\mathcal{T}$. That is, transactions might be completely independent (payment paths are disjoint), or all of them can cancel out (e.g. $s$ and $r$ want to pay each other the same amount), leading to a smaller demand vector, or anything between these two extremes. This form of canceling out is beneficial regardless of channel capacities. We give an example in Figure 4. Note that the cancelling out feature of transaction aggregation might render a list of transactions feasible, even though channel capacities do not suffice for sequential execution.

In what follows, we will refer to oracles $O$ as algorithms that can solve the transaction aggregation problem, possibly under certain assumptions on $G$ and $\mathcal{T}$, to return a feasible sublist $\mathcal{T}^*$ that maximizes a linear function of $\mathcal{T}$. The transaction aggregation problem is generally computationally hard as we show in Section 4.

## 2.4 Protocol overview

In this section, we introduce a protocol abstraction or the "skeleton" protocol for solving and executing the transaction aggregation problem in a payment channel network formulated as described in Section 2.3 under our assumptions.

Our skeleton protocol consists of two phases: a flow computation phase and an execution phase. The goal of the flow computation phase is to privately compute and output an optimal transaction flow from the list of transaction requests submitted to the protocol. The goal of the execution phase is to execute the flow in an atomic fashion. An illustration of the flow of the skeleton protocol can be found in Figure 5.
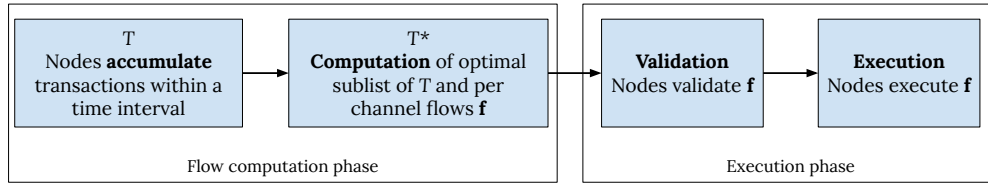
| | | | | |
|---|---|---|---|---|
| **T** Nodes **accumulate** transactions within a time interval | → | **T\*** **Computation** of optimal sublist of T and per channel flows **f** | | |
| | Flow computation phase | | | |

| | | |
|---|---|---|
| **Validation** Nodes validate **f** | → | **Execution** Nodes execute **f** |
| | Execution phase | |

**Figure 5: Skeleton protocol for transaction aggregation.**

*Flow computation phase.* The flow computation phase initiates when a pre-specified amount of time elapses or when sufficiently many transactions have been collected by the users[2]. In this phase, the list of transactions $\mathcal{T}$ as well as the respective channel balances are taken as input. The output of the flow computation phase is a sublist of transactions $\mathcal{T}^*$ that will be eventually executed in the PCN, along with a resultant flow $\mathbf{f}$ that routes the aggregate of these transactions.

*Execution phase.* Out of the computed solution $\mathcal{T}^*$ and $\mathbf{f}$, individual users $v$ are provided with the sublist of $\mathcal{T}^*$ containing only those transactions where $v$ is a sender or recipient, and the flow $\mathbf{f}(e)$ for edges $e$ adjacent to $v$. With this data, each user verifies the validity of the solution, and that balance security holds, i.e., the user does not lose any money given the resulting flow and the transactions to be executed. After this verification, the users must execute $\mathbf{f}$ atomically. The flow output from the previous flow computation phase might involve disconnected components of the network; therefore, simple HTLCs may not suffice to atomically execute the transactions $\mathcal{T}^*$. Instead, a clever solution is needed where atomicity of all transactions in $\mathcal{T}^*$ is maintained regardless of the graph topology.

We will introduce the specific protocols that will be leveraged to implement the skeleton protocol described above in Section 3.

## 2.5 Desired Properties

In this section, we discuss the desiderata of our protocol WISER and then formally define the protocol goals.

First, we demand that our protocol is practical, i.e., that the solution of the transaction aggregation problem is computationally feasible. Second, WISER should maintain security of the channel balances, meaning that the users should not lose money due to the protocol execution. In particular, the intermediaries (hubs) should not lose any money, while the users should only have a deficit equal to the sum of the amounts of the selected transactions in $\mathcal{T}^*$. This security property is encapsulated by *Balance Security*.

Furthermore, we demand two optimization properties: one that ensures that our solution transfers the maximum amount of value possible, termed *Optimality*, and one that ensures our solution is cheaper (or equivalent at worse) than executing the same transactions sequentially in the PCN, termed *Cost Efficiency*. The former property guarantees that WISER does

not output a trivial solution and the solution maximizes the transaction throughput, while the latter guarantees that WISER is beneficial in terms of fees compared to the no-aggregation solution where each user executes its transactions sequentially and without the need for coordination on the PCN.

Lastly, due to the potential aggregation of transactions per user, the resulting flow may enhance the privacy of the protocol compared to the simple sequential execution per user. For instance, if the transaction aggregation selects two transactions for a user that partially cancel out (e.g., the user receives 5 coins and sends 3 coins), the user's channel to the hub will only be updated by the difference (e.g., 5-3=2 coins); thus, the hub only learns the aggregated flow and not the individual transactions. To capture this improvement on privacy guarantees, we define the *Privacy* property.

In particular, we define the notion of privacy we wish to achieve during the entire course of our protocol, i.e. both the flow computation and execution phases. We want to enforce, firstly, that each uninvolved user should only learn that they are not involved in the protocol. Secondly, each involved user (whether client or hub) should only learn the flow output on each of the incident channels to the hubs. We stress that the above two conditions imply value privacy: users will not know the flow (and thus the transaction amounts) along channels they are not a part of. Finally, we allow each involved user to learn the number of other involved users in the protocol [3].

Formally, WISER should achieve the following properties:

*Definition 2.3 (Computational Feasibility).* The transaction aggregation problem as per Definition 2.1 is fixed-parameter linear (FPL complexity class) [15], i.e., its running time is polynomial in the number of clients and exponential in the number of hubs.

*Definition 2.4 (Balance Security).* The change in user balances, after the execution of our protocol, must be the aggregated demand vector of some sublist $\mathcal{T}' \subset \mathcal{T}$ of transactions.

*Definition 2.5 (Optimality).* For a given list of transactions, the aggregation should select a maximal sublist in the sense of total demand fulfilled.

*Definition 2.6 (Cost Efficiency).* The total fees levied to users should be less than or equal to the amount that would have been levied had the transactions been processed sequentially.

---

[2]The decision of when the protocol triggers is left to the users. The core idea is to be able to accumulate many transactions per user may that be after some time interval or when a specific number of transactions per user is reached.

[3]Ideally, involved users should not learn any information about the set of other involved users in the protocol. However, our proposed flow execution protocol requires the number of involved users to be known in advance. We are working on future work to bound the number of involved users each involved user needs to know.

*Definition 2.7 (Privacy).* Our notion of privacy is based on the following indistinguishability game between an passive adversary $\mathcal{A}$ and a transaction aggregation and execution protocol $\Pi$ when run on the subgraph of users interested in participating in the protocol $G = (V, E)$:

- $\mathcal{A}$ chooses a subset of users $V^{\mathcal{A}} \subset V$ to corrupt, i.e., $\mathcal{A}$ gains access to the transcripts of every corrupted user $v \in V^{\mathcal{A}}$. $V^{\mathcal{A}}$ can consist solely of hubs, clients, or a mix of both. Let $G^{\mathcal{A}}$ be the subgraph created by taking the union of all corrupted nodes and their incident edges. Additionally, if a hub node is in $V^{\mathcal{A}}$, we add all the other hubs in the network and all their incident edges to $G^{\mathcal{A}}$.
- $\mathcal{A}$ chooses the following for $i \in \{0, 1\}$: $\mathcal{A}$ creates a list of transactions tuples $\mathcal{T}^i = \left[(x_j, s_j, r_j)\right]_{j=1}^{n_i}$ where the $j$th transaction tuple consists of a transaction amount $x_j$, the vertex $s_j$ that requested the transaction, and the vertex $r_j$ that should receive the transaction. $n_i$ is the number of transaction tuples in the $i$th list.
- For every node and channel in $G^{\mathcal{A}}$, the following condition needs to hold: the resulting flow returned by $\Pi$ restricted to the subgraph $G^{\mathcal{A}}$ when run on $G$ and when given transaction tuple lists $\mathcal{T}^0$ and $\mathcal{T}^1$ has to be the same. Additionally, the set of involved users as computed by $\Pi$ should be the same when given $\mathcal{T}^0$ or $\mathcal{T}^1$ as input.
- (Challenge phase.) We choose a random bit $b \in \{0, 1\}$ and run $\Pi$ with transaction tuple list $\mathcal{T}^b$.
- $\mathcal{A}$ gets the flow output sent to each corrupted user from $\Pi$.
- $\mathcal{A}$ outputs a bit $b'$. If $b' = b$, $\mathcal{A}$ wins the game.

We say a transaction aggregation and execution protocol $\Pi$ is $\epsilon$-private if $\mathcal{A}$ wins the above game with probability at most $\frac{1}{2} + \epsilon$, and private if it is $\epsilon$-private for some negligible $\epsilon$.

# 3 WISER ARCHITECTURE

In this section, we present WISER, a protocol that solves the transaction aggregation problem under our system model. In Section 3.1, we present the topology over which WISER is designed to operate. This restriction is necessary to design a computationally tractable solution to the otherwise NP-hard transaction aggregation problem (NP-hardness is proved in Section 4.1). Then we present WISER, which is illustrated in an abstract level in Figure 6.

WISER consists of a flow computation phase (Section 3.2) and an execution phase (Section 3.3). Nodes accumulate transactions within a fixed time interval and then secret share these transactions and their balances to all hub nodes, which in turn run a flow computation module. This module privately computes an optimal (in terms of maximum throughput) sublist of transactions to be executed and a flow that realizes these transactions. WISER then proceeds to the execution phase. Each node locally validates the flow and selected transactions received by the computation module and then proceeds to atomically executing the flow. That is, either all flows are executed or none. We summarize WISER in Algorithm 1 (Section 3.4).

## 3.1 PCN topology & Channel Factories

We consider the restricted topology presented in Section 2.2. That is, the set of nodes is composed by a small set of hub nodes $H$ and a set client nodes, such that each client is connected to exactly one hub. This assumption ensures that transactions, and general demand vectors, can only be routed by unique flows in our PCN. In such a PCN, transaction aggregation problem is fixed parameter tractable (see Section 4.1 for more details). Moreover, we assume that the hub nodes are interconnected with a channel factory.

Channel factories were first introduced as an intermediate construction between layer 1 on-chain transactions and layer 2 payment channel transactions[9]. These factories are set up between 3 or more individuals, and henceforth can be used to instantly open and close off-chain standard payment channels between any 2 of the involved parties.

Two parties set up a conventional payment channel by committing some amount of their individual funds into a 2-signature wallet considered as a shared account. In much the same way, $k$ parties may set up a channel factory among themselves by locking funds in a $k$-signature wallet. The internal working of a channel factory relies in fact on conventional payment channels, however these are not broadcast publicly nor included in the blockchain. Since the funds in the internal channels of the factory can be reallocated to new channels instantly, channel factories provide greater flexibility and scalability.

A conventional payment channel can execute arbitrarily many transactions between two parties provided the resultant balance lies within the channel's capacity, and a channel factory can achieve the same functionality for 3 or more parties. Formally, a channel factory between $r$ parties $v_1, \ldots v_r$ of current capacity $(c_1, c_2, \ldots c_r)$ (the total capacity is $C = c_1 + c_2 + \ldots c_r$ and is fixed) can route any demand vector $(d_1, \ldots d_r)$ provided $d_i \le c_i, \forall 1 \le i \le r$. Another equivalent description of channel factories is that the transaction aggregation problem on $r$ parties connected by a channel factory reduces to the bank clearing problem on $r$ banks.

There is a natural generalization of fees from payment channels to channel factories, where a node $v_i$ would charge a fee for a certain state transition from $\mathbf{c} = (c_1, c_2, \ldots c_r)$ to $\mathbf{c}' = (c_1', c_2', \ldots c_r')$ if $c_i' < c_i$ ie. if their balance decreases. Our assumption on the fee function as stated in Section 2.2 generalized to the following: the total forwarding fee must be a non-negative function $F(\mathbf{c}_0, \mathbf{c}_1)$ of the initial and final states $\mathbf{c}_0, \mathbf{c}_1$ of the channel factory. $F$ must also satisfy the triangle inequality: for any states $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2$, we have

$$F(\mathbf{c}_0, \mathbf{c}_2) \le F(\mathbf{c}_0, \mathbf{c}_1) + F(\mathbf{c}_1, \mathbf{c}_2).$$

The improvements of channel factories come at the cost of minor liveness assumptions, namely, that the internal channels in a factory can only be reorganized when all involved parties are online. Although this assumption is generally impractical, it is a reasonable one to make in the setting of WISER. Only the hubs, and not the clients, are required to set up a channel factory. Based on their role as payment service providers, it is fair to expect them to be online regularly.

Channel factories are not crucial to our protocol, but they provide an increase in throughput compared to solely using
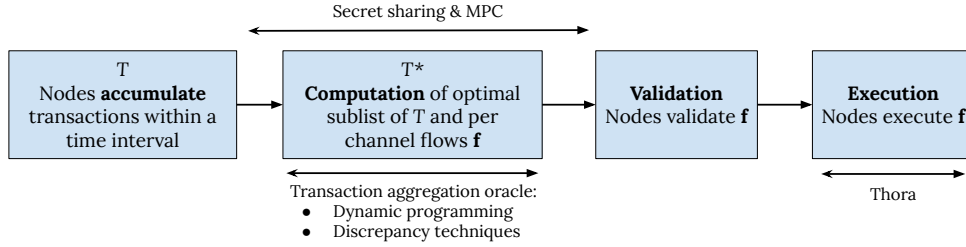
**Figure 6: Proposed implementation of the skeleton protocol for transaction aggregation. Secret sharing is used before the computation module to select the delegates and share the inputs, MPC is used throughout the computation module and until the module's output is delivered to the nodes. The tools used for the computation and (atomic) execution modules are noted below those boxes.**

payment channels. We recommend a channel factory between the hubs, as opposed to the hubs building various channels between each other for the following reason: suppose the hubs commit the same amount of capital to a set of payment channels between them to create a PCN $G_H$ between them. Any demand vector $\mathbf{d}$ that can feasibly be routed in $G_H$ can also be routed by the channel factory. On the other hand, there are always demands $\mathbf{d}$ that a given $G_H$ cannot route but the factory can. In this way, the throughput of the network is maximized for a given amount of capital per hub.

## 3.2   Flow computation phase

The flow computation phase begins when sufficiently many transactions have been collected by the users, or when a pre-specified time has elapsed. In this phase, the transaction aggregation problem is solved. In WISER we aim to preserve the privacy of transactions. For this reason, we employ secret sharing and secure multi-party computation to solve the optimization problem. The output of the flow computation phase is a list of transactions $\mathcal{T}^*$ and a flow $\mathbf{f}$ to be executed in the next phase. The two separate components of this phase are discussed below.

*Secret Sharing the Transaction Aggregation Input.* With secret sharing we refer to protocols that allow a user to distribute a secret among a group of participants, each of whom is allocated a share of the secret. The secret can be then reconstructed but only when a sufficient number of shares are combined (threshold), meaning that a sufficient number of participants cooperate. Furthermore, the individual shares (or a subset of the shares with cardinality less than the threshold) leak no information on the secret. When $n$ is the number of participants that get shares and $t$ is the reconstruction threshold, the protocol is called $(t, n)$−threshold secret sharing. WISER is agnostic to the selection of a specific secret sharing protocol, so any such scheme can be used such as Shamir's secret sharing [39].

In WISER, the list of transactions $\mathcal{T}$ as well as the channel capacities in either direction are secret shared by the users, in order to perform a multi-party computation of the transaction aggregation problem. In particular, we assume the protocol specifies a specific block header from the underlying blockchain that provides common randomness to the users. Using this

randomness, the users select $k$ delegates among the hubs, which will be the ones that will later execute the optimization oracle to provide a solution of the transaction aggregation problem to all users in a privacy-preserving manner. The users then $(k, k)$−threshold secret share to the $k$ delegates their inputs, specifically their outgoing transactions and their channel balances. For privacy reasons, we demand that all users send transactions – even if they are of zero value, when a user has only incoming transactions.

We stress that the $k$ delegates should be sampled from the set of hubs (as opposed to sampling from clients or a mixture of both) for the following reasons: firstly, as hubs are large business holders in the PCN, it becomes expensive, hence difficult, for an adversary to launch Sybil attacks to counterfeit them in order to control a disproportionate fraction of the delegates. Secondly, computing the solution to the transaction aggregation problem imposes resource and liveness constraints on the delegates: all delegates need to have sufficient computational power to perform the computation, and they need to be online throughout the computation process. It would therefore be more reasonable to assume that hubs, as large financial service providers in the PCN, indeed have the capacity to fulfil these requirements.

*Input Validation.* Once the transactions and constraints have been secret shared, the delegates should validate the well-posedness of the input. In particular, the delegates verify that no user submits outgoing transactions that would exceed the capacity of their channel with their hub, as assumed in Section 2.2.

*Solving the Transaction Aggregation Problem.* Let $O$ denote an optimization oracle that, when given a PCN graph $G$ and a list of transactions $\mathcal{T}$, returns a sublist $\mathcal{T}^*$ along with a resultant flow $\mathbf{f}$ that routes the aggregate of $\mathcal{T}^*$. $\mathcal{T}^*$ is optimal in the sense that the throughput $\sum_{\mathbf{t}_i \in \mathcal{T}^*} |\mathbf{t}_i|$ is maximized. We elaborate on our proposed optimization oracle below.

As stated in Section 3.1, our PCN consists of hubs, connected by a channel factory, and clients connected to one hub only. The assumption on $\mathcal{T}$ is that for each client $v$, their channel with the hub contains enough liquidity to route all transactions in $\mathcal{T}$

**Algorithm 1:** WISER – pseudocode for node $u$

**Local input:** $out(\mathcal{T}, u)$ a set of outgoing transactions from node $u$, where $\mathcal{T} := \cup_u out(\mathcal{T}, u)$ and the balances of adjacent channels $\{c(u, w), c(w, u) \mid u$ and $w$ share a channel$\}$.

**Local output:** Result of atomic execution of a flow that realizes a sublist $out(\mathcal{T}^*, u) \subseteq out(\mathcal{T}, u)$, such that $\mathcal{T}^* = \cup_u out(\mathcal{T}^*, u)$ is optimal in terms of total throughput $\sum_{\mathbf{t} \in \mathcal{T}} |\mathbf{t}|$.

**Macros:**

$\mathbf{f}(u)$: denotes the entries of a flow $\mathbf{f}$ referring to edges adjacent to $u$;

$in(\mathcal{T}, u), out(\mathcal{T}, u)$: the list of incoming, and respectively, outgoing transactions with respect to node $u$ in the transaction list $\mathcal{T}$;

```
                                    /* Flow computation phase                                              */
```
1  $secretSharingProtocol(out(\mathcal{T}, u), \{c(u, w), c(w, u) \mid u$ and $w$ share a channel$\})$;         `/* share input to hubs */`
2  **if** $u$ *is a hub (i.e., MPC delegate)* **then**
```
       /* Input validation                                                                       */
```
3     **for** $v \in V \setminus H$ **do**
4        **if** $\sum_{\mathbf{t} \in out(\mathcal{T}, v)} |\mathbf{t}| \geq c(v, h_v)$ **then** $\mathcal{T} \leftarrow \mathcal{T} \setminus out(\mathcal{T}, v)$;         `/* exclude txns violating outgoing capacity */`
5        **if** $\sum_{\mathbf{t} \in in(\mathcal{T}, v)} |\mathbf{t}| \geq c(h_v, v)$ **then** $\mathcal{T} \leftarrow \mathcal{T} \setminus in(\mathcal{T}, v)$;         `/* exclude txns violating incoming capacity */`
6     *optimizationOracle()*;
7     **return** $out(\mathcal{T}^*, v), \mathbf{f}^*(v)$ to each node $v$;
8  $out(\mathcal{T}^*, u), \mathbf{f}^*(u) \leftarrow MPC()$;         `/* u receives MPC output from hubs */`
```
   /* Execution phase                                                                             */
```
9  **if** $\exists \mathbf{t} \in out(\mathcal{T}^*, u)$ s.t. $\mathbf{t} \notin out(\mathcal{T}, u)$ **then** abort;         `/* validation of out(T*,u) */`
10 **if** $\exists \mathbf{t} \in in(\mathcal{T}^*, u) \cup out(\mathcal{T}^*, u)$ s.t. $\mathbf{t} \notin in(\mathcal{T}^*, v) \cup out(\mathcal{T}^*, v) \wedge \mathbf{t}$ is a transaction between $u$ and $v$ **then** abort;
11 **if** $\mathbf{f}^*(u)$ does not match the entries for $u$ in $\mathbf{f}^*(v)$ s.t. transactions between $u$ and $v$ appear in $\mathcal{T}^*$ **then** abort;
12 **if** $\sum\limits_{e=(v,u) \in E} \mathbf{f}(e) - \sum\limits_{e=(u,v) \in E} \mathbf{f}(e) \neq \mathbf{d}(v)$ **then** abort         `/* validation of d vs f */`
13 $Thora(\mathbf{f}^*(u))$;         `/* u executes f(u) by participating in Thora [1] */`

---

where $v$ is the sender. The same channel must also have sufficient liquidity in the opposite direction to route all transactions where $v$ is the recipient.

Under these assumptions, the transaction aggregation problem reduces to an integer program with $|\mathcal{T}|$ variables and $|H|$ constraints. Although solving such integer programs generally takes time exponential in $|\mathcal{T}|$, we employ a recent result by Eisenbrand and Weismantel [18] that solves integer programs in time linear in $|\mathcal{T}|$, albeit exponential in $|H|$, i.e., the algorithm is fixed-parameter linear [15]. Their algorithm uses discrepancy techniques (which address the question of rearranging a sequence of vectors to limit the norm of their partial sums) to facilitate dynamic programming. In Theorem 4.3 of Section 4.1 we prove that the proposed optimization oracle solves the transaction aggregation problem and compute its computational complexity.

## 3.3 Execution phase

The flow computation phase concludes with a solution to the transaction aggregation problem, albeit encoded as secret shares. The solution consists of the list of accepted transactions $\mathcal{T}^* \subseteq \mathcal{T}$, along with a resultant flow $\mathbf{f}$ on the network. An execution of this flow would be equivalent to a simultaneous execution of every transaction in $\mathcal{T}^*$. However, the solution must first be checked for validity, and then be executed atomically though the entire network. In the following, we discuss how the validation

of the solution will be performed as well as the execution of the accepted transactions through the network in an atomic manner. We further propose a specific structure for the interconnection of the hubs, known as payment channel factories [9].

*Flow Validation.* To check that a solution $(\mathcal{T}^*, \mathbf{f})$ is valid is to check that $\mathcal{T}^* \subseteq \mathcal{T}$, that aggregation of $\mathcal{T}^*$ yields a feasible demand vector $\mathbf{d} = \sum\limits_{\mathbf{t} \in \mathcal{T}^*} t$, and that the flow $\mathbf{f}$ routes $\mathbf{d}$.

A trusted third party could easily verify the above, but we require users to verify these conditions *locally*. We present a method for local verification, such that a solution is valid if and only if every user locally verifies the above conditions.

A user $v$ is supplied with a list $\mathcal{T}_v^* \subset \mathcal{T}^*$ such that a transaction $t \in \mathcal{T}_v^*$ whenever $v$ is the sender or the recipient. Representing transactions as demand vectors, $\mathcal{T}_v^* = \{\mathbf{t} \in \mathcal{T}^* : \mathbf{t}_v \neq 0\}$. Moreover, $v$ is supplied with $\mathbf{f}(e)$ for every edge $e$ adjacent to $v$ ie. $e = (u, v)$ or $e = (v, u)$.

The users can verify that the provided local data is consistent with that of other users: for every $t \in \mathcal{T}$, the sender-recipient pair $u, v$ can verify that $t \in \mathcal{T}_u^*$ if and only if $t \in \mathcal{T}_v^*$. Similarly, every pair of adjacent vertices $u, v$ with $(u, v) \in E$ can verify they have both been provided the same value of $\mathbf{f}(u, v)$.

$v$ verifies that every transaction in $\mathcal{T}_v^*$ also belongs in $\mathcal{T}_v$ and therefore in $\mathcal{T}$. $v$ computes their aggregate demand $\mathbf{d}(v)$ under $\mathcal{T}_v^*$. To verify that $\mathbf{f}$ routes $\mathcal{T}^*$ locally, $v$ simply checks if the

net flow of $\mathbf{f}$ through $v$ is equal to $\mathbf{d}(v)$:

$$\sum_{e=(v,u)\in E} \mathbf{f}(e) - \sum_{e=(u,v)\in E} \mathbf{f}(e) = \mathbf{d}(v).$$

*Atomic Execution.* After this verification, the users must execute $\mathbf{f}$ atomically, i.e., ensuring that our protocol executes the set of payments atomically and efficiently after performing transaction aggregation, is critical for security. An incomplete update of channels does not typically correspond to any meaningful financial activity like the execution of a sublist of aggregated transactions. The flow output from the previous flow computation phase might involve disconnected components of the network. To ensure atomicity of channel updates across all disjoint paths, we employ the solution proposed in Thora [1] that ensures any number of disjoint channels can be atomically updated within a constant time interval.

We first note that existing HTLC-based solutions like [25, 27, 33] can guarantee atomicity of the channel updates only when the subgraph induced by the set of channels form a path. Moreover, these solutions require users along the path to lock their payment amount for a time linear in the length of the path. For these reasons, in WISER we leverage Thora, which is solution that performs the multi-channel updates atomically in constant time [1]. Furthermore, Thora operates in any graph structure as it does not depend on the PCN topology or the connectivity among channels.

In more detail, let $\mathbf{f}$ denote the flow in which all channels need to be updated atomically. Let us denote the support of the flow by $E^* = \{e_i = (s_i, r_i)\}_{i=1}^m$, where $s_i$ denotes the sender and $r_i$ denotes the receiver in the $i$th channel. In Thora, each receiver first creates and signs a special transaction, $\text{tx}^{\text{ep}}$, which contains "dummy" outputs for every receiver $r_i$, $i \in [n]$ of value $\epsilon$ and requires a signature from each sender $s_i$. Each sender $s_i$ then creates a transaction to update the state of channel $e_i$ and another payment transaction for their corresponding receiver $r_i$. A payment transaction between $s_i$ and $r_i$ takes as input the output to $r_i$ in any $\text{tx}^{\text{ep}}$, and must be spent by a timeout period $T$. Thus, as long as any $\text{tx}^{\text{ep}}$ is posted by any receiver to the ledger before a given timeout period of $T$, receivers can use their corresponding output in the $\text{tx}^{\text{ep}}$ to ensure they get their payment and spend it. If any receiver does not spend their payment transaction by the timeout period $T$, all senders will get a refund with the refund amount being the amount of the payment. In this way, atomicity of channel updates is ensured.

In Thora, the timeout period $T$ is independent of $m$, the number of channels involved. In particular, if all users are honest, all channel updates and payments can be instantaneous, while security is guaranteed even in the presence of malicious adversaries. We note that Thora guarantees value privacy with respect to channel balances, but requires all users involved to know the public keys of the other involved users. These public keys can nevertheless be made pseudonymous (see Section 5.3 for more details).

## 3.4 Algorithm description

We consolidate the description of WISER in Algorithm 1. We present WISER from the point of view of a node $u$; be it client or hub. The input to the protocol is the list of transactions $out(\mathcal{T}, u)$ originating from $u$ and the balances of channels adjacent to $u$. The output of WISER is the result of the atomic execution of a flow $\mathbf{f}^*$ that realizes a sublist $\mathcal{T}^*$ of $\mathcal{T}$, which is optimal in terms of total throughput $\sum_{\mathbf{t}\in\mathcal{T}} |t|$.

Lines 1–8 consist of the flow computation phase. In line 1 node $u$ secret shares its local input to the hubs (MPC delegates). That is, the outgoing transactions (or a single zero transaction if there is no outgoing transaction) and the capacities of all adjacent channels. Then, the hub nodes run lines 2–8. They first check if the incoming and outgoing transactions submitted are feasible with the given channel capacities (lines 3–5) and exclude user inputs otherwise. Subsequently, they run the optimization oracle (line 6), and return the output to all nodes (line 7). With the given output (line 8), node $u$ proceeds to the execution phase. It first validates the output in lines 9–12 and then participates in Thora with its local input $f^*(u)$ (line 13).

## 4 FORMAL ANALYSIS

In this section, we prove that WISER satisfies the protocol goals. We first discuss the complexity of the transaction aggregation problem, and show that the presented algorithm is fixed-parameter linear in the number of hubs. Later, we demonstrate that WISER satisfies balance security, optimality and cost efficiency. Lastly, we prove security with our indistinguishability game as per Definition 2.7.

### 4.1 Complexity of the Transaction Aggregation Problem

In section 3.2, we referred to an optimization oracle $O$ to abstract out the algorithmic challenge of computing an optimal sublist $\mathcal{T}^* \subset \mathcal{T}$. Here, we will first show that the problem is NP-hard in general via reduction from the well-known subset sum problem. Then we will restrict attention to our specific graph and show that the problem is fixed-parameter linear. That is, we show that the computational complexity of the optimization oracle is linear in the number of transactions $|\mathcal{T}|$ but exponential in $m$, the number of hubs.

THEOREM 4.1. *The Transaction Aggregation Problem is NP-hard.*

PROOF. In fact, the problem is NP-hard even for a graph on two vertices and no edges. Consider an instance of the Subset Sum problem, well-known to be NP-hard: given a positive integer $A$ and a set $\{a_1, \ldots, a_k\}$ of positive integers, find a subset of them that sum to $A$.

Let $G = (V, E)$ with $V = \{v_1, v_2\}$, $E = \emptyset$. Let $\mathcal{T} = \{(a_1, -a_1), (a_2, -a_2), \ldots (a_k, -a_k), (-A, A)\}$. Finding *any* nonempty subset of $\mathcal{T}$, let alone one optimal in terms of throughput, which can feasibly be aggregated, yields a solution of the Subset Sum problem. □

*Remark:* Transaction aggregation coincides with BCP when the graph has only two vertices. Thus, the NP hardness (and in fact inapproximability) follows from the NP-hardness and inapproximability of BCP with two participants[21].

We now proceed to calculating the computational complexity of the optimization oracle (Section 3.2). Let us first state the

relevant theorem from [18] before proceeding to apply it to our setting.

THEOREM 4.2 (THEOREM 8 IN [18]). *An integer program of the form[4]*

$$\max \mathbf{w}^{\mathrm{T}}\mathbf{x} \quad \textit{such that}$$

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}, 0 \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^k.$$

*for* $\mathbf{c} \in \mathbb{Z}^k$, $\mathbf{A} \in \mathbb{Z}^{h \times k}$, $\mathbf{b} \in \mathbb{Z}^h$, $\mathbf{u} \in \mathbb{Z}^k_+$, *can be solved in time* $O(k(h\Delta)^{h^2})$, *provided all entries of* $\mathbf{A}$ *are bounded in absolute value by* $\Delta$.

THEOREM 4.3. *The Transaction Aggregation problem of* WISER *can be solved in time* $O(k(h\Delta)^{h^2})$, *where* $\Delta$ *is an upper bound on the demand of every transaction, k is the number of transactions in* $\mathcal{T}$ *and h is the number of hubs.*

PROOF. It is sufficient to reformulate the transaction aggregation problem as one satisfying the hypothesis of Theorem 4.2 above.

Consider an arbitrary ordering of the transactions in $\mathcal{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_k\}$. For a vector $x \in \{0, 1\}^k$, let $\mathcal{T}(x) = \{\mathbf{t}_i : x_i = 1\}$. We need to construct a matrix $\mathbf{A}$ and vectors $\mathbf{w}, \mathbf{b}$ such that finding the optimal sublist $\mathcal{T}^* \subset \mathcal{T}$ is equivalent to finding a vector $\mathbf{x}$ maximizing $\mathbf{w}^{\mathrm{T}}\mathbf{x}$ subject to $\mathbf{A}\mathbf{x} = \mathbf{b}$ and $0 \leq \mathbf{x} \leq \mathbf{u}$, where $\mathbf{u}$ is the all ones vector.

The main idea behind the construction of $\mathbf{A}$ is that, by our assumption on client-hub channels having sufficient capacity to route all transactions from $\mathcal{T}$, we need only consider feasibility of the demand vector of a sublist $\mathcal{T}'$ with respect to the channel factory between the hubs.

So let us order the hubs as $h_1, h_2, \ldots, h_m$ and define $\mathbf{b} = (b_1, b_2, \ldots, b_m)$ as: $b_i$ is the current balance of $h_i$ in the channel factory. Now, for every transaction $\mathbf{t}_i \in \mathcal{T}$ of demand $|\mathbf{t}_i|$, we define $\mathbf{A}_i$, the $i$th row of $\mathbf{A}$ coordinate-wise as: $\mathbf{A}_{i,j} = |\mathbf{t}_i|$ if the sender of $\mathbf{t}_i$ is connected to the hub $h_j$. $\mathbf{A}_{i,j} = -|\mathbf{t}_i|$ if the recipient of $\mathbf{t}_i$ is connected to hub $h_j$, and 0 otherwise. We define $\mathbf{w}$ as $\mathbf{w}_i = |\mathbf{t}_i|$.

The vector $\mathbf{A}\mathbf{x}$ represents the requisite flow in the channel factory for routing the aggregate of $\mathcal{T}(\mathbf{x})$. Due to our assumption on the capacity of client-hub channels, this is sufficient for checking for the feasibility of the aggregate. Since $\mathbf{b}$ represents the current state of the channel factory, the aggregate of $\mathcal{T}(\mathbf{x})$ is feasible if and only if $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. We conclude the proof by observing that maximizing the objective $\max \mathbf{c}^{\mathrm{T}}\mathbf{x}$ is equivalent to maximizing the throughput $\sum |\mathbf{t}_i|$ of the aggregate. □

Theorem 4.3 immediately implies the following.

COROLLARY 4.4. WISER *satisfies computational feasibility as in Definition 2.3.*

---

[4]The integer program in [18] is presented in the so-called standard form of $\mathbf{A}\mathbf{x} = \mathbf{b}$, whereas we use the inequality form $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ here for simplicity of exposition. Any integer program can be reformulated from inequality form to standard form by the introduction of slack variables, which doubles the number of variables. This is inconsequential in our case as our dependence on the number of variables is linear.

## 4.2 Balance Security

Here, we show that the users of WISER do not lose any money through their participation in the protocol.

THEOREM 4.5. WISER *satisfies balance security as in Definition 2.4.*

PROOF. The flow execution phase of WISER is the only phase that involves monetary flow, hence requiring atomicity. In WISER, flow execution is done using Thora, which guarantees atomicity of channel updates assuming rational adversaries (see [1] for more details). Thus, WISER also inherits atomicity against rational adversaries. As a result, either the atomic execution of a flow that routes our optimal list $\mathcal{T}^*$ succeeds, implies that the change in user balances corresponds to the same sublist $\mathcal{T}^* \subset \mathcal{T}$. Or, the atomic execution failed, in which case the user balances are unchanged, corresponding to the empty sublist $\emptyset \subset \mathcal{T}$. □

## 4.3 Optimality & Cost Efficiency

Here, we first show that the selected set transaction is maximal in terms of throughput. Next, we show that the users of WISER may benefit from their participation in the system in terms of fees.

THEOREM 4.6. WISER *satisfies optimality as in Definition 2.5.*

PROOF. This follows immediately from the objective function of the transaction aggregation problem (Definition 2.1) and the correctness of the optimization oracle (Theorem 4.3). □

THEOREM 4.7. WISER *satisfies cost efficiency as in Definition 2.6.*

PROOF. We wish to prove that total fee levied to users should be no greater than the fees that would have been levied had the transactions been processed sequentially. Although WISER routes an optimal sublist $\mathcal{T}^*$, we prove the above for any list $\mathcal{T}$ of transactions that can feasibly be aggregated. The main idea of this proof is that forwarding fees satisfy the triangle inequality, and that transactions can only be routed along unique paths (Section 2.2).

Suppose $\mathcal{T} = [\mathbf{t}_1, \ldots \mathbf{t}_k]$ and the flow $\mathbf{f}$ routes $\mathcal{T}$. Note that a sequential execution of transactions may not be possible although the atomic execution of the aggregate (this is in fact one of the benefits of transaction aggregation), in which cost efficiency holds trivially. Suppose that a sequential execution of transactions is indeed possible, and without loss of generality assume this order is $\mathbf{t}_1, \mathbf{t}_2 \ldots \mathbf{t}_k$.

Consider any channel or channel factory $C$ in our PCN, and let $a_0$ be its initial state. Suppose the atomic execution of $\mathbf{f}$ leaves $C$ in the final state $a_f$, and this incurs fee $F = F(a_0, a_f)$. Next, suppose the sequential execution of transactions takes $C$ along states $a_0, a_1, \ldots a_k = a_f$, so that the routing of $\mathbf{t}_i$ involves a state transition from $a_{i-1}$ to $a_i$ and costs $F(a_{i-1}, a_i)$. Since the fee function satisfies the triangle inequality,

$$F(a_0, a_f) = F(a_0, a_k) \leq \sum_{i=1}^{k} F(a_{i-1}, a_i).$$

Observing that the total fees incurred is simply the sum of forwarding fees per channel, this establishes cost efficiency. □

## 4.4 Privacy

Our transaction aggregation solution as well as the execution protocol satisfies our privacy notion, i.e.,

THEOREM 4.8. WISER *satisfies the privacy notion from Definition 2.7 (assuming the users running the MPC protocol satisfy the trust assumptions required by the underlying MPC protocol).*

PROOF. Consider the indistinguishability game as defined in Definition 2.7. Let us assume during the challenge phase of the game there exists a trusted third party (TTP) that computes a flow $\mathbf{f}_b$ from the challenge transaction tuple list $\mathcal{T}^b$. Suppose additionally that the computed flow satisfies all the constraints as detailed in Definition 2.7. The TTP subsequently lets each user in $V$ know how much they need to send or receive on each of their incident edges.

Each corrupted user learns, from the TTP, only the amount they need to send or receive on each of their incident channels. Since the flow restricted on $G^{\mathcal{A}}$ has to be the same when the protocol is run on either $\mathcal{T}^0$ or $\mathcal{T}^1$, the joint view of the adversarial nodes when the protocol is run on $\mathcal{T}^0$ or $\mathcal{T}^1$ is identical (and thus trivially indistinguishable).

Now we can replace the above assumption of a TTP with an MPC protocol that computes the same functionality with the same privacy guarantees. If the parties running the MPC protocol satisfy the trust assumption of the underlying protocol, privacy is preserved. A natural choice would be to use a computationally (rather than information theoretically) secure MPC protocol (so we have privacy unless all the users are corrupted), and (for efficiency reasons) pick a small random subset of the users to run the MPC protocol after receiving the secret-shared inputs from all the participating users. This way privacy is preserved unless the entire randomly picked subset of users has been corrupted by the adversary.

The execution phase of our protocol uses the Thora protocol as a black box, and Thora guarantees that payment values along any channel that needs to be updated are not known to anyone except the channel owners. This again ensures that involved users only know the payments and hence flow on their incident channels. Thora leaks the set of involved users, but they can be made pseudonymous using ephemeral keys (see Section 5.3). Thus, the only information involved users gain is the number of all other involved users, which is allowed in our privacy definition.

Since we restrict the set of involved users to be exactly the same when the protocol computes the flow given transaction tuples $\mathcal{T}^0$ and $\mathcal{T}^1$, the view of an adversarial node when the execution phase of the protocol is run on $\mathcal{T}^0$ or $\mathcal{T}^1$ is again indistinguishable, hence the execution phase is private. □

## 5 DISCUSSION & EXTENSIONS

In this section, we first discuss the WISER architecture and its various components. Then, we elaborate on our privacy definition and potential extensions. Finally, we revisit the assumptions of this work and in particular the fees requested for routing by the hubs.

## 5.1 WISER modularity

WISER implements the skeleton protocol described in Section 2.4. However, the specific components of the skeleton protocol are modular, in the sense that they can be replaced with other protocols that maintain the same guarantees.

As mentioned previously, there are various suitable MPC frameworks we can use. As an active area of research, we expect improvements on the side of secret sharing and MPC protocols, which would directly apply to WISER. The delegates conducting the MPC can also be chosen from any set of parties that have sufficient computing power and strong identities. The MPC may even be entirely replaced by a semi-trusted third party if the application does not require privacy of transactions and channel balances.

The rest of the components are also replaceable. Executing the flow derived from aggregating transactions can be implemented by any protocol, like Thora, that provides atomicity of channel updates.

In WISER we assume the hubs have created an on-chain channel factory. Although this construction benefits the protocol in terms of throughput, lower liquidity requirements as well as runtime, this structure is by no means restrictive or necessary for our protocol execution. Hubs that are already connected via traditional payment channels in an arbitrary graph structure can implement transaction aggregation without paying for constructing channel factories.

Most importantly, the optimization oracle is also replaceable. In Section 5.2 below, we demonstrate the equivalence of BCP, the Bank Clearing Problem and transaction aggregation on channel factories. Due to the modularity of WISER, we can use any of the algorithms developed for BCP as our optimization oracles. We also discuss applicable algorithms below.

## 5.2 Connection to Netting

As observed in Section 3.1, the use of channel factories between $h$ hubs with capacities $\mathbf{c} = (c_1, c_2, \ldots c_r)$ allows for routing of any demand vector $\mathbf{d} = (d_1, d_2, \ldots d_r)$ so long as $\mathbf{d} \leq \mathbf{c}$. Observe that this feasibility constraint is identical to that of the Bank Clearing Problem when $h$ participants have capital (or cover money) $\mathbf{c}$. Since both transaction aggregation and BCP aim to maximize throughput or clearing volume, the problem of transaction aggregation on a channel factory is equivalent to BCP.

On one hand, this means our fixed-parameter linear algorithm from Section 4 also shows that BCP is fixed-parameter linear in the parameter $(h\Delta)$. This linear dependence on the number of transactions is not only best possible (since simply reading a list of $k$ transactions requires $O(k)$ time), but also improves upon the previous known polynomial dependence on $k$.

Although the algorithm we present in Section 4.1 enjoys the best known asymptotic complexity, other, simpler algorithms such as the pseudopolynomial time dynamic programming

solution of [21] may be faster in certain parameter regimes. Moreover, although BCP has been shown to be inapproximable unless P = NP, [38] propose a fast heuristic for approximately solving it, which may also be employed here to achieve even faster implementations at the cost of optimality.

## 5.3 Privacy

Here we discuss two privacy issues regarding our protocol.

*Revealing the full set of involved users.* As briefly mentioned in Definition 2.7, we use Thora to execute the computed flow atomically, and this leaks the full set of involved users. This is because every receiver in Thora has to create a special transaction $tx^{ep}$ that contains outputs for every other receiver and signed with a signature from each sender. Not only is it possible, but it is also good practice, to use fresh ephemeral public keys (also known in the literature as receive addresses) when receiving tokens in UTXO-based cryptocurrencies. If all users employ fresh receive addresses each time they participate in our protocol, the only information leaked would be the number of involved users. We consider this a non-issue, as the set of involved users is not revealed, but only the size of this set.

*Hiding transaction requests from hubs.* In WISER, users secret share their transaction requests and the transaction aggregation problem is solved using MPC. Thus, the flow computation stage conceals transactional data.

In the execution phase, the hubs see the aggregated flow output on all their incident channels, but they cannot confidently determine sender-recipient pairs or demands since the number of transactions is not known to them. This gives users an additional potential privacy benefit when using our protocol as compared to when the hubs sequentially execute transaction requests.

## 5.4 Optimizing for Fees

The algorithm presented in Section 4.1 seeks a feasible sublist of transactions that maximizes throughput without considering the fees incurred in the execution of flow. As a result, we can only guarantee the weak notion of cost efficiency as given in Definition 2.6: that the fee is no greater than the case without transaction aggregation at all ie. had the transactions been processed sequentially.

WISER can in fact achieve greater cost efficiency by modifying the optimization oracle. In fact, only the objective function, and not the algorithm itself, must be modified to a sum of throughput and total fees incurred, with a certain trade-off factor relating the importance of these two objectives. By leaving the constraints of the integer program unchanged, this extension enjoys the same runtime as given by Theorem 4.3.

## 6 RELATED WORK

Payment channel networks have received much attention recently as a promising way to increase the limited transaction throughput of blockchains. Payment channels were originally introduced by Spilman [41], but the first bidirectional constructions followed later with the Bitcoin Lightning Network [33]

and the Duplex Micropayment Channels [13]. There is currently a flurry of payment channel proposals each optimizing a different aspect [2, 3, 6, 7, 12, 16, 17, 28, 29]; see [14] for a recent survey.

*Routing on PCNs.* The original proposals for payment channels faced many practical challenges. A major challenge was how to design an efficient routing algorithm for a PCN: as the current balances on the network are not known, identifying a viable short path from sender to receiver proved to be difficult. Several routing protocols for PCNs have been developed to provide efficient solutions to this problem. Flare [34] and SilentWhispers [24] both employ highly connected nodes to route the payments in order to improve the scalability of the routing algorithm. SpeedyMurmurs [37] and VOUTE [36] improve the process by leveraging a routing approach called prefix embeddings. Flash [47] uses a modified max-flow algorithm to find the optimal path, while Perun [16] introduces virtual channels to avoid routing through intermediaries. On a different front, other routing discovery algorithms focus on maintaining the anonymity and privacy during the route discovery [5, 32] or the transaction execution [44].

However, all of these works focus on a different problem, that of efficient (and often privacy-preserving) routing in PCNs with arbitrary graph structure. We, on the other hand, simplify the network structure and examine how the users of such a graph topology can benefit from other functionalities of transaction aggregation such as canceling out and "transferwise", in order to increase throughput and decrease the intermediaries' fees.

*Rebalancing PCNs.* The previously mentioned routing approaches generally ignore the issue of balance depletion; channels that transfer coins mainly in one direction will soon get depleted and must top-up the balance on-chain. The rebalancing problem was identified and addressed in Revive [22], which was the first work to propose the use of rebalancing strategies. Many interesting follow-up works spawned afterwards: Pickhardt et al. [31] improved the balances of a PCN as a sequence of rebalancing operations of the channel funds, while Avarikioti et al. presented Hide & Seek, [8] a privacy-preserving decentralized rebalancing protocol that improved upon Revive. Also the Bitcoin Lightning Network comes with rebalancing plugins, e.g., c-lightning[5] and lnd[6]. The Spider Network [40] splits payments into smaller units and routes them over multiple paths using waterfilling, aiming to efficiently route transactions while maintaining balanced channels. The Merchant [45] utilises adaptive fee strategies to incentivize the balanced use of payment channels, and [23] uses estimated payment demands along channels to plan the amount of funds to inject into a channel during channel creation. There also exist several interesting approaches to save fees in payment channel networks, e.g., virtual channels [16], and studies on the security implications of such fee mechanisms [43].

Transaction aggregation is intrinsically related to the question of rebalancing in the sense that multiple flows can execute

---

[5]https://github.com/lightningd/plugins/tree/master/rebalance
[6]https://github.com/bitromortac/lndmanage

a given aggregate of transactions, and an appropriate choice of flow inevitably results in minimal channel depletion. In other words, rebalancing can be interpreted as transaction aggregation of an empty list of transactions. However, none of these works leverage transaction aggregation to increase throughput in PCNs and hence do not enable routing transactions even in a disconnected graph topology with the newly introduced "transferwise" functionality.

*Atomic multi-hop transaction execution.* Traditional payment channel networks like the Bitcoin Lightning network achieve the atomic execution of multi-hop payments using Hash Timelock Contracts or HTLCs [13, 33]. however, HTLC suffer from sepcific attacks such as the wormhole attack [26]. In a recent line of work, several proposals exist that improve upon the use of simple HTLCs to guarantee the atomic execution of transactions, typically targeting efficiency and security in an extended model [1, 4, 42]. These works are complementary to ours because our proposed solution requires the use of a protocol that can execute transactions in an atomic manner in general topologies where the graph can even be disconnected. Therefore, any efficiency improvements on the atomic transaction execution in generic topologies implies immediately an efficiency improvement on WISER.

*Netting.* Our work is closely related to netting systems in financial markets, specifically netting for inter-bank settlements, which involve using a central entity (typically a central bank) to settle all liabilities of involved financial institutions [20]. The netting problem for banks, also known as the Bank Clearing Problem, is NP-complete and inapproximable. However, heuristic-based algorithms that yield approximate solutions in practice have been developed in [21] and [38]. On the side of exact algorithms, a pseudopolynomial time exact algorithm for BCP was presented in [21]. This exact algorithm is based on dynamic programming, but processes transactions in the same order as input. Our algorithm, also based on dynamic programming, improves upon theirs by exploiting discrepancy techniques to reorder the transactions and hence reduce time and memory requirements to linear.

Besides the algorithmic connection, netting is also related to our work in terms of privacy and decentralization. Although inter-bank netting traditionally relies on central banks as mediators, there are multiple reasons to seek decentralized alternatives. Central banks are trusted to preserve the confidentiality of transactional data, as well as perform the netting correctly, which incurs greater liability for all involved parties. As observed in [10], finding trusted mediators for cross-border multi-currency transfer is also challenging. To this end, there are some works which aim to implement decentralised netting systems [10, 11]. In particular, the work of Cao et al. provides a decentralized and optimal netting solution that also guarantees the privacy of payment amounts using smart contracts and zero knowledge proofs on the blockchain [10]. Our work, in contrast, aims to address this problem for payments on payment channel networks, and thus all our proposed solutions in WISER seek to reduce the usage of the underlying blockchain for arbitration.

## 7 CONCLUSION

In this work, we formulated the transaction aggregation problem and presented a protocol solving it. Transaction aggregation can potentially increase throughput and even realize payments that were infeasible when executed sequentially. We presented a skeleton protocol for a hub-based business model that abstracts the components required to (i) privately compute the flows through each channel maximizing the transaction throughput, and (ii) execute those flows atomically. We then presented WISER, an implementation of our skeleton protocol: In WISER we employed channel factories as the intra-hub channel structure. We further proposed a fixed-parameter linear algorithm to solve the transaction aggregation problem, which we execute via an MPC to maintain privacy of transactions and channel balances. Finally, WISER utilized Thora to execute the flow atomically across the PCN. With WISER we (a) enhanced the liquidity of the PCN, (b) effectively reducing the transaction fees to the hubs, while (c) maintaining the privacy of the transaction's and channels' data.

We regard our work as a first step toward more general protocols. Specifically, an interesting future work is to design a computationally tractable protocol for more general topologies than the hubs/clients topology we use. Algorithmic breakthroughs in the specific topology we consider would also benefit centralized or decentralized netting protocols. Also, we can improve the privacy guarantees by anonymizing the involved users or hiding transaction requests from hubs. Moreover, it is orthogonal to our work to compute a fee mechanism charged by the hubs. Note that including fees in the optimization component is straightforward and does not affect the protocol's complexity, although its effect on throughput is not understood.

## 8 ACKNOWLEDGEMENTS

# REFERENCES

[1] Lukas Aumayr, Kasra Abbaszadeh, and Matteo Maffei. 2022. Thora: Atomic And Privacy-Preserving Multi-Channel Updates. *IACR Cryptol. ePrint Arch.* (2022), 317. https://eprint.iacr.org/2022/317

[2] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2020. Bitcoin-Compatible Virtual Channels. *IACR Cryptology ePrint Archive, Report 2020/554* (2020).

[3] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2020. Generalized Bitcoin-Compatible Channels. *IACR Cryptology ePrint Archive, Report 2020/476* (2020).

[4] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2021. Blitz: Secure Multi-Hop Payments Without Two-Phase Commits. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 4043–4060. https://www.usenix.org/conference/usenixsecurity21/presentation/aumayr

[5] Zeta Avarikioti, Mahsa Bastankhah, Mohammad Ali Maddah-Ali, Krzysztof Pietrzak, Jakub Svoboda, and Michelle Yeo. 2021. Route Discovery in Private Payment Channel Networks. *IACR Cryptol. ePrint Arch.* (2021), 1539. https://eprint.iacr.org/2021/1539

[6] Zeta Avarikioti, Eleftherios Kokoris Kogias, Roger Wattenhofer, and Dionysis Zindros. 2021. Brick: Asynchronous Incentive-Compatible Payment Channels. In *FC*. https://fc21.ifca.ai/papers/168.pdf

[7] Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. 2020. Cerberus channels: Incentivizing watchtowers for bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 346–366.

[8] Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. 2022. HIDE & SEEK: Privacy-Preserving Rebalancing on Payment Channel Networks. In *International Conference on Financial Cryptography and Data Security (FC)*. https://arxiv.org/abs/2110.08848

[9] Conrad Burchert, Christian Decker, and Roger Wattenhofer. 2017. Scalable Funding of Bitcoin Micropayment Channel Networks - Regular Submission. In *SSS*.

[10] Shengjiao Cao, Yuan Yuan, Angelo De Caro, Karthik Nandakumar, Kaoutar Elkhiyaoui, and Yanyan Hu. 2020. Decentralized Privacy-Preserving Netting Protocol on Blockchain for Payment Systems. In *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12059)*, Joseph Bonneau and Nadia Heninger (Eds.). Springer, 137–155. https://doi.org/10.1007/978-3-030-51280-4_9

[11] James Chapman, Rodney Garratt, Scott Hendry, Andrew McCormack, and Wade McMahon. 2017. Project Jasper: Are Distributed Wholesale Payment Systems Feasible Yet?

[12] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. 2018. eltoo: A Simple Layer2 Protocol for Bitcoin. https://blockstream.com/eltoo.pdf.

[13] Christian Decker and Roger Wattenhofer. 2015. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*. Springer, 3–18.

[14] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. 2021. Survey on Blockchain Networking: Context, State-of-the-Art, Challenges. In *Proc. ACM Computing Surveys (CSUR)*.

[15] Rodney G Downey and Michael Ralph Fellows. 2012. *Parameterized complexity*. Springer Science & Business Media.

[16] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2017. Perun: Virtual payment hubs over cryptocurrencies. In *IEEE Symposium on Security and Privacy*. 327–344.

[17] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. 2019. Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks. In *Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security*. ACM, 801–815.

[18] Friedrich Eisenbrand and Robert Weismantel. 2017. Proximity results and faster algorithms for Integer Programming using the Steinitz Lemma. *CoRR* abs/1707.00481 (2017). arXiv:1707.00481 http://arxiv.org/abs/1707.00481

[19] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2017. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO*. 10.1007/978-3-319-63688-7_10

[20] Rodney Garratt and Peter Zimmerman. 2020. Centralized netting in financial networks. *Journal of Banking & Finance* 112 (2020), 105270. https://doi.org/10.1016/j.jbankfin.2017.12.008 Challenges to global financial stability: interconnections,credit risk, business cycle and the role of market participants.

[21] Michael M. Güntzer, Dieter Jungnickel, and Matthias Leclerc. 1998. Efficient algorithms for the clearing of interbank payments. *Eur. J. Oper. Res.* 106, 1 (1998), 212–219. https://doi.org/10.1016/S0377-2217(97)00265-8

[22] Rami Khalil and Arthur Gervais. 2017. Revive: Rebalancing Off-Blockchain Payment Networks. In *CCS*. 10.1145/3133956.3134033

[23] Peng Li, Toshiaki Miyazaki, and Wanlei Zhou. 2020. Secure Balance Planning of Off-blockchain Payment Channel Networks. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 1728–1737. https://doi.org/10.1109/INFOCOM41043.2020.9155375

[24] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2017. SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks. In *NDSS*. 10.14722/ndss.2017.23448

[25] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2019. Concurrency and Privacy with Payment-Channel Networks. *CoRR* abs/1911.09148 (2019). arXiv:1911.09148 http://arxiv.org/abs/1911.09148

[26] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *NDSS*. 10.14722/ndss.2019.23330

[27] Subhra Mazumdar and Sushmita Ruj. 2022. CryptoMaze: Privacy-Preserving Splitting of Off-Chain Payments. *IACR Cryptol. ePrint Arch.* (2022), 123. https://eprint.iacr.org/2022/123

[28] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. 2019. Pisa: Arbitration outsourcing for state channels. In *AFT*. 10.1145/3318041.3355461

[29] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. 2019. Sprites and State Channels: Payment Networks that Go Faster Than Lightning. In *International Conference on Financial Cryptography and Data Security*. 508–526.

[30] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.

[31] Rene Pickhardt and Mariusz Nowostawski. 2020. Imbalance measure and proactive channel rebalancing algorithm for the Lightning Network. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2020, Toronto, ON, Canada, May 2-6, 2020*. IEEE, 1–5. https://doi.org/10.1109/ICBC48266.2020.9169456

[32] Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, and Michelle Yeo. 2021. LightPIR: Privacy-Preserving Route Discovery for Payment Channel Networks. In *IFIP Networking*. 10.23919/IFIPNetworking52078.2021.9472205

[33] Joseph Poon and Thaddeus Dryja. 2015. The Bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf.

[34] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. 2016. Flare: An Approach to Routing in Lightning Network. shorturl.at/adrHP.

[35] Mohammed El Qorchi, Samuel Munzele Maimbo, and John F. Wilson. 2003. *Informal Funds Transfer Systems: An Analysis of the Informal Hawala System*. International Monetary Fund, USA. https://doi.org/10.5089/9781589062269.084

[36] Stefanie Roos, Martin Beck, and Thorsten Strufe. 2016. VOUTE-Virtual Overlays Using Tree Embeddings. *arXiv: 1601.06119* (2016). arXiv:1601.06119 http://arxiv.org/abs/1601.06119

[37] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. 2018. Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions. In *NDSS*. 10.14722/ndss.2018.23254

[38] Yakov M. Shafransky and Alexander A. Doudkin. 2006. An optimization algorithm for the clearing of interbank payments. *Eur. J. Oper. Res.* 171, 3 (2006), 743–749. https://doi.org/10.1016/j.ejor.2004.09.003

[39] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (nov 1979), 612–613. https://doi.org/10.1145/359168.359176

[40] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. 2020. High throughput cryptocurrency routing in payment channel networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 777–796.

[41] Jeremy Spilman. 2013. Anti DoS for tx replacement. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html. Accessed: 2020-11-22.

[42] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. 2021. A$^2$L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 1834–1851. https://doi.org/10.1109/SP40001.2021.00111

[43] Saar Tochner, Aviv Zohar, and Stefan Schmid. 2020. Route Hijacking and DoS in Off-Chain Networks. In *Proc. ACM Conference on Advances in Financial Technologies (AFT)*.

[44] Somanath Tripathy and Susil Kumar Mohanty. 2020. MAPPCN: Multi-hop Anonymous and Privacy-Preserving Payment Channel Network. In *International Conference on Financial Cryptography and Data Security*.

Springer, 481–495.

[45] Yuup van Engelshoven and Stefanie Roos. 2021. The Merchant: Avoiding Payment Channel Depletion through Incentives. In *IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2021, Online Event, August 23-26, 2021*. IEEE, 59–68. https://doi.org/10.1109/DAPPS52256.2021.00012

[46] By Matthew Wall. 2015. Money may make the world go round, but at what cost? https://www.bbc.com/news/business-31639262. [Published on 23 March 2015].

[47] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. 2019. Flash: Efficient Dynamic Routing for Offchain Networks. In *International Conference on Emerging Networking Experiments And Technologies*. 370–381. https://doi.org/10.1145/3359989.3365411