X-Transfer: Enabling and Optimizing Cross-PCN Transactions

Lukas Aumayr[1,5], Zeta Avarikioti[2,5], Iosif Salem[3,6], Stefan Schmid[3], and Michelle Yeo[4]

[1] University of Edinburgh
[2] TU Wien
[3] TU Berlin
[4] National University of Singapore
[5] Common Prefix
[6] ZeroPoint Technologies

Abstract. Blockchain interoperability solutions allow users to hold and transfer assets among different chains, and in so doing reap the benefits of each chain. To fully reap the benefits of multi-chain financial operations, it is paramount to support interoperability and cross-chain transactions also on Layer-2 networks, in particular payment channel networks (PCNs). Nevertheless, existing works on Layer-2 interoperability solutions still involve on-chain events, which limits their scalability and throughput. In this work, we present X-Transfer, the first secure, scalable, and fully off-chain protocol that allows payments across different PCNs. We formalize and prove the security of X-Transfer against rational adversaries with a game theoretic analysis. In order to boost efficiency and scalability, X-Transfer also performs transaction aggregation to increase channel liquidity and transaction throughput while simultaneously minimizing payment routing fees. Our empirical evaluation of X-Transfer shows that X-Transfer achieves at least twice as much throughput compared to the baseline of no transaction aggregation, confirming X-Transfer's efficiency.

Keywords: Payment channel networks · Layer-2 · interoperability · optimization · transaction aggregation · cryptocurrencies

1 Introduction

Payment channel networks (PCNs) [47,43,23,11,22,40] are a promising solution to mitigate the limited transaction throughput of blockchains [21]. Two parties that wish to transact with each other can open a payment channel between themselves by depositing funds into a "common account" on the blockchain only to be used in this channel. Whenever the parties transact with each other, they update the distribution of funds in the channel by decreasing the funds of the sender and increasing the funds of the receiver by the payment amount. To close a payment channel, parties can publish the last agreed distribution of funds

on-chain either cooperatively or unilaterally. As such, with just a constant number of blockchain transactions, any two parties can make an unlimited number of costless transactions between themselves. A network of users and payment channels between pairs of users constitute a payment channel network (PCN), which also allows for multi-hop routing of payments between users that are not directly connected through intermediary nodes [43]. Examples of PCNs are Bitcoin's Lightning Network [43] and Ethereum's Raiden [3].

An important open problem in PCNs is to design secure and scalable cross-PCN payment solutions in order to fully unlock their interoperability potential, complementing existing cross-chain solutions on the blockchain itself [1,34,44]. Existing solutions for cross-blockchain payments rely on *bridges* [1,17,34,37,44,45,51], which condition the occurrence of some transaction on a destination blockchain given the occurrence of a specific event on a source blockchain. These solutions, however, still involve on-chain events and thus do not fully leverage the scalability that fully off-chain solutions can provide. The main challenge in adapting these solutions to the *purely off-chain* setting is the absence of global events off-chain, as off-chain state updates only occur among pairs of users in payment channels. This makes conditioning the occurrences of off-chain state updates an extremely difficult exercise in coordination and incentive-alignment, and remains an open challenge.

Our contributions. In this paper, we present X-Transfer, the first secure and scalable cross-PCN transaction protocol that relies purely on off-chain events. X-Transfer comprises of an aggregation followed by an execution phase. For the aggregation phase, we assume a specific "star" topology among all PCNs whereby users are connected to a single "hub node" that forms the center of the star. The reason for this realistic assumption (more details in Section 3.1) is twofold: first, it is necessary in order to ensure that solving the transaction aggregation problem is feasible (we show it is polynomial in the number of transactions and exponential in the number of PCNs), and second, we use the specific assumptions about hub nodes (that there is at least one PCN that contains wallets of all hub nodes and channels between them) in order to execute transactions securely in the second execution phase of X-Transfer. During the aggregation phase, multiple transactions across PCNs are aggregated such that the resulting aggregated transactions occur simultaneously rather than sequentially. In this way, transactions could "cancel" each other out which reduces transaction fees and increases liquidity. Furthermore, our optimization problem in the transaction aggregation phase involves both maximizing the total volume of transactions selected for aggregation while minimizing the volume of cross-PCN transactions which involves heftier cross-PCN transaction fees. In doing so, we ensure the largest amount of throughput possible across all PCNs while ensuring that fees are kept minimal.

In the second execution phase, the aggregated transactions are executed. To ensure balance security (i.e., that the balance of involved users across PCNs does not change apart from what they should send or receive) of involved users, we first simultaneously execute the transactions only among the hubs. This effectively ex-

ecutes all cross-PCN transactions. We then show that assuming all involved parties are rational, using incentive alignment arguments and strategically-chosen execution time parameters, we can ensure that the execution of all cross-PCN aggregated transactions is the off-chain variant of the "global event" necessary to induce updates of all subsequent aggregated transactions in all PCNs. We employ Thora [6], an existing single-PCN atomic channel update protocol, to ensure all hub-to-hub channels and well as channels within each PCN can be atomically updated.

We summarize our contributions as follows:

- We present the building blocks of X-Transfer, the first purely off-chain cross-PCN transaction protocol which also performs aggregation/optimization, in Section 3. We also include formal definitions of the desiderata of X-Transfer (which includes security, privacy, feasibility and optimality definitions) as well as model assumptions.
- We detail both the aggregation and execution phases of X-Transfer in Section 4, including the design principles behind the protocol.
- We analytically show that X-Transfer achieves the aforementioned desiderata of security, privacy, feasibility and near-optimality in Section 5.
- We perform an empirical evaluation of X-Transfer's performance in Appendix I under the metrics of transaction throughput and computational overhead. We show that X-Transfer achieves at least twice as much throughput compared to the baseline of no aggregation with little additional overhead.

1.1 Related Work

PCNs. Payment channels [11,39,22,13,8,40,27,9,26] emerged as a promising technology to improve blockchain transaction throughput. Originally introduced by Spilman [47], the first bidirectional channels followed with the Lightning Network [43] and Duplex Micropayment Channels [23]. See [25] for a recent survey.

Transaction aggregation. Transaction aggregation in PCNs is the problem of finding an optimal subset of transactions that maximizes the total satisfiable transaction volume with a minimal number and volume of actual transactions carried out. Typically, this is done by finding transactions that "cancel" each other out. In the context of PCNs, the problem was first proposed and studied in [48] but only for the single PCN setting. To make the computational problem tractable, [48] proposed a "star topology" where clients connect directly to several hubs arranged in a clique. Our work extends the problem considered in [48] to the multiple PCN setting where transactions across several PCNs can cancel each other out. Although we also adopt a similar star topology as in [48] in each PCN, a novel and key focus of our work is finding the optimal set and volume of cross-PCN monetary flows, as well as ensuring that the resulting aggregated transactions can be executed across PCNs atomically. We also note that both the centralized [31,46] and decentralized [19,18] variants of the netting problem (interbank liabilities are aggregated and settled) are also similar to the problem

studied in our work. In particular, the work of [18] uses smart contracts on the blockchain. In our work, though, we focus on off-chain aggregation of cross-PCN transactions, which avoids the usage of costly blockchain transactions.

Atomic cross-chain payments. In the single PCN setting, there are several tools [43,38,10,6] that govern the atomic updates of channel states. Our work, however, addresses atomic channel updates across multiple PCNs. Jia et al. [35] propose using a trusted third party (TTP) to open and close a payment channel between two users in different blockchains. Guo et al. [32] present a protocol for cross-PCN channels using expensive cryptography. Zhang and Qian [52] propose a hub-based cross-PCN structure, but their protocol relies heavily on deposits to prevent rational users from deviating, such as by failing to execute a transfer and stealing funds. These deposits serve as an incentive for hubs to follow the protocol. In contrast, X-Transfer does not rely on a TTP and is lightweight, requiring neither expensive cryptography nor substantial deposits – only small deposits for paying fees to the hubs for their services. Alba [45] is a decentralized bridge [1,17,34,37,44,45,51] that can condition executions on the destination blockchain on off-chain events. However, Alba's executions still occur on (and thus involve) the destination blockchain, whereas our protocol enforces fully off-chain conditional executions.

2 Background and Notation

Payment channel networks (PCNs). Several payment channels opened on the same blockchain form a payment channel network (PCN). A node on a PCN represents a channel party, and an edge represents a channel among the two nodes/parties it connects. Refer to Appendix A for more details about routing in PCNs and payment fees.

Thora [6]. Thora is a single-PCN channel state update protocol that ensures that any number of (possibly disjoint) channels in a PCN can be updated atomically. The key idea behind Thora's atomic updates is the preparation and signing of a specific "enable-payment" transaction that allows receiving parties to enforce payments from their corresponding sending parties in their payment channels. Indeed, after setting up Thora, every recipient has this transaction along with its necessary signatures. If the corresponding sender refuses to update, the recipient can post the enable-payment transaction. As long as one such enable-payment transaction appears on the blockchain (which should only happen during a dispute), every other involved user can enforce their promised payments. In the optimistic case, nothing goes on-chain.

Thora achieves two properties: (i) *atomicity* ensures that either all channels update or revert, and malicious users cannot deviate from this outcome except by forfeiting their money to the honest parties, and (ii) *strong value privacy*, which ensures that the update value of any channel out of the set of to-be-updated channels is only known to the two channel users in the optimistic case. We describe further details of Thora in Appendix C.

Transaction Aggregation and Wiser [48]. For transaction aggregation, we will build upon Wiser, a single-PCN private aggregation and execution protocol. Wiser consists of 2 phases: transaction aggregation and execution. In the aggregation phase, transactions are chosen to maximize the total demand in the network. To ensure privacy, parties secret share their transactions and channel balances, and then the optimization problem is solved using multiparty computation (MPC) among a selected number of delegates. Thora is used to execute all transactions atomically during the transaction execution phase.

Notation and Transaction Model. For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, \ldots, n\}$. Let $\epsilon > 0$ be the smallest amount of cryptocurrency funds that can be sent in the blockchain. We use H to denote a globally available cryptographic hash function, e.g., SHA256. We assume all underlying blockchains implement an Unspent Transaction Output (UTXO) model [5,41]. In this model, transactions are mappings between inputs and outputs. We can write a transaction as $t_i = [o_j^1, o_j^2, o_k^1, \ldots] \mapsto [o_i^1, o_i^2, \ldots]$ where i, j, k are transaction identifiers. We further denote a UTXO output $o = (x|C)$ as having a monetary value of x coins, which are only spendable if the Boolean expression C evaluates to True. In our work we are mainly interested in 3 types of literals in C: (1) timelocks, where we simply use a constant to denote that it evaluates to True after the specified amount of time has elapsed. (2) hashlock, where we use $H(\mathfrak{s})$ to denote that it evaluates to True when one provides the preimage $\mathfrak{s}$ of $H(\mathfrak{s})$. (3) signature locks, where we use σ_{u_i} to denote that it evaluates to True if the signature of user u_i is provided. When multisignatures between a specific set of k parties $u_1, \ldots, u_k$ are required, we specify them as $\sigma_{u_1,\ldots,u_k}$. We use # to denote UTXO inputs that are irrelevant to a given transaction design, e.g., $t_i = \# \mapsto [o_i^1, o_i^2, \ldots]$.

3 Model

3.1 System Model and Assumptions

We assume we have k PCNs, each supporting a different blockchain, whose users wish to interact with each other. We restrict each PCN to a single hub node and several client nodes, and we denote the hub node in the ith PCN as h_i for $i \in [k]$. The nodes in each PCN are arranged in a star topology with the hub as the star center. We adopt this topological assumption to make the computation of the cross-chain transaction aggregation problem tractable. We stress, however, that this assumption corresponds to the high degree of centralization observed in the Lightning Network in reality [42,50], is also adopted in previous work [48], and is shown to be stable [12,14]. We further assume each hub node has wallets in all other PCNs, and there exists a PCN that contains all channels between hubs. We make the reasonable assumption that the capacities of the inter-hub channels are a lot larger than the capacity of channels between clients and hubs, as cross-PCN transfers could potentially be a lot larger than transfers within a PCN. Although we do not specify the requisite capacity of these inter-hub channels, we assume for the rest of the paper that the capacity of these channels

is large enough to handle all cross-PCN transfers. Finally, we assume that hub nodes only participate in routing transactions in the protocol, and do not send or receive transactions. Let $G_i = (V_i, E_i)$ represent the ith PCN.

The input to the problem is a set of transactions $\mathcal{T} := \{(x_i, s_i, r_i)\}_{i=1}^{n}$ where x_i represents the size of the ith transaction in the list, and s_i, r_i represent the sender and recipient of the ith transaction. Note that s_i and r_i can be in different PCNs. A payment x_i can be sent from $s_i \in G_1$ to $r_i \in G_2$ in the following way: s_i sends x_i to h_1 along the channel (s_i, h_1). Assuming G_2 is the PCN that contains the channel between h_1 and h_2, h_1 sends x_i to h_2 along the channel (h_1, h_2). Finally, h_2 sends x_i to r_i along the channel (h_2, r_i). As these between-hub channels effectively shift payments from one PCN to another, we call these between-hub channels *cross-PCN channels*.

We assume payments going through cross-PCN channels are significantly more expensive compared to payments routed within a PCN. The main reason behind this assumption is that the hubs have to lock funds in several PCNs in order to provide this service to users, which incurs a high opportunity cost. Additionally, if users were to go with a traditional swap, the user would have to find a trusted service provider, which would incur high fees, or use atomic swaps on the blockchain, which also incur a larger cost in terms of gas fees to run the smart contracts [34,24]. Formally, suppose we have a transaction (x_i, s_i, r_i) with s_i, r_i in different PCNs. Denote the payment path the transaction takes from s_i to r_i as $\pi = (e_1, e_2, \ldots, e_m)$ where $e_j \in \pi$ can either be channels within a single PCN or cross-PCN channels. Let us further suppose the total number of cross-PCN channels in π is $m' < m$. Then, the fee incurred for this transaction would be $\sum_{j=1}^{m-m'} f(x_i) + m'\alpha$ for some affine function f and large positive constant α.

Assumptions. We further make some usual assumptions concerning cryptographic primitives, the underlying blockchains, the communication model, and the adversary. In particular, we assume the existence of secure communication channels between users. We also assume all underlying blockchains are censorship-resistant, and also satisfy persistence and liveness as defined in [29]. In addition, we assume a synchronous network model, i.e., there is a known network delay that bounds the time needed for any user to receive any incoming message. We assume that all PCNs operate using the same underlying tokens. Further, we assume all parties (hubs and clients) are rational, i.e., they may deviate from the honest protocol execution if they may increase their profit. Finally, we assume that hubs and clients are not colluding. We discuss collusion and potential mitigation in Section 6.

3.2 Desired Properties

In the following, we define the desiderata of our protocol.

Firstly, X-Transfer should maintain the safety of channel funds for users that follow the protocol, encompassed by the following property.

Definition 1 (Balance security). *No honest party loses more than a negligible amount of funds*[7] *as a result of participating in* X-Transfer.

Moreover, our protocol should ensure that users incur minimal fees.

Definition 2 (Fee minimization). *The solution of* X-Transfer *should execute the list of transactions $\mathcal{T}$ such that the total fees are minimized.*

The computational complexity of the problem depends on the aggregation, which is a hard optimization problem. Accordingly, we postulate computational efficiency in the sense that a solution must be fixed-parameter tractable, i.e., polynomial in the number of transactions:

Definition 3 (Computational feasibility). *The aggregation problem is fixed-parameter linear, i.e., polynomial in the number of input transactions n and exponential in the number of PCNs k.*

This is reasonable since transactions likely involve only a few PCNs.

Privacy is a key aspect of PCN protocols, as payment channels inherently protect users' balances and transactions. We adapt the privacy definition from [48] to multiple PCNs, providing an informal description below and leaving the formal definition to Appendix D. Uninvolved users learn only that they do not participate. Involved parties know the flow output on their incident channels and their direct counterparties. Receiving parties also learn all other recipients within the same PCN. Hub nodes additionally learn the set of involved users across all participating PCNs.

4 The X-Transfer Protocol

This section outlines the design principles and details of X-Transfer. We first provide an overview, followed by a detailed description of its phases, covering both the optimization solution and transaction execution. An example implementation with three PCNs is presented in Appendix F.

4.1 Protocol Overview

X-Transfer proceeds in two phases: an aggregation phase and an execution phase. In the aggregation phase, our protocol privately computes an aggregation of the input transactions such that the resulting aggregation optimizes transaction throughput while minimizing fees. Then, each user u receives a monetary flow $\mathbf{f}(e)$ representing either inflow or outflow of funds to or from u for all channels incident to u. The user u needs to check whether the computed flow is

[7] In the aggregation phase of X-Transfer, we require that hub clusters that are disconnected from other hub clusters are connected by a payment channel that sends negligible amount of funds $\epsilon > 0$ between them. Total connectivity of hubs is required to ensure atomicity of execution in the execution phase of X-Transfer, and this will be the only portion of X-Transfer where additional funds of ϵ are transferred.

correct (that is, u will not lose money if the flow is executed). The actual execution of the transactions happens after all users have verified the correctness of the flow computation. To ensure that the computed flow is executed atomically both within and across PCNs, our protocol employs Thora to execute transactions within each PCN as well as cross-PCN transactions, with carefully chosen Thora time parameters to connect all these executions together. Figure 1 depicts a high-level overview of both phases of our protocol.

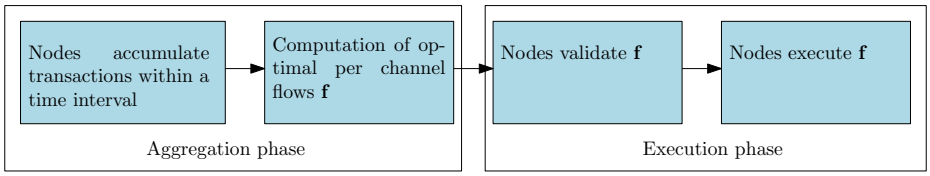

Fig. 1. X-Transfer phases for cross-PCN transaction aggregation

4.2 Aggregation Phase

Secret sharing inputs. The aggregation phase begins after sufficiently many transactions have been accumulated. To preserve the privacy of the input transactions and channel balances, our protocol requires each party to secret-share their transactions and their balance information along their incident channels. These shares are given to a group of delegates that will validate the correctness of the inputs, i.e., check that no party submits a transaction that exceeds their balance in their channel with their hub, and compute the solution of the transaction aggregation optimization problem using multiparty computation (MPC) [49]. We stress that our protocol is agnostic to the type of secret sharing scheme, MPC protocol, as well as how the delegates are chosen, so long as the group of delegates satisfies the trust assumptions of the underlying MPC protocol.

Optimization problem. The optimization problem is to maximize the volume of successful transactions (throughput) while minimizing the flow amounts (or fees since they are linear in the flows). Recall that in every PCN, clients connect directly to a hub, and the hub-to-hub balances are assumed to be high enough to accommodate for all input transactions. Thus, we can independently find the subset $\mathcal{T}^* \subseteq \mathcal{T}$ that maximizes transaction volume and is feasible and then find the flows routing the transactions in $\mathcal{T}^*$. We will solve the first problem by reducing it to the optimization problem solved in Wiser [48] (in FPL[8] time complexity) and the second one with a polynomial-time greedy algorithm, which we prove to use $k-1$ links (where k is the number of hubs), as required by X-Transfer's execution phase. Thus, the total time complexity is in FPL.

[8] FPL (Fixed Parameter Linear) is a complexity class where a decision problem has time complexity $O(f(k) \cdot |x|)$, with x and k as inputs. The complexity is *linear in* $|x|$ but can be *arbitrary* (often exponential) in k. FPL is a subset of FPT (Fixed Parameter Tractable), which includes problems that are computationally hard but remain tractable when exponential complexity is confined to a specific parameter k.

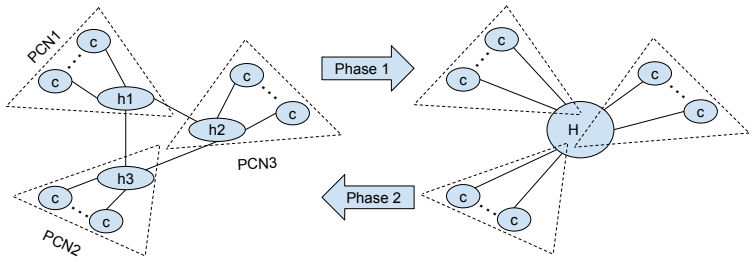

Fig. 2. Sample graph structures for the optimization problem. Clients connect directly to the hub of each PCN and every hub is connected to all other hubs. We contract the hub clique to one node $\mathcal{H}$ for phase 1 (finding the subset $\mathcal{T}^* \subseteq \mathcal{T}$ that maximizes throughput) and find the flows realizing $\mathcal{T}^*$ in the original graph in phase 2.

In more detail, to find the subset $\mathcal{T}^* \subseteq \mathcal{T}$ that maximizes the transaction volume and respects channel capacities, we replace the hub-to-hub network with a single node $\mathcal{H}$ (see Figure 2 for details). The resulting topology is a star graph in which all clients in all PCNs connect to $\mathcal{H}$. This abstraction takes advantage of the much larger capacity of hub-to-hub channels compared to client-to-hub channels, thus relegating the problem of optimally realising $\mathcal{T}$ to client-to-hub channels. We model our optimization problem as an integer linear program (ILP) as follows:

$\max \sum_{i=1}^{n} |t_i| x_i$, subject to

$$\sum_{i:\, t_i=(u,*,*)} |t_i| x_i - \sum_{j:\, t_j=(*,u,*)} |t_j| x_j \leq c(u, \mathcal{H}), \text{ for every client } u \quad (1)$$

such that $x_i \in \{0, 1\}$ is a decision variable for choosing whether a transaction $t_i = (s, r, x)$ is included in the output solution or not, $|t_i| = x$ is the transaction amount, $c(u, v)$ is the capacity of channel (u, v), and $*$ denotes any existing value.

The ILP states that we aim at finding an assignment of values to the x_i variables, such that the volume is maximized, while for every client-to-hub channel the sum of outgoing flow minus the incoming flow is bounded by the channel capacity. This is the exact same transaction aggregation problem of Wiser in one PCN, which is shown to be NP-hard [48, Theorem 1]. It can be solved in $O(n(k\Delta)^{k^2})$ time through the work of [28, Theorem 8], where $\Delta = \max_{i \in [n]} |t_i|$, i.e., the maximum transaction amount (or a bound on that), n is the number of transactions, and k is the number of hubs. That is, the complexity is linear in the number of transactions and exponential in the maximum transaction amount and the number of hubs. Therefore, it belongs to the FPL complexity class.

Given the optimal transaction subset $\mathcal{T}^*$ from the ILP, we now have to compute the cheapest flows realizing it in the actual network. Since the client-to-hub flows are already computed in the first phase, we need to compute the flows on exactly $k - 1$ links connecting the hubs and realizing the computed flows. We formally describe a polynomial-time greedy algorithm (Algorithm 2) to solve this problem in Appendix E.1. Informally, the algorithm works by aggregating the total inflow and outflow of each hub node from the solution specified by $\mathcal{T}^*$ and using this to define supply (resp. demand) hubs as hubs that need to

send (resp. receive) funds to (resp. from) other hubs. The supply and demand hubs are sorted in descending order, and for each demand amount specified by a demand hub, we add hub-to-hub links with as many supply hubs as needed to fulfill the demand. The supply hubs are re-sorted and the procedure is repeated. The following lemma (proof in Appendix G.1) shows that our greedy algorithm outputs at most $k-1$ hub-to-hub links.

Lemma 1. *Let $|E_{greedy}|$ the number hub-to-hub of links created by our greedy algorithm and $|\mathcal{H}_s|$, $|\mathcal{H}_d|$ be the number of supply and demand hubs, respectively. Then,* $\max\{|\mathcal{H}_s|, |\mathcal{H}_d|\} \leq |E_{greedy}| \leq k-1$.

Finally, note that if the algorithm outputs less than $k-1$ hub-to-hub links, we simply add links of size $\epsilon > 0$ to connect any disconnected hub components.

Restricting the topology among hubs to a path. At this point, we note that the above greedy algorithm connects all hubs in a DAG topology, which we denote as G. In X-Transfer, we restrict the topology of the hub-to-hub channels to a path. The main reason for this restriction is that we use some secret from receiving hubs (i.e., hubs that only receive funds from other hubs) to link all payments together during the execution phase to ensure atomicity. However, the setting with more than one receiving hub opens a vulnerability whereby the first receiving hub that reveals their secret can get their funds stolen. We describe these vulnerabilities in detail in Appendix E.2. To convert the DAG topology G among the hubs into a path topology P, we employ another polynomial-time algorithm (Algorithm 3 in Appendix E.2) to create a path topology P from G while maintaining the invariant of *balance conservation* of the vertices, i.e., the difference between the sum of all incoming and outgoing edges is the same for all vertices in G and in P. We leave the details of the procedure to Appendix E.2.

Computing execution time parameters for execution phase. In addition to the path graph representing the flow of funds between hubs, the aggregation phase of X-Transfer also outputs some time parameters which determines the sequence of both cross-PCN and within-PCN fund transfers in the execution phase of X-Transfer. The reason why these time parameters are computed during the aggregation and not the execution phase is mainly to preserve privacy. We present an informal description of the procedure as well as give an intuition of correctness and leave the formal description and details to Appendix E.3.

Informally, let us denote the client-to-hub flows in a PCN as outputted by the ILP as the *net flow* of the PCN. We observe that we can classify PCNs into three categories: PCNs that have positive, negative, or zero net flow. Positive (resp. negative) net flow PCNs have hubs that have positive (resp. negative) *inflow* of funds from their clients, and neutral net flow PCNs have zero out or inflow. Now assuming that the hub-to-hub flows have already been executed, we note that hubs with positive or neutral net flow will have incentive to execute the within-PCN transfers to receive funds from the process[9]. These PCNs are deemed

[9] The argument in the case for why neutral net flow PCNs are incentivized to execute the within-PCN transfers even though their balance does not change is mainly due

"safe" and will have smaller execution time parameters. We now state two crucial observations that underlie the correctness of our procedure: (1) there always exists a recipient in an "unsafe" PCN (let us call it G_i) with a corresponding sender in a safe PCN. G_i can then be added to the safe set and given a larger execution time parameter, which allows enough time for senders in safe PCNs to propagate necessary information to their recipients. As we use Thora to update channels within PCNs (more details in Section 4.3), the recipient in G_i can use this information to enforce all payments in the case where h_i refuses to execute the channel updates in G_i. (2) This process always terminates with all PCNs labeled safe. We show this with the proof of Lemma 3 in Appendix E.3.

4.3 Execution Phase

The execution phase starts once each user u verifies the aggregation output, following a flow validation process similar to [48].

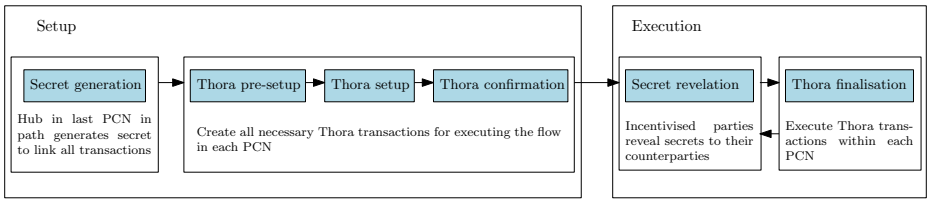

Fig. 3. X-Transfer Execution phase stages

Recall that the output of the aggregation phase is a path topology which defines a depth-ordering $h_1, \ldots, h_k$ among the hubs (and corresponding PCNs), as well as time parameters for each PCN which determines their execution order. The main design challenge of the execution protocol is to ensure the atomicity of transactions: as long as one transaction is executed in our protocol, all other (involved PCNs as well as cross-PCN) transactions should also be executed. We stress that this challenge stems from the fact that there does not exist any off-the-shelf protocol that guarantees atomicity for cross-PCN transactions. Indeed, while Thora ensures the atomicity of transactions within a single PCN, there are no atomicity guarantees for transactions going out of the PCN.

Strawman protocol. A simple strawman protocol to address the lack of atomicity between PCNs would be to use 1 Thora for each PCN to execute the transactions inside the PCN (i.e., transactions between clients and hubs). Then, since we assume that (1) hubs have wallets in each PCN, (2) there is at least one PCN (say G_j) in which all hubs have channels with each other, and (3) the hubs are connected in a path topology, we can set up hashed timelock contracts (HTLCs, see Appendix A for details) between the hubs in G_j to handle the cross-PCN transfers. Nevertheless, this strawman protocol leaves a glaring issue unresolved: hubs can steal funds from other hubs in an attack similar to

to the fact that they gain some funds as compensation when X-Transfer gets executed successfully. More details in Section 6.

the wormhole attack as described by [38]. In fact, unlike the classic wormhole attack where the victim simply loses out on the payment fees, the attack is more devastating in our setting as it actually impairs the balance security of our protocol. We illustrate this with a simple example with four PCNs G_1, G_2, G_3 and G_4 with corresponding hubs h_1, h_2, h_3, h_4. We assume an aggregated flow of x_1 coins going from h_1 to h_2, x_2 coins going from h_2 to h_3, and x_3 coins going from h_3 to h_4. If we use HTLCs locked with a secret generated by h_4 to transfer funds between the hubs, h_4 could collude with h_2 and skip revealing the secret to h_3 by directly revealing the secret to h_2, resulting in a loss of x_2 funds for h_3 and hence balance security of our protocol. Although this would mean that h_4 will lose out on getting x_3 coins, h_2 would profit from not sending x_2 coins to h_3. Thus, if x_2 is sufficiently larger than x_3, the total profit of h_4 and h_2 would be larger under the attack. Hence, it is imperative that X-Transfer imposes *an atomic method for updating channels between hubs.*

Atomic channel updates between hubs. To ensure that channel updates between hubs are atomic, that is, either no hub can update their channels or all hubs have the means to do so, X-Transfer uses another Thora protocol to update the channels among the hubs. In doing so, the channel updates are guaranteed to be atomic from the Thora protocol. Nevertheless, at this point there are still a few problems left unanswered, chief of which is *a method that links all of these payments together.*

Secret generation. A simple way to link all the transactions (both transactions within all PCNs and cross-PCN transactions) would be to use a common secret as an additional hashlock on the transactions, making all transactions unspendable unless the secret is revealed. The choice of which parties should generate the secret is important, as not all parties have an incentive to reveal the secret at a later stage so as to force the transactions to go through. For instance, consider a simple example with three PCNs G_1, G_2 and G_3 with an aggregated flow of x_1 coins going hub $h_1 \in G_1$ to hub $h_2 \in G_2$, and x_2 coins going from h_2 to h_3 (see Figure 6 for a depiction of this setting). Because h_1 has a net cash outflow, using h_1 to generate the secret would lead to h_1 only revealing the secret to the users in G_1 but not h_2, thus gaining a profit of x. To counter this, *we only allow secret generation to be done by the hub in the last PCN as defined by the path topology returned from the aggregation phase of* X-Transfer *(i.e., h_k).* While common secrets linking transactions play a crucial role in ensuring balance security during the execution of the protocol, there can still be violations of balance security if the setup or execution order is bad. Thus, a related challenge is *defining a setup and execution order such that balance security is preserved.*

Setup and execution order. We first highlight a problematic scenario with a bad setup order and then show how X-Transfer avoids this. Consider again the simple case of 3 PCNs illustrated in Figure 6. If the Thora updating the channel between the hubs is set up before the individual PCN Thoras, as h_3 knows the secret, h_3 can enforce the channel updates and steal x_2 coins from h_2 even before the other Thoras are set up. Thus, X-Transfer ensures that all within PCN Thoras are set up first before the Thora updating all inter-hub channels is set

up. In this way, there is no incentive for the secret-generating hub to reveal the secret at any point during setup. This is because the secret generating hub, as the hub in the last PCN in the within-hub transaction path and thus is receiving funds from other PCNs, is in a positive balance *after executing the transactions in their PCN and without executing the inter-hub transactions.*

During the Thora setup, each PCN uses the Thora time parameter as output from the aggregation phase, with the additional Thora between the hubs assigned a time parameter T_0 such that T_0 is smaller than all other time parameters as returned from the aggregation phase. This ensures that the Thora responsible for updating cross-PCN transfers executes first. Once these cross-PCN transfers have been executed, the net flow positions of all other hubs together with the assigned time parameters ensure that there is at least one user (hub or client) in each PCN that is incentivized to enforce the Thora execution in each PCN (stated and proved in Lemma 3 and Lemma 5).

X-Transfer execution phase details. The actual execution phase of X-Transfer can be broken down into 2 stages: setup and execution. Figure 3 depicts all intermediate steps in both stages of the execution phase, and a formal description of the execution stage of X-Transfer is detailed in Algorithm 1. The setup stage begins with h_k sampling a secret $\mathfrak{s}$ as well as computing a hash of the secret $H(\mathfrak{s})$. $H(\mathfrak{s})$ is then broadcasted to all the hubs and will be used as an additional hashlock on all transactions.

Once all hubs have verified that they have received $H(\mathfrak{s})$, all individual PCN Thoras are set up to handle the updates of the transactions inside each PCN (Lines 3 and 4 in Algorithm 1). The time parameter that is used as input to the Thora setup phase for each PCN G_i is the time parameter T_i corresponding to G_i returned from the aggregation phase. Here we stress that this process can be done in parallel. Once all individual PCN Thoras are set up and the correctness of the setup stage is verified, an additional Thora is set up in G_j to handle the updates of all cross-PCN transactions. The time parameter used for this Thora is $T_0 < T_i \forall i$. Note that the Thora pre-setup, setup, and confirmation stages in our protocol follow almost exactly as described in [6], with the exception that some transactions are locked with all the extra hashlocks generated during secret generation. We detail these changes in Algorithms 5, 6 and 7 in Appendix E.4.

After the setup, the protocol moves on to the actual execution, which begins with h_k revealing the secret $\mathfrak{s}$ to the other hubs. The hubs verify that $\mathfrak{s}$ is the preimage of $H(\mathfrak{s})$. Thereafter, the hubs can use $\mathfrak{s}$ to enforce the inter-hub channel updates, which execute the cross-PCN transactions. Following that, the Thoras handling the updates of the channels within each PCN can be executed. Note that all channel updates follow exactly as per the Thora protocol.

5 Analysis

In this section, we show that X-Transfer satisfies all desired properties outlined in Section 3.2. We will provide informal arguments and description of

Protocol 1: Execution Phase of X-Transfer

Data: PCNs $G_1, \ldots, G_k$, times $T_0, T_1, ..., T_k$, blockchain delay parameter Δ
Result: Broadcasted secret $\mathfrak{s}$, Updated incident channels

/* Secret Generation */
h_k generates a random secret $\mathfrak{s}$
h_k broadcasts $H(\mathfrak{s})$ to all other hubs

/* Thora setup */
Each PCN G_i runs Algorithm 5 with inputs $G_i, \Delta, H(\mathfrak{s})$
Each PCN G_i runs Algorithm 6 with inputs $G_i, T_i, \Delta, H(\mathfrak{s})$
Thora confirmation on G_i follows as per Thora protocol (see Algorithm 7)
After confirming that each Thora is set up correctly, hubs $h_1, \ldots h_k$ set up another Thora in PCN G_j with the involved channels being their inter-hub channels and time parameter T_0

/* Thora Execution */
h_k reveals $\mathfrak{s}$ to hubs, enabling the update of the inter-hub channels as per the Thora protocol
Doing so reveals $\mathfrak{s}$ to each hub, which enables the updates of each involved channel in each PCN as per the Thora protocol.

techniques used to show how these properties are satisfied and leave all formal statements and proofs to Appendix H.

Informally, balance security is preserved in the aggregation phase due to the definition of the optimization problem, the correctness of the underlying solver, as well balance conservation in the conversion from a DAG to path hub topology. We show balance security is preserved in the execution phase of X-Transfer by first defining an underlying extensive form game induced by the execution phase of X-Transfer. Thereafter, we show that following the protocol as stipulated by X-Transfer is a strict subgame perfect equilibrium in the underlying game, which rules out unilateral deviations from rational players at any step of the execution phase of X-Transfer. A key ingredient in the proof of execution phase balance security is the Thora time parameters as computed by Algorithm 4 as well as the correctness of the algorithm in the proof of Lemma 3, which shows that as long as the Thora protocol among the hubs is executed, all PCNs will eventually execute their Thora and update their channels. Computational feasibility of X-Transfer stems from a similar analysis to [48], with the additional terms in the complexity coming from sorting the list of hubs for the greedy algorithm to connect the flows within hub components. We note that the DAG hub topology satisfies optimality as per Definition 2 but our path hub topology solution is not optimal and we provide a worst-case example in Appendix H.2. Nevertheless in Appendix E.2 we conjecture that the DAG solution is impossible without the use of on-chain events, and in Appendix H.2 we detail heuristics utilized in Algorithm 3 to minimize the cost of our path topology solution. Finally, X-Transfer achieves privacy as per Definition 4 so long as the delegates satisfy the assumptions of the underlying MPC protocol.

6 Discussion and Limitations

Participation incentives. As discussed in Section 4.2, neutral net flow hubs are incentivised to execute within-PCN transfers due to potential fee benefits from successful executions of X-Transfer. Here, we elaborate on the incentives for all participants in X-Transfer, including hubs and clients. For clients, the primary incentive is access to a larger pool of transaction partners, including those holding different cryptocurrencies, without incurring high exchange fees. Additionally, X-Transfer's transaction aggregation can lower payment routing fees and enhance liquidity in clients' payment channels with hubs. For hubs, we assume a payment structure where clients deposit fees payable upon the successful execution of X-Transfer (e.g., as in [48] for single-PCN transaction aggregation). The total fees must exceed the hubs' costs, such as computational expenses and the opportunity cost of locked funds, to ensure even neutral net flow hubs find participation financially worthwhile. Nevertheless, determining the exact fee structure for hubs is beyond the scope of this work.

Liquidity management. A complementary challenge to our work is liquidity management, particularly how to prevent channel depletion, which could compromise X-Transfer's liveness. A straightforward but costly solution is for users (both hubs and clients) to return to the blockchain to close and reopen channels with additional funds. A more efficient alternative is off-chain rebalancing [36,15,16], which shifts funds in a cycle of payment channels in order to "top up" a depleted channel; thus incurring significantly less cost than closing and reopening channels on-chain. While Section 3.1 constrains the PCN topology to a star configuration with the hub at the center, this can be viewed as a restricted subgraph of a larger PCN containing multiple cycles. This restricted subgraph is only considered for the execution of X-Transfer, whereas the entire PCN can participate in off-chain rebalancing efforts.

Fixed exchange rates. Our model assumes that tokens in the PCNs are either identical or have fixed exchange rates, which may not always hold true in practice. To address this limitation, one approach is to account for fluctuating exchange rates by leveraging semi-trusted price oracles, such as Chainlink [2], to retrieve real-time rates. Participants can specify their tolerance for exchange rate "slippage"[4], allowing the system to exclude transactions that exceed this tolerance during the aggregation phase of X-Transfer. Alternatively, a base currency, such as Bitcoin or USDC, can be designated, and then wrap all other tokens around the base token. In this setup, the price oracle would be queried only when users need to convert wrapped tokens back to their native currency. For further implementation details of this approach, see [45].

Failure in verification. We examine the setting where there is a failure in the verification process in the aggregation phase of X-Transfer. This could happen if there is some mistake in the inputs (e.g., a client u sending an amount x but x is larger than the total capacity of u's channel with u's hub), or adversarial behavior among some of the participants performing the computation. For input errors, X-Transfer performs a verification process at the start of the MPC

responsible for computation and output of the aggregation phase and removes all faulty inputs. Adversarial behavior during the computation could also impact the output of the MPC (either by leaking information and violating privacy or corrupting the output). We stress, however, that X-Transfer is oblivious to the choice of the underlying MPC protocol and correctness and privacy of the aggregation phase of X-Transfer always holds as long as the adversarial model satisfies the assumptions of the underlying MPC protocol.

Beyond our system model. X-Transfer relies on the system model assumptions outlined in Section 3.1. Understanding its behavior under violations of these assumptions is crucial—particularly in the presence of Byzantine parties, collusion, or periods of asynchrony. While Byzantine clients do not break security, a Byzantine hub can lead to a loss of funds. Collusion between hubs and senders may prevent honest receivers from enforcing updates unless hubs are publicly known and at least one remains honest. Finally, asynchrony, e.g., network partitions, message delays, or offline parties, can cause protocol abortion or loss of funds, especially under malicious hubs. Due to space constraints, we defer a detailed discussion on these threats and potential mitigation strategies to Appendix B.

7 Conclusion

In this work, we presented the first fully off-chain cross-PCN transaction aggregation protocol. We analytically show that X-Transfer is secure, private, computationally feasible, and near-optimal. We envision our work as a first step in achieving secure, fully off-chain interoperability. That said, our work also relies on some assumptions (for instance the path topology) and would it would be an interesting direction for future work to alleviate these assumptions.

Acknowledgments

This work was partially supported by MOE-T2EP20122-0014 (Data-Driven Distributed Algorithms), the Austrian Science Fund (FWF) through the SFB SpyCode project F8512-N, the project CoRaF (grant agreement ESP68-N), the WWTF through project 10.47379/ICT22045, the German Research Foundation (DFG), grant SPP 2378 (ReNO), 2023-2027, and Input Output (http://iohk.io) through their funding of the Edinburgh Blockchain Technology Lab.

References

1. Arbitrum, https://arbitrum.io/, last accessed September 2024
2. Chainlink. https://chain.link/, accessed: 2025-01-14
3. Raiden network, https://raiden.network/, last accessed 4 October 2024
4. Alpos, O., Amores-Sesar, I., Cachin, C., Yeo, M.: Eating sandwiches: Modular and lightweight elimination of transaction reordering attacks. In: OPODIS. LIPIcs, vol. 286, pp. 12:1–12:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)
5. Androulaki, E., Cachin, C., Caro, A.D., Kokoris-Kogias, E.: Channels: Horizontal scaling and confidentiality on permissioned blockchains. In: ESORICS (1). Lecture Notes in Computer Science, vol. 11098, pp. 111–131. Springer (2018)
6. Aumayr, L., Abbaszadeh, K., Maffei, M.: Thora: Atomic and privacy-preserving multi-channel updates. In: CCS. pp. 165–178. ACM (2022)
7. Aumayr, L., Avarikioti, Z., Salem, I., Schmid, S., , Yeo, M.: X-transfer implementation github repository. https://github.com/iosifsalem/X-Transfer (2024)
8. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Bitcoin-compatible virtual channels. IACR Cryptology ePrint Archive, Report 2020/554 (2020)
9. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostakova, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized bitcoin-compatible channels. IACR Cryptology ePrint Archive, Report 2020/476 (2020)
10. Aumayr, L., Moreno-Sanchez, P., Kate, A., Maffei, M.: Blitz: Secure multi-hop payments without two-phase commits. In: USENIX Security Symposium. pp. 4043–4060. USENIX Association (2021)
11. Avarikioti, G., Kokoris-Kogias, E., Wattenhofer, R.: Brick: Asynchronous state channels. CoRR **abs/1905.11360** (2019)
12. Avarikioti, Z., Heimbach, L., Wang, Y., Wattenhofer, R.: Ride the lightning: The game theory of payment channels. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 12059, pp. 264–283. Springer (2020)
13. Avarikioti, Z., Litos, O.S.T., Wattenhofer, R.: Cerberus channels: Incentivizing watchtowers for bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 346–366. Springer (2020)
14. Avarikioti, Z., Lizurej, T., Michalak, T., Yeo, M.: Lightning creation games. In: ICDCS. pp. 1–11. IEEE (2023)
15. Avarikioti, Z., Pietrzak, K., Salem, I., Schmid, S., Tiwari, S., Yeo, M.: Hide & seek: Privacy-preserving rebalancing on payment channel networks. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 13411, pp. 358–373. Springer (2022)
16. Avarikioti, Z., Schmid, S., Tiwari, S.: Musketeer: Incentive-compatible rebalancing for payment channel networks. In: AFT. LIPIcs, vol. 316, pp. 13:1–13:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024)
17. Bünz, B., Kiffer, L., Luu, L., Zamani, M.: Flyclient: Super-light clients for cryptocurrencies. In: SP. pp. 928–946. IEEE (2020)
18. Cao, S., Yuan, Y., Caro, A.D., Nandakumar, K., Elkhiyaoui, K., Hu, Y.: Decentralized privacy-preserving netting protocol on blockchain for payment systems. In: Bonneau, J., Heninger, N. (eds.) 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, pp. 137–155. Springer (2020). https://doi.org/10.1007/978-3-030-51280-4_9, https://doi.org/10.1007/978-3-030-51280-4_9

19. Chapman, J., Garratt, R., Hendry, S., McCormack, A., McMahon, W.: Project jasper: Are distributed wholesale payment systems feasible yet? (2017)
20. Community, L.N.R.: Data about the past and current structure of the lightning network, https://github.com/lnresearch/topology, last accessed September 2024
21. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A.E., Miller, A., Saxena, P., Shi, E., Sirer, E.G., Song, D., Wattenhofer, R.: On scaling decentralized blockchains - (A position paper). In: Financial Cryptography Workshops. Lecture Notes in Computer Science, vol. 9604, pp. 106–125. Springer (2016)
22. Decker, C., Russell, R.: eltoo : A simple layer 2 protocol for bitcoin (2018), https://api.semanticscholar.org/CorpusID:49253813
23. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Pelc, A., Schwarzmann, A.A. (eds.) Stabilization, Safety, and Security of Distributed Systems. pp. 3–18. Springer International Publishing, Cham (2015)
24. DeCred: Decred-compatible cross-chain atomic swapping. https://github.com/decred/atomicswap
25. Dotan, M., Pignolet, Y.A., Schmid, S., Tochner, S., Zohar., A.: Survey on blockchain networking: Context, state-of-the-art, challenges. In: Proc. ACM Computing Surveys (CSUR) (2021)
26. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: IEEE Symposium on Security and Privacy. pp. 327–344 (2017)
27. Egger, C., Moreno-Sanchez, P., Maffei, M.: 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. pp. 801–815. ACM (2019)
28. Eisenbrand, F., Weismantel, R.: Proximity results and faster algorithms for integer programming using the steinitz lemma. ACM Transactions on Algorithms (TALG) **16**(1), 1–14 (2019)
29. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 9057, pp. 281–310. Springer (2015)
30. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 12059, pp. 201–226. Springer (2020)
31. Güntzer, M.M., Jungnickel, D., Leclerc, M.: Efficient algorithms for the clearing of interbank payments. Eur. J. Oper. Res. **106**(1), 212–219 (1998). https://doi.org/10.1016/S0377-2217(97)00265-8, https://doi.org/10.1016/S0377-2217(97)00265-8
32. Guo, Y., Xu, M., Yu, D., Yu, Y., Ranjan, R., Cheng, X.: Cross-channel: Scalable off-chain channels supporting fair and atomic cross-chain operations. IEEE Trans. Computers **72**(11), 3231–3244 (2023)
33. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024), https://www.gurobi.com
34. Herlihy, M.: Atomic cross-chain swaps. In: Proceedings of the 2018 ACM symposium on principles of distributed computing. pp. 245–254 (2018)
35. Jia, X., Yu, Z., Shao, J., Lu, R., Wei, G., Liu, Z.: Cross-chain virtual payment channels. IEEE Trans. Inf. Forensics Secur. **18**, 3401–3413 (2023). https://doi.org/10.1109/TIFS.2023.3281064, https://doi.org/10.1109/TIFS.2023.3281064
36. Khalil, R., Gervais, A.: Revive: Rebalancing off-blockchain payment networks. In: CCS. pp. 439–453. ACM (2017)

37. Madathil, V., Thyagarajan, S.A.K., Vasilopoulos, D., Fournier, L., Malavolta, G., Moreno-Sanchez, P.: Cryptographic oracle-based conditional payments. In: NDSS. The Internet Society (2023)
38. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: NDSS. The Internet Society (2019)
39. McCorry, P., Bakshi, S., Bentov, I., Meiklejohn, S., Miller, A.: Pisa: Arbitration outsourcing for state channels. In: AFT (2019), 10.1145/3318041.3355461
40. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 11598, pp. 508–526. Springer (2019)
41. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
42. Pietrzak, K., Salem, I., Schmid, S., Yeo, M.: Lightpir: Privacy-preserving route discovery for payment channel networks. In: Networking. pp. 1–9. IEEE (2021)
43. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf (2015)
44. Scaffino, G., Aumayr, L., Avarikioti, Z., Maffei, M.: Glimpse: On-demand pow light client with constant-size storage for defi. In: USENIX Security Symposium. pp. 733–750. USENIX Association (2023)
45. Scaffino, G., Aumayr, L., Bastankhah, M., Avarikioti, Z., Maffei, M.: Alba: The dawn of scalable bridges for blockchains. IACR Cryptol. ePrint Arch. p. 197 (2024)
46. Shafransky, Y.M., Doudkin, A.A.: An optimization algorithm for the clearing of interbank payments. Eur. J. Oper. Res. **171**(3), 743–749 (2006). https://doi.org/10.1016/j.ejor.2004.09.003, https://doi.org/10.1016/j.ejor.2004.09.003
47. Spilman, J.: Anti dos for tx replacement. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html (2013)
48. Tiwari, S., Yeo, M., Avarikioti, Z., Salem, I., Pietrzak, K., Schmid, S.: Wiser: Increasing throughput in payment channel networks with transaction aggregation. In: AFT. pp. 217–231. ACM (2022)
49. Yao, A.C.C.: Protocols for secure computations. 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982) pp. 160–164 (1982), https://api.semanticscholar.org/CorpusID:62613325
50. Zabka, P., Förster, K., Decker, C., Schmid, S.: Short paper: A centrality analysis of the lightning network. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 13411, pp. 374–385. Springer (2022)
51. Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.J.: XCLAIM: trustless, interoperable, cryptocurrency-backed assets. In: IEEE Symposium on Security and Privacy. pp. 193–210. IEEE (2019)
52. Zhang, X., Qian, C.: A cross-chain payment channel network. In: ICNP. pp. 1–11. IEEE (2023)

A Additional Details on PCNs and Routing

Payment channels. Payment channels [47,43,23,11,22,40] were introduced as a mitigation technique for the limited transaction throughput of blockchains [21]. Two parties u and v can open a payment channel by depositing funds into a "common account" on the blockchain. These funds are locked and can only be

used in this channel between u and v. Every time the parties want to transact with each other, they update the distribution of their funds. For instance if u, v initially locked c_u, c_v coins respectively, and then u sends δ coins to v, the updated balances will be $c_u - \delta, c_v + \delta$. Note that the balances must always be non-negative, meaning that a party can never send more money than it owns in the last update. Furthermore, the sum of the balances is always the same and equal to the initial funds locked in the channel, $c_u + c_v$. The channel updates occur off-chain, hence the only parties that know the current balances of the channel between u, v are the parties u, v. Note that to update the channel only the two parties u, v need to agree, therefore enforcing any operation that does not benefit one of the parties is impossible. For instance, we cannot enforce "burning money" in a payment channel off-chain (without using the blockchain), because the two parties can simply update the distribution of funds at a later point and split the burned coins as they both benefit from it.

To close a payment channel, a closing transaction is posted on-chain that distributes the initially locked funds (ideally) as agreed in the last update among the parties. To enforce the last update is indeed the one posted on-chain, so no party tried to cheat, different mechanism are in place depending on the payment channel construction we refer to, e.g., replacement via decreasing timelocks [23], incentives via punishment [43]. In this work, we are agnostic to the exact security mechanism in place and assume the payment channel construction operates securely, i.e., if the channels are updated correctly and in an incentive-compatible to the parties manner, then they will close correctly if needed.

Payment channel networks (PCNs). Several payment channels opened on the same blockchain form a payment channel network or PCN. A node on a PCN represents a channel party, and an edge represents a channel among the two nodes/parties it connects. Via this network, parties can transact with each other even if they are not directly connected via a payment channel. Instead, the sender of a transaction may find a path of channels that connects sender and receiver where each channel on the path has enough funds (on the correct direction) to route the transaction. For instance, if u wants to send δ coins to v via node z, then u must own at least δ coins in the channel (u, z) and z must own at least δ coins in the channel (z, v). If such a path exists, the (multi-hop) transaction can be routed atomically though the path, i.e., either all transactions on the path will be executed or none. Each channel will be consequently updated with the new balances.

To enforce the atomicity of multi-hop transactions typically Hash Timelock Contracts (HTLCs) are employed [43]. An HTLC is a smart contract between two parties u, v that locks some coins and enforces that either the receiver (say, v) can claim the coins only if he reveals a secret (preimage of a hash, enforced via a hashlock) within a specific time period T (enforced via a timelock), or the sender (say, u) can then reclaim the coins after T elapses.

Routing fees. In multi-hop transactions typically the intermediaries that enable the routing of a transaction ask for a routing service fee. This fee is typically

proportional to the transaction value (plus a base fee). When a sender u routes a transaction of δ coins to v through z and y that ask for fees f_z and f_y respectively, the total amount u must send is $\delta + f_z + f_y$. For further details, we refer the reader to [30].

HTLCs [43]. To enforce the atomicity of multi-hop transactions typically Hash Timelock Contracts (HTLCs) are employed. An HTLC is a smart contract between two parties s, r that locks some coins and enforces that either the receiver r can claim the coins only if they reveal a secret (preimage of a hash, enforced via a hashlock) within a specific time period T (enforced via a timelock), or the sender s can then reclaim the coins after T elapses. We write $HTLC(s, r, x, h, T)$ to denote an HTLC between a sender s and a receiver r of amount x, hashlock h and timelock T.

B Beyond our system model

Here, we expound on the security and liveness of X-Transfer under violations of the system model assumptions (see Section 3.1) in adverse conditions. Specifically, we examine the effects of Byzantine parties – who may deviate arbitrarily, even at a financial loss – as well as periods of asynchrony and collusion.

a) Byzantine behaviour. We examine Byzantine behavior in X-Transfer for both clients and hubs. With *Byzantine clients* (but rational hubs), security holds: once Thora instances are set up, execution cannot be blocked, as rational hubs will enforce it. Any deviation before setup only leads to an incomplete setup, allowing the process to restart without the non-cooperating party. However, a *Byzantine hub* compromises security, potentially causing other parties to lose funds. Consider three PCNs, G_1, G_2, G_3, with hubs h_1, h_2, h_3. If h_1 and h_2 have positive net flow while h_3 has negative net flow, X-Transfer relies on rational hubs's incentives to execute Thora after the between-hub Thora completes. A malicious hub (e.g., h_1) could refuse to execute Thora in G_1, even when profitable, leading to fund loss for senders in G_2. Ensuring Byzantine resilience in cross-chain PCNs remains an open challenge for future research.

b) Collusion. Without additional assumptions, X-Transfer is vulnerable to collusion between clients and hubs. A hub incentivized to execute the Thora update in its PCN can collude with a sender to withhold the preimage, compensating the hub for any loss, leaving an honest receiver unable to enforce the update. This issue is mitigated if hubs are publicly known and at least one is honest, allowing affected parties to query all hubs and obtain the preimage, thereby ensuring that the update is enforceable.

c) Periods of asynchrony. We examine violations of our synchrony assumption, including network partitions, message delays or losses, offline parties, and blockchain liveness disruptions (e.g., censoring attacks). X-Transfer relies on Thora for safety, which holds only under a synchronous network model (due to timelocks), parties remain online and can post messages on-chain within a bounded timeframe (i.e., synchronous blockchains with bounded liveness [29]). Network partitions, timing violations, offline parties, and message losses have

similar effects. During setup (e.g., secret generation or Thora setup), they cause protocol abortion. During execution, they can violate balance security, at least for the affected party, e.g., an offline recipient risks losing their funds. In the worst case, such as with a malicious hub, going offline could further endanger the balance security of other parties, exacerbating the impact.

C Thora details

Thora [6] is a multi-channel update protocol that allows updating a set of channels in a PCN atomically without any restriction of the topology of the to-be-updated channels. For each channel that is to be updated, denote the sending party (i.e., party that pays coins) as s and the receiving party (i.e., party that gains coins) as r.

Transaction Details. The key transactions in Thora are the state update transaction, payment transaction, refund transaction, and enable payment transaction. The state update transaction $tx_{s,r}^{state}$ creates outputs that correspond to outcomes of the channel update. The refund transaction $tx_{s,r}^{r}$ refunds the locked coins to the sending party s after a set amount of time has elapsed, specified by a global refund timeout value T. Note that as long as T amount of time has elapsed, all sending users in all channels can retrieve their coins using this refund transaction. The payment transaction $tx_{s,r}^{p}$ makes the payment from s to r. Finally, the enable payment transaction tx_{r}^{ep} is used to enforce all payments from sending parties to receiving parties in all channels, ensuring atomicity of payments. This is done by enabling all receiving parties to enforce payments along their channels as long as there exists a single $tx_{r_i}^{ep}$ on the blockchain corresponding to some receiving party r_i, in particular, by having tx_{r}^{ep} be an input to $tx_{s,r}^{p}$. Usually, tx_{r}^{ep} would be posted on the blockchain due to malicious activity of senders in order to allow receivers to enforce payments. Finally, a second unit of time $t_c < T$, is the upper bound on the time needed to close a channel and is included in all enable payment transactions to account for potential race conditions when closing channels.

Protocol Overview. The protocol proceeds in four phases. In the (i) *pre-setup*, each receiver r of a to-be-updated channel creates its own tx_{r}^{ep}, which includes an output for each receiver and requires the signatures from all senders and r. In the (ii) *setup*, each sender s creates $tx_{s,r}^{state}$ reflecting the desired update and $tx_{s,r}^{p}$, sign it, and send it to their receiving neighbor. Then, each channel updates to $tx_{s,r}^{state}$ and sends a confirmation to all other parties. After receiving such confirmation from every channel receiver, in the (iii) *confirmation* phase, each sender signs every tx_{r}^{ep} transaction and sends this signature to the corresponding r. Once a receiver has such a signature from every sender, they send another confirmation to every sender. Finally, in the (iv) *finalization* phase, all channels can be updated to a new state reflecting the update (optimistic case), or else the receiver can enforce the update by posting tx_{r}^{ep}. Note that step (iii) ensures that the receivers have every required signature, and a sender will only agree to an update if they have a confirmation of every receiver. Thus, atomicity is ensured.

Thora Properties. Thora guarantees two properties, as informally presented below. The full, formal definitions are game-based on top of a UC ideal functionality and can be found in [6].

- **Atomicity.** A multi-channel update protocol has atomicity if there are no two channels with at least one honest user each, where one update fails, and the other one is successful. The exception is if there is an (irrational) adversarial user in one of these channels who can change the outcome by forfeiting at least the amount of coins of the update to the honest user. This means that (i) a malicious receiver cannot enforce the update of her channel if it should fail, and (ii) a malicious sender cannot let the update fail even though it should succeed, without forfeiting their coins to the honest party in their channel.
- **Strong value privacy.** A multi-channel update protocol has strong value privacy if no party except for the users of a channel learns about the value of the update in the optimistic case.

D Formal definition of privacy

We formally describe a game-based notion of privacy we want X-Transfer to achieve. Let us denote by $\mathcal{A}$ a passive adversary and Π a cross-chain transaction aggregation protocol. Recall that $G = (V, E)$ denotes the graph which is the union of all PCNs. Let $G' = (V', E')$ denote the subgraph of G that is formed by all users that are interested in participating in the protocol.

Definition 4 (Privacy). *We say a cross-chain transaction aggregation and execution protocol Π is ϵ-private if the probability that $\mathcal{A}$ wins the following indistinguishability game is upper bounded by $\frac{1}{2} + \epsilon$ for some $\epsilon > 0$. If ϵ is negligible, we say Π is private.*

- *$\mathcal{A}$ chooses a subset of nodes $V^{\mathcal{A}} \subset V'$ to corrupt. Note that $\mathcal{A}$ gains access to transcripts of each corrupted node. Let $G^{\mathcal{A}}$ denote the union of all corrupted nodes and their incident edges. The set $V^{\mathcal{A}}$ can consist of hubs, clients, or a mix of both. If the set $V^{\mathcal{A}}$ consists of a hub node, we add all hubs in G' and their incident edges to $G^{\mathcal{A}}$. Let h_j^i denote the ith corrupted hub in PCN j. To connect the graph $G^{\mathcal{A}}$, for each corrupted hub h_j^i, we add connecting edges from h_j^i to all other hubs in G' which are in PCNs $k \neq j$ to the subgraph $G^{\mathcal{A}}$.*
- *For $i \in \{0, 1\}$, $\mathcal{A}$ creates the following list of transaction tuples $\mathcal{T}^i = \{x_j, s_j, r_j\}_{i=1}^{n_i}$ where x_j represents the transaction amount of the jth transaction in the list, and $s_j, r_j \in V'$ represent the sender and recipient of the jth transaction respectively. n_i is the number of transactions in the ith list.*
- *For every node and channel in the subgraph $G^{\mathcal{A}}$, the following condition needs to hold: the resulting aggregated flow returned by Π restricted to $G^{\mathcal{A}}$ when run on the graph G' when given transaction tuple lists $\mathcal{T}^0$ and $\mathcal{T}^1$ has to be the same. Moreover, the set of involved users as outputted by Π has to be the same when given $\mathcal{T}^0$ or $\mathcal{T}^1$ as input.*

- *Challenge phase: $\mathcal{A}$ uniformly random bit $b \in \{0,1\}$ is sampled and the protocol Π is run on input $\mathcal{T}^b$.*
- *$\mathcal{A}$ gets the flow output sent to each corrupted user from Π.*
- *$\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{A}$ wins the game if $b' = b$.*

E Technical details of X-Transfer

E.1 Greedy algorithm for computing links between hubs

Algorithm 2: Aggregation phase: computing links between hubs

Input : PCN channel attributes (nodes, channels, balances) and the optimal client-to-hub flows computed by the ILP
Output: per channel flows (computed from the list of successful transactions)

Sort senders and receivers in descending order of amounts and leave the 0-flow nodes as a separate set to be dealt later;
while *(there is only one sender AND it's currently selected) OR (there are at least two senders overall AND there are at least two senders from the original list of senders with an amount to be sent)* **do**
  Pick the sender with the current largest amount, say x is the amount;
  Draw an edge of weight x to the receiver with the largest amount;
  if *the receiver awaits for more funds* **then**
    we declare this a sink node
if *the receiver awaits for exactly x* **then**
  we call this path fixed
if *the receiver awaits for an amount y that is smaller than x* **then**
  then the receiver is marked as a new sender with amount x-y and added to an ordered list of senders
/* This creates paths that are the longest possible according to the requirements in the while loop. It also covers the case of a star demand with one sender. This case gives a $\Theta(n^2)$ lower bound of flows, where $\Theta(n)$ is possible with a DAG. */

Let us denote the optimal solution as outputted by the ILP by $(x_1^*, \ldots, x_n^*)$. We first notice that the demand vector for the optimal solution $d^* := \sum_{t_i \in \mathcal{T}^*} x_i^* t_i$, i.e., the vector that aggregates all incoming and outgoing flows per client for the optimal subset $\mathcal{T}^*$, specifies outgoing and incoming flows for each client. When summing the client-hub flows within a PCN, we obtain a positive value when the PCN's hub needs to send funds to other hubs and a negative value when the hub needs to receive funds. We refer to hubs with outgoing and incoming flow as supply and demand hubs respectively. We classify hubs with neither outgoing or incoming flow, i.e., incoming funds equals outgoing funds, as supply or demand hubs arbitrarily. Our algorithm is described in Algorithm 2 and works as follows. We sort the transaction amounts of the supply and demand hubs in descending order. Then, for every demand, we add hub-to-hub links with as many supplying hubs needed to cover that demand. We re-sort the list of supplying hubs (the last supplying hub is updated with the remaining amount to

send) and repeat for the next demand, until all of them are satisfied (by design, total demand matches total supply). Since the set of supply and demand hubs are disjoint, the resulting graph is a bipartite graph, which is not necessarily connected. Indeed if the greedy algorithm yields less than $k-1$ links, then we connect the disconnected components by adding a link from a sender of one to a receiver of another one with flow $\epsilon > 0$.

E.2 Restricting the topology of the hubs to a path

The case for a path-based solution. A crucial component of X-Transfer is to restrict the aggregated flow of transactions between hubs to a path topology. That is, each hub receives transactions from and sends transactions to *at most* one other hub. The main reason for doing so is to avoid the case where the hub topology has 2 or more leaf nodes. This is because the execution phase of X-Transfer employs secrets that link all transactions together so as to guarantee the atomicity of transaction execution. We first observe that any hub generating the secret has to be a hub that only receives funds from other hubs (so that the hub is incentivized to reveal the secret later and pull the funds from the other hubs). Indeed Section 4.3 provides an example of an attack that could happen when a sending hub generates the secret.

With multiple receiving hubs as could happen with a DAG hub topology, there are essentially two options for secret generation: (1) all involved transactions in all PCNs are linked by a common secret, and (2) some transactions in some PCNs are linked by a common secret, while the remainder PCN have individual secrets. Unfortunately, there are attacks that could happen when employing either of these options. In what follows, we will describe these two options in detail, as well as delineate the plausible attacks. We will use the simple example of 3 PCNs (one sending PCN and two receiving PCNs) depicted in Figure 4 to illustrate these problems with the DAG hub topology.

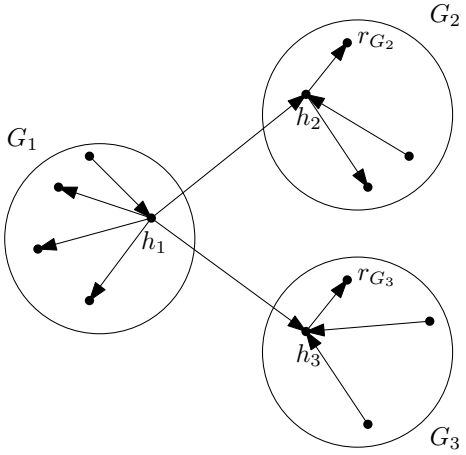

Fig. 4. Executing payments on a DAG

The first option for linking all transactions together is to use a common secret, which can be generated by the two receiving hubs h_2 and h_3. More specifically, h_2 and h_3 independently sample secrets $\mathfrak{s}_2$ and $\mathfrak{s}_3$ and combine them using some simple MPC procedure. For instance, this can be done by computing $\mathfrak{s} = \mathfrak{s}_2 \oplus \mathfrak{s}_3$ where $\oplus$ denotes the XOR operator on two binary strings. Then, $H(\mathfrak{s})$ is used as a common secret that connects all involved transactions in all PCNs.

The second option would be to use individual secrets to govern the execution of transactions within each receiving PCN. That is, h_2 and h_3 independently sample $\mathfrak{s}_2$ and $\mathfrak{s}_3$ and the transactions within G_2 and G_3 are governed by $H(\mathfrak{s}_2)$ and $H(\mathfrak{s}_3)$ respectively. The transactions in G_1 as well as hub-to-hub channel updates can then be governed by some combination of both secrets, e.g., $H(\mathfrak{s}_2) \oplus H(\mathfrak{s}_3)$.

In both of these settings, the main issue is that in first party that reveals the secret in the revelation phase is disadvantaged. In the first setting with a common secret, suppose h_3 reveals $\mathfrak{s}_3$ first. Then h_2 could pretend to go offline and collude with some recipients in G_3 (e.g., r_{G_3} in Figure 4) to leak $\mathfrak{s}_2$ to them for a small cut, resulting in them pulling funds from h_3 while preventing h_3 from pulling funds from h_1 as h_3 does not know $\mathfrak{s}_2$. In the second setting, suppose h_2 reveals $\mathfrak{s}_2$ during the revelation stage and h_3 goes offline. h_1 could collude with r_{G_2} and send r_{G_2} $\mathfrak{s}_2$ to enable r_{G_2} to pull funds from h_2 (in fact if we use Thora to update channels all involved transactions in G_2 will be executed) in exchange for a small cut.

The above issues could be resolved if we create some global on-chain event, for instance a transaction that contains both secrets, such that all cross-PCN transactions can be executed upon the existence of this event on some blockchain. We stress that a pure off-chain solution would not be able to replicate this public and global dissemination of information as per an on-chain event, as all information in PCNs is either localized to users or shared between two parties in channels. More generally, we conjecture that simultaneous release of secrets (as in the case with multiple receiving hubs) is impossible with rational users without on-chain events, which implies that without on-chain events atomicity of transaction execution in the cross-PCN setting where there could be multiple receiving hubs as in the DAG topology is impossible. We leave further exploration of this conjecture to future work.

Algorithm to convert the DAG output to a path. We denote the output of the above greedy algorithm that aims to connect all hubs as $G = (V, E)$ where the vertices of G are the hubs. Note that G is a DAG. For each hub vertex $v \in G$, we assign two attributes:

1. the *type* of v, that is, whether v is a sending, receiving, or a middle vertex. Sending vertices are vertices with in-degree 0, receiving vertices are vertices with out-degree 0, and vertices with both nonzero in and out degrees are classified as middle vertices.
2. the *depth* of v, that is, the length of the shortest path between v and any sending vertex.

Algorithm 3: Aggregation phase: computing a path from a DAG

input: DAG $G = (V, E)$, each vertex in G with attributes type and depth
$E' = \emptyset$
$W = 0$
$S = \{v \mid v.type = \mathsf{sending}\}$, sorted in ascending order of total edge weights
$R = \{v \mid v.type = \mathsf{receiving}\}$, sorted in descending order of total edge weights
$M = \{v \mid v.type = \mathsf{middle}\}$, sorted in ascending order of total outgoing weights
$V = S \parallel M \parallel R$
for i *in* $[|V| - 1]$
  create edge $e = (v_i, v_{i+1})$
  $W = W + w^G_{out}(v) - w^G_{in}(v)$
  set the edge weight of e to w
  $E = E \cup e$
return $P = (V, E')$

We describe the process of converting G into a path graph P in Algorithm 3 and illustrate this transformation on an example in Figure 5. The invariant that Algorithm 3 maintains is *balance conservation* of the vertices, that is, the difference between the sum of outgoing edges and incoming edges must be the same for all vertices in both G and P. Formally, let $w^G_{out}(v)$ (resp. $w^G_{in}(v)$) denote the total weights of outgoing (resp. ingoing) edges of v in graph G. Then for all $v \in G$, $w^G_{out}(v) - w^G_{in}(v) = w^P_{out}(v) - w^P_{in}(v)$. To convert G into a path, the algorithm simply creates creates edges in ascending order among all sending vertices with the weight of each newly created edge being the total amount of funds sent out by the previous sending vertices in the sequence. This process is similarly repeated for the middle vertices (in ascending order in the total outgoing edge weights), this time removing the amount each middle node should receive from the newly created edge weights. Finally, we repeat the process again for the receiving nodes in descending order of total edge weights, this time subtracting the amount each receiving node should receive from the weight of each newly created edge.

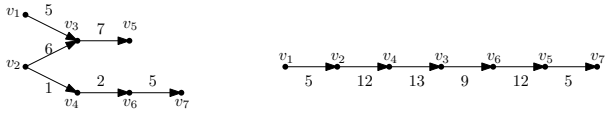

Fig. 5. Transformation of DAG to path

E.3 Computing execution time parameters

Let us denote by P the path graph computed from the previous stage where the vertices of P are the hubs $h_1, \ldots, h_k$. Let $V_i = G_i \setminus \{h_i\}$ be the nodes in the ith PCN G_i excluding the hub h_i. Let $\mathsf{flow}(v, h_i) \in \mathbb{R}$ be the monetary transfer between a single node $v \in V_i$ and h_i, where $|\mathsf{flow}(v, h_i)|$ denotes the size of the transaction, and $\mathsf{flow}(v, h_i) > 0$ represents incoming flow of funds from v to h_i and $\mathsf{flow}(v, h_i) < 0$ represents outgoing flow of funds from h_i to v. Let $tf_{h_i} = \sum_{v \in V_i}(\mathsf{flow}(v, h_i))$ denote the total flow, that is, the total amount of monetary transfers from all nodes in V_i to h_i. Finally, let $nf_{h_i} = \mathsf{sgn}(tf_{h_i})$ where sgn is the sign function denote the *net flow* to h_i. We stress that both the total and net flow *do not include cross-PCN monetary flows* and is simply meant to measure the overall profit or loss a hub is in when looking only at flows internal to their PCN.

We now state an important lemma regarding net flows in hubs.

Lemma 2. *Either all PCNs G_i have $nf_{h_i} = 0$ or at least one PCN G_i has $nf_{h_i} > 0$ and at least one other PCN G_j has $nf_{h_i} < 0$.*

Proof. Wlog, suppose we only have one hub h_i with $nf_{h_i} > 0$ and all other hubs h_j have $nf_{h_j} = 0$. This means that the total sum received among all users over all PCNs is larger than the total sum sent. This violates balance conservation, a property which is preserved during the aggregation phase. The case for only one hub with $nf_{h_i} < 0$ and all other hubs having neutral total flow proceeds similarly. □

Algorithm 4: Aggregation phase: computing execution phase time parameters.

input: Path $P = (V, E)$, transactions $\mathcal{T}$, initial time T_0, time increment δ
$T = \emptyset$
$S =$ all PCNs $G_i \mid nf_{h_i} \geq 0$
append $(S, T_0 + \delta)$ to T
$i = 1$
$P = P \setminus S$
while $P \neq \emptyset$ **do**
  $i = i + 1$
  $S' =$ all PCNs $G_i \mid \exists (s, r) \in \mathcal{T}$ with $r \in G_i$ and $s \in S$
  append $(S', T_0 + i \cdot \delta)$ to T
  $S = S \cup S'$
return T

Algorithm 4 details the process of obtaining these time parameters, as well as the set of PCNs corresponding to each time parameter. The time parameters are computed based on two assumptions regarding the execution phase of X-Transfer. The first is the assumption that hubs that are in a position of positive or neutral net flow will be incentivized to execute the protocol and update the channels inside their own PCNs. For hubs in a positive net flow position, the

incentive comes in the funds they will receive in the process. For hubs in a neutral net flow position, we assume that they would still rather execute the protocol than not in order to benefit from the fees gained from a successful execution of X-Transfer. The second assumption is the existence of a mechanism that allows clients in a PCN to enforce channel updates, which will be enforced in the execution phase of X-Transfer by the usage of Thora to update channels.

Informally, Algorithm 4 computes these time parameters by first assigning all PCNs with hubs that have positive or neutral net flow, i.e., $nf_{h_i} = 1$ or $nf_{h_i} = 0$, the smallest time parameter. The rationale behind this goes back to our first assumption that, assuming all cross-PCN flows are already fulfilled and their corresponding channels updated, hubs that are in a position of positive or neutral net flow will be incentivized to update the channels inside their own PCNs. Thus, these hubs should execute the channel updates within their PCNs first. Let us label these PCNs as "safe PCNs". After these PCNs have been updated, Algorithm 4 then looks for PCNs with negative net flow hubs, but contain a recipient r such that the corresponding sender to r lies in a safe PCN (Lemma 3 shows this always exists). These PCNs should execute their internal channel updates next as there will be a sender who has already sent out funds and their corresponding recipient would be incentivized to enforce the updates in their PCN so as to obtain their funds. We then augment the set of safe PCNs by these new PCNs and assign these PCNs a longer execution time parameter. Finally, we repeat the process until no PCNs are left.

The following lemma proves the correctness of Algorithm 4.

Lemma 3. *Algorithm 4 terminates.*

Proof. Choose a random PCN G_i with hub h_i with negative net flow. Since $nf_{h_i} < 0$ and P is connected, this means that h_i has to be receiving funds from another PCN as we assume that all hubs only facilitate the flow of funds and do not send or receive money in their PCN. Choose a random recipient in G_i with a corresponding sender outside their PCN. Note that this always exists since the hub has negative net flow so the outgoing funds to recipients in G_i is more than the incoming funds from senders in G_i. If the sender lies in a PCN say G_j also with negative net flow hub, choose a random recipient in G_j with sender outside the PCN that is also not in G_i. Again this must exist since the sum of total incoming funds from senders in G_i and G_j is smaller than the total outgoing funds to recipients in both PCNs. From Lemma 2, this process must terminate at a PCN with a hub with either positive or neutral net flow. This observation shows that all PCNs will eventually be added to the safe set and thus Algorithm 4 terminates. □

E.4 Execution phase

Modifications to Thora pre-setup, setup, and confirmation phases. Here we detail the modifications made to the pre-setup, setup, and confirmation phases of Thora to enable the usage of an additional hashlock to lock transactions.

Algorithm 5: Thora pre-setup with hashlock

input: PCN G, blockchain delay parameter Δ, hashed secret $H(\mathfrak{s})$

$S :=$ set of senders in G
Each recipient $r_i \in G$ creates $tx^{in}_{r_i} = \# \mapsto [(\epsilon|\sigma_{r_i,S})]$
Each $r_i \in G$ creates $tx^{ep}_{r_i} = [(\epsilon|\sigma_{r_i,S})] \mapsto [(\epsilon|\sigma_{r_i} \wedge H(\mathfrak{s}) \wedge \Delta)]$
Each $r_i \in G$ sends $tx^{ep}_{r_i}$ to all involved parties in G

Algorithm 6: Thora setup with hashlock

input: PCN G, time parameter T, blockchain delay parameter Δ, hashed secret $H(\mathfrak{s})$

$C := \{e_i\}$ set of involved channels in G
for *each edge* $e_i = (s, r) \in C$
  $p :=$ payment amount
  $b_s :=$ balance of sender s
  $b_r :=$ balance of recipient r
  s creates $tx^{state}_{s,r} = \# \mapsto [(p|(\sigma_s \wedge T) \vee (\sigma_{s,r} \wedge \Delta)), (b_s - p|\sigma_s), (b_r|\sigma_r)]$
  s creates $tx^{r}_{s,r} = (p|(\sigma_s \wedge T) \vee (\sigma_{s,r} \wedge \Delta)) \mapsto (p|\sigma_s)$
  s creates $tx^{p}_{s,r} = [(p|(\sigma_s \wedge T) \vee (\sigma_{s,r} \wedge \Delta)), (\epsilon|\sigma_r \wedge H(\mathfrak{s}_l) \wedge \Delta)] \mapsto (p + \epsilon|\sigma_r)$
  s sends $tx^{state}_{s,r}$ and the signed $tx^{p}_{s,r}$ to r
$E = \emptyset$
for *edge* $e_i = (s, r) \in C$
  r checks transactions sent from s
  if *all transactions are correct and valid* **then**
    add endorsement to E
if $|E| = m_j$ **then**
  broadcast final endorsement

F A Simple Example of X-Transfer: Three PCNs

We illustrate how X-Transfer works on a simple example of three PCNs G_1, G_2, G_3 with corresponding blockchains $B_{G_1}, B_{G_2}, B_{G_3}$ and hubs h_1, h_2, h_3. Multiple parties in each PCN are involved in the protocol, and we assume that the resulting transaction flow after the aggregation phase is a transaction path from G_1 to G_2 to G_3 transferring x_1 from h_1 to h_2 and x_2 from h_2 to h_3. Figure 6 depicts this setting.

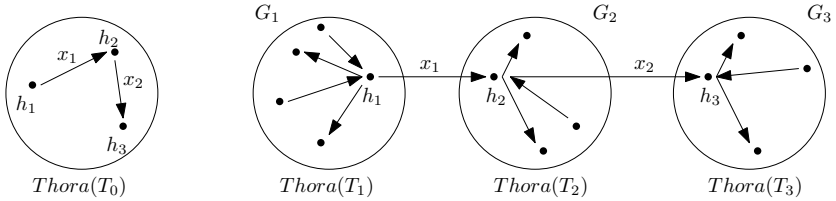

Fig. 6. Execution of X-Transfer on 3 PCNs G_1, G_2, G_3.

Algorithm 7: Thora confirmation with hashlock

input: PCN G

$R :=$ set of all receivers in G
$C := \{e_i\}$ set of involved channels in G
for *each edge* $e_i = (s, r) \in C$
  if *s gets an endorsement from r $\forall r \in R$* **then**
    update e_i using $tx_{s,r}^{state}$
  if *e_i is updated successfully* **then**
    for $j \in [|C|]$
      send $\sigma(tx_j^{ep})$ to receiver of e_j
for *each edge* $e_i = (s, r) \in C$
  if *r receives all signatures on tx_r^{ep}* **then**
    check if e_i is updated successfully
    send endorsement to all parties

Aggregation phase The aggregation phase begins with all interested parties secret sharing their transactions as well as their balance information. This information is given to the MPC delegates and the ILP optimization problem as defined in Equation 1 is then solved among these delegates. Additionally, the delegates compute the path solution, as well as the relevant Thora time parameters for each PCN. We stress that all of the above computation is done using MPC. The output of this protocol at this phase will be the requisite funds each involved party would need to transfer to or receive from their respective hubs, the amount of funds each hub sends to each other, and the Thora time parameters.

Execution phase. The execution phase begins with setting up the Thora protocols among the involved parties. First, the hub h_3 generates a random secret $\mathfrak{s}$ and broadcasts $H(\mathfrak{s})$ to the other hubs. Then, each hub uses $H(\mathfrak{s})$ to set up a Thora in their own PCN with the additional hashlock $H(\mathfrak{s})$ and time parameter as output by the aggregation phase. Once all of these Thoras are verified to be set up correctly, the hubs set up another Thora among their channels with a smaller time paramter T_0.

Once the set up is complete, h_3 broadcasts $\mathfrak{s}$ to the hubs which allows them to execute the Thora among their channels (the Thora protocol with parameter T_0 in Figure 6). After the execution and update of the within-hub channels, h_1 is the hub with positive net flow within G_1 as h_1 is sending funds out to h_2. Thus, h_1 would be incentivized to execute the Thora protocol within G_1. From Lemmas 2 and 3, either h_2 also has an incentive to execute the Thora protocol (if it has neutral net flow within G_2) or there would be a sender reciever pair (s, r) with s in G_1 and r in G_2. Thus, once the execution of Thora in G_1 is done, s would know the secret $\mathfrak{s}$ and can correspondingly inform r, which enables the execution of Thora in G_2. Note that the time parameter for the Thora execution in G_2 as input in the set up phase would naturally account for the time taken to

communicate this secret. Finally, a similar argument enables the execution of Thora in G_3.

G Omitted Lemmas and Proofs

G.1 Proof of Lemma 1

Lemma 1. *Let $|E_{greedy}|$ the number hub-to-hub of links created by our greedy algorithm and $|\mathcal{H}_s|$, $|\mathcal{H}_d|$ be the number of supply and demand hubs, respectively. Then,* $\max\{|\mathcal{H}_s|, |\mathcal{H}_d|\} \leq |E_{greedy}| \leq k-1$.

Proof. The lower bound follows from the fact that there is at least one outgoing link from every demand node and at least one incoming link to every supply node. We prove the upper bound by induction on the number of hubs k. We denote the supplying hubs with $\mathcal{H}_s = \{H_1^s, \ldots, H_x^s\}$, the demand hubs with $\mathcal{H}_d = \{H_1^d, \ldots, H_y^d\}$, and the amount to be sent or received with $|H|$, $H \in \mathcal{H}_s \cup \mathcal{H}_d$.
Induction base: The claim is true for $k = 2$.
This trivially holds, since there is only one supply and one demand hub, which the algorithm connects. The case of $k = 3$ is also trivial. The case of $k = 4$ is the first non-trivial one, as it can be solved with 2, 3, or 4 nodes in general, depending on the actual flows. Even though it is not necessary for the base case (but good for intuition), applying the algorithm gives either 2 links when the amounts match exactly or 3 links otherwise $(k-1)$.
Induction hypothesis: We assume that the upper bound holds for $k = n$.
Induction step: We will show that the upper bound holds for $k = n+1$. Consider the sorted demands and supply amounts. We distinguish the demand hub with the smallest amount as H_j^d. From the descending order of amounts of the supply hubs, we distinguish the smallest suffix $\{H_{i_1}^s, \ldots, H_{i_\ell}^s\}$ such that $|H_{i_1}^s| + \ldots + |H_{i_\ell}^s| \geq |H_j^d|$. We consider two cases (see Figure 7 for details): $|H_{i_1}^s| + \ldots + |H_{i_\ell}^s| = |H_j^d|$ and $|H_{i_1}^s| + \ldots + |H_{i_\ell}^s| > |H_j^d|$. In the first case, we define $\mathcal{H}'_s = \mathcal{H}_s \setminus \{H_{i_1}^s, \ldots, H_{i_\ell}^s\}$ and $\mathcal{H}'_d = \mathcal{H}_d \setminus \{H_j^d\}$. We run the greedy algorithm over the $n+1-\ell-1 = n-\ell < n+1$ hubs in $\mathcal{H}'_s \cup \mathcal{H}'_d$. The base case gives us a solution with at most $n-\ell-1 = n-j-1$ links. We augment that solution by adding $(H_{i_1}^s, H_j^d), \ldots, (H_{i_\ell}^s, H_j^d)$, thus ℓ more links, that satisfy the demand of H_j^d and the supplies of the excluded supply hubs. By that, we gain a solution for all hubs, with at most $n-1$ links. This is expected, since the part we excluded is independent from the remainder.

We now consider the case of $|H_{i_1}^s| + \ldots + |H_{i_\ell}^s| > |H_j^d|$. In this case, we artificially duplicate $H_{i_1}^s$ to node A and node B. For node B, we define $|B| = |H_j^d| - |H_{i_2}^s| - \ldots - |H_{i_\ell}^s|$ (i.e., the part of $|H_{i_1}^s|$ needed to sum up to $|H_j^d|$) and $|A| = |H_{i_1}^s| - |B|$. Since $\{H_{i_1}^s, \ldots, H_{i_\ell}^s\}$ is the smallest suffix of ordered supply amounts that is larger than $|H_j^d|$, $|H_{i_1}^s|$ is the largest number in that set and removing it implies $|H_{i_2}^s| + \ldots + |H_{i_\ell}^s| < |H_j^d|$. Then, we define $\mathcal{H}'_s = (\mathcal{H}_s \setminus \{H_{i_1}^s, \ldots, H_{i_\ell}^s\}) \cup \{A\}$ and $\mathcal{H}'_d = \mathcal{H}_d \setminus \{H_j^d\}$, and run the greedy algorithm

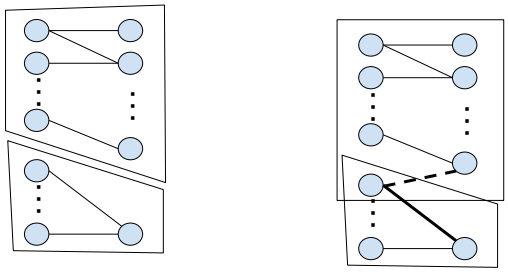

Fig. 7. Left: the first case, where H_j^d is satisfied by ℓ nodes in the suffix exactly. Right: the second case, which is the opposite of the first. In this case $H_{i_1}^s$, i.e., A, will have an extra link (compared to the previous case), to the nodes.

on the $n+1-\ell+1-1 = n+1-\ell$ hubs in $\mathcal{H}'_s \cup \mathcal{H}'_d$. The fact that A is part of $\mathcal{H}'_s$ ensures that the demands and supplies match in $\mathcal{H}'_s \cup \mathcal{H}'_d$. By the induction hypothesis, the solution size is at most $n-\ell$. We augment the solution by (i) adding the ℓ links $(B, H_j^d), (H_{i_2}^s, H_j^d), \ldots, (H_{i_\ell}^s, H_j^d)$, and (ii) merging nodes A and B to the original hub $H_{i_1}^s$. This step produces a solution for all hubs, as it satisfies $|H_j^d|$. The solution has at most $n-\ell+\ell = n$ links. Hence, the induction step and the induction are proved. □

H Analysis of X-Transfer

H.1 Balance security

Lemma 4. *Balance security is satisfied in the aggregation phase, i.e., the aggregated demand vector should correspond to some sublist $\mathcal{T}' \subset \mathcal{T}$ of the input transactions.*

Proof. We will show balance security is satisfied in the solution of the ILP optimization problem, as well as the subsequent algorithms that augment the links between the hubs as well as convert the DAG to a path. We begin with the solution of the ILP. Recall that the solution of the ILP as specified in Equation 1 computes the optimal subset $\mathcal{T}^*$ of transactions to execute and that the demand vector $d^* = \sum_{t_i \in \mathcal{T}^*} x_i^* t_i$ sums up the total incoming and outgoing flows per user. Thus, by definition, d^* corresponds to the the subsequence $\mathcal{T}^*$ of the input transaction sequence $\mathcal{T}$ and so balance security is not violated in this phase.

Next, we show that our greedy algorithm for computing the flows between hubs does not alter the balance of any hub by more than a negligible amount of funds. First observe that the total demand among the hubs is equal to the total supply. This comes from the fact that the solution of the ILP is $\mathcal{T}^* \subset \mathcal{T}$ and the hubs do not send or receive funds. In our greedy algorithm, the hubs are connected such that each demanding hub's demand is fulfilled by some other supplying hub's supply, and when the demand is exactly fulfilled, any excess supply goes to another demanding hub. Since the total hub demand equals total

hub supply, the process terminates with a DAG G (with possibly disconnected components).

Let $\epsilon > 0$ be some negligible fund amount. In the subsequent phase of the greedy algorithm, each connected component in G is connected to the another with a payment channel sending ϵ amount of funds. Thus, the change in balance of any given hub is at most $k\epsilon$ which is negligible.

In the final phase of aggregation, the resulting DAG G is connected to a path P as per Algorithm 3. Note firstly that the graphs G and P share the exact same set of vertices, and that Algorithm 3 maintains the invariant that for all vertices v in both graphs G and P, $d_v^G := w_{out}^G(v) - w_{in}^G(v) = w_{out}^P(v) - w_{in}^P(v) =: d_v^P$. Since the difference in outgoing and incoming flow for each vertex v is the same in both graphs, P inherits the balance security of G, which from the above analysis is shown to preserve balance security. As such, we conclude that balance security of preserved during the aggregation phase of X-Transfer. □

Lemma 5. *Balance security is satisfied in the execution phase, i.e., no rational party can gain a larger utility from deviating from the protocol.*

We will prove this by defining an underlying strategic game corresponding to the execution phase and showing that the strategy that follows Algorithm 1 is a subgame perfect equilibrium. In the following paragraphs, we define the requisite game theoretic and equilibrium concepts as well as a formal description of the game induced by the execution phase.

Notation and terminology. For a set S, we use $\Delta(S)$ to denote the set of all probability distributions on S. For a distribution $\alpha \in S$, we use $x \leftarrow \alpha$ to denote sampling an element x from S according to the distribution α. We use the notation $x \leftarrow S$ to denote that the element x is sampled from S uniformly at random. For an n-dimensional vector $v \in S^n$, we use the indexing notion v_{-i} to denote all elements in v except for the ith element.

Strategic games and Nash equilibrium. Let $\Gamma = (N, (A_i), (u_i))$ be an N player simultaneous strategic game where A_i is a finite set of actions for each player $i \in [N]$ and denote by $A := A_1 \times \cdots \times A_N$ the set of action profiles. The utility function of each player i, $u_i : A \to \mathbb{R}$, gives the payoff player i gets when an action profile $a \in A$ is played. A *strategy* $\sigma_i \in \Delta(A_i)$ of a player $i \in [N]$ is a distribution over all possible actions of the player. We say player i's strategy is *pure* if it a Dirac distribution over A_i.

Definition 5. *(Nash equilibrium). A Nash equilibrium (NE) of Γ is a product distribution $\alpha \in \times_{j \in [N]} \Delta(A_j)$ such that for every player $i \in [N]$ and for all a_i' in A_i,*

$$\mathbb{E}_{a \leftarrow \alpha}[u_i(a)] \geq \mathbb{E}_{a \leftarrow \alpha}[u_i(a_i', a_{-i})]$$

Extensive form games and subgame perfect equilibria. Games that span multiple rounds (where players' actions arrive sequentially) are modelled as extensive-form games. An extensive-form game can be represented as a finite game tree where for every non-leaf vertex x there are functions that describe the player that moves at x, the set of all possible actions at x, and for each action a, the child node that leads from x given a. Moreover, in the imperfect information setting, or a mixed setting where players can make simultaneous moves, all player vertices are further partitioned into information sets I which captures the idea that the total information about the game given to a player that makes a move at any vertex $x \in I$ is the same as making a move from any other vertex $x' \in I$. Thus, the player is effectively rendered uncertain of their precise location in the game tree modulo the vertices in their information set. A path from the root of the game tree to a leaf vertex corresponds to a *game play* in Γ which is a sequence of player moves made by the players in the game. Each leaf node is assigned a payoff vector which represents the payoff of each player if the game terminates at this leaf. A subgame of an extensive-form game corresponds to a subtree rooted at any non-leaf vertex x that belongs to its own information set I, i.e., there are no other vertices that are in I except for x. A strategy profile is a *subgame perfect equilibrium* (SPE) if it is a Nash equilibrium for all subgames in the extensive-form game.

Game induced by the execution phase of X-Transfer. The execution phase of X-Transfer induces a 5-stage extensive form game Γ. The first stage of Γ corresponds to secret generation and broadcast and only involves a move by h_k. Here, the set of actions h_k can make would be to either sample and broadcast the hashed secret (for readability, we merge both actions into a single action called *broadcast*), or to not broadcast the hashed secret. Playing the action broadcast would advance Γ into the next stage, while choosing to play $\neg broadcast$ will abort the process and terminate Γ.

In the second stage of Γ, all users (clients and hubs) proceed to set up Thora in their respective PCNs. Again for readability purposes, we merge both Thora pre-setup and setup into a single stage in Γ, and we simplify the actions of all users without the secret (i.e., all users except for h_k) to $\{run, \neg run\}$ where playing run in Γ corresponds to participating in the setup of Thora in X-Transfer, and playing $\neg run$ corresponds to dropping offline/aborting the protocol at this stage. For h_k, the actions available are $\{run, \neg run, leak \wedge run, leak \wedge \neg run\}$, where $run, \neg run$ are defined as above but $leak$ refers to h_k leaking the secret to one or more users at this stage. Note that since X-Transfer does not impose any setup order in this stage and hubs are allowed to independently setup Thora in their PCNs, this stage of Γ involves simultaneous play.

The third stage of Γ corresponds to the Thora confirmation in Algorithm 1. Here, the simplified set of actions for all users except h_k are $\{confirm, \neg confirm\}$ and for h_k, $\{confirm, \neg confirm, leak \wedge confirm, leak \wedge \neg confirm\}$. Playing $confirm$ in Γ corresponds to adhering with the Thora confirmation protocol. Playing $\neg confirm$ corresponds to refusing the follow through with Thora confirmation and aborting the protocol. Finally, the action $leak$ is

only available to h_k and corresponds to leaking the secret to one or more users at this stage. As the move order is again inconsequential in this stage, we assume all users move simultaneously.

The fourth stage of Γ corresponds to the hubs setting up and confirming a Thora among themselves. The simplified set of actions for all hubs except h_k would be $\{run, \neg run\}$ where run corresponds to participating in the setup and confirmation of Thora in X-Transfer, and playing $\neg run$ corresponds to aborting the protocol. For h_k, the action set consists of $\{run, \neg run, leak \wedge run, leak \wedge \neg run\}$ where $leak$ again corresponds to leaking the secret to some subset of users. Note that in this stage only hubs can make moves and again since the order of moves is inconsequential, we assume simultaneous moves.

The last stage of Γ corresponds to the actual execution of X-Transfer, and begins with the move of h_k. Here, the set of actions for h_k are $\{reveal, \neg reveal, leak\}$ where $reveal$ corresponds to revealing the secret to all hubs, $\neg reveal$ corresponds to not revealing the secret and $leak$ corresponds to leaking the secret to a subset of hubs. Once h_k has made a move, other other hubs play an action from the set $\{execute, \neg execute\}$, where $execute$ and $\neg execute$ denote executing and not executing Thora in their PCNs respectively. Finally, each sender has to play an action from the set $\{send, \neg send\}$ where $send$ and $\neg send$ denote sending the revealed secret to their corresponding recipient and not sending the secret respectively.

Player utilities. At this point we formally describe the utilities of each player in Γ. We first reiterate that hubs, being users that do not send or receive money, should not have any change in their balance at the end of the protocol. While we leave the precise specification of the hubs' utility function open, we make two important assumptions about their utility function. First, the utility of hubs is strictly monotone in any change in their balance (i.e., any positive change in balance increases their utility, while a negative change decreases their utility). Second, given two strategies, one honest (following the stipulated protocol of X-Transfer) and one adversarial, with both strategies giving the hubs the same expected utility, we assume that the hubs would have a preference for the honest strategy given their main function as facilitators of funds transfers. In practice, this is easily enforced by giving fees to hubs for each successful execution of X-Transfer. Next, we again assume the utility function for each client (sender or recipient) is also monotone in their balance changes. However, we also assume that senders, once they sent out funds, would achieve a higher expected utility under strategies that ensure their intended recipients receive their funds. This is also easily enforced in practice by reputation systems and the existence of a legal/penalty system that allow recipients to demand their intended payment from senders the moment some goods exchange hands (e.g., the recipients sent out some goods that the sender bought and is awaiting the funds transfer from the sender).

Honest strategy. Let us denote by σ the following pure strategy:

- In stage 1 of Γ, h_k plays action *broadcast*.

- In stage 2 of Γ, all players play action run.
- In stage 3 of Γ, all players play action $confirm$.
- In stage 4 of Γ, all hubs play action run.
- In stage 5 of Γ, h_k plays action $reveal$. Then all hubs play action $execute$, and senders in their PCNs play $send$.

We now show that σ is a strict SPE in Γ.

Lemma 6. *σ is a strict SPE in Γ.*

Proof. We will proceed by backwards induction. In stage 5 of Γ, observe that in all subgames which require senders to play an action from the set $\{send, \neg send\}$, the senders would strictly be better off playing $send$. This is because these senders are already in PCNs that have executed their Thoras and so they sent out their funds. From the description of the senders' utilities as specified above, for all sender i, $u_i(send) > u_i(\neg send)$. Thus the pure strategy that plays $send$ is a strict NE in this subgame.

Now we go up a level in the game tree and look at all subgames that stem from playing $\neg execute$ (i.e., some hub $h_j, j \neq k$ does not execute Thora in their PCN. From Lemma 3 and the fact that $send$ is the NE in the subgame one level deeper than this subgame, we know that the payoffs from playing $\neg execute$ would either be the same as playing $execute$ (in the case where h_j belongs to a PCN where some other sender notified h_j's recipient which then executes Thora), or lower in the case where h_j has already sent out money to some other hub but has not yet received funds from the execution of Thora. Either way, as $u_j(\neg execute) \leq u_j(execute)$, the pure strategy that plays $execute$ is the strict NE in this subgame.

We then go up to the next level of the game tree. Here, the utility of h_k playing $\neg reveal$ is simply 0 since no one would be able to execute the protocol without the secret and no money changes hands. The utility of h_k when playing $leak$ is also 0, since a single hub knowing the secret is sufficient to execute Thora among the hubs, and we eliminated all subgames at the deeper level in which $h_j, j \neq k$ does not execute. Finally, although h_k does not gain any funds when playing $reveal$, with the preference for honest behaviour as specified in our utility function, the utility of h_k would be larger under the pure strategy of $reveal$ compared to any other strategy. Thus, the pure strategy that plays $reveal$ is the strict NE in this subgame and we can eliminate all subgames at this level that do not begin with $reveal$.

	run	$\neg run$
run	(ϵ, ϵ)	$(0, 0)$
$\neg run$	$(0, 0)$	$(0, 0)$
$run \wedge leak$	$(0, 0)$	$(-\alpha, \beta)$
$\neg run \wedge leak$	$(-\alpha, \beta)$	$(-\alpha, \beta)$

Table 1. Payoff matrix for the subgames at round 4 of Γ.

Now we proceed to stage 4 of Γ. Table 1 describes the payoff matrix for all subgames in the reduced game at this stage. The payoffs of the row player (i.e.,

h_k) is denoted by the first element in the payoff pair, while the payoff of the column player (all other hubs) is denoted by the second element. We use $\epsilon > 0$ to denote the payoff of the hubs when following the honest strategy. Note that the payoff where h_k plays $run \wedge leak$ while other hubs play run is 0 for all hubs because this technically advances the game to the next stage where hubs execute their Thora. Nevertheless, it is not the honest strategy thus the expected payoff is not ϵ for all hubs. We also use $\alpha > 0$ to denote the expected loss of funds that happens when h_k leaks the secret but the hub Thora protocol is not setup. Recall that at this stage all PCN Thoras apart from the hub Thoras are set up and that h_k is the last hub on the aggregated funds transfer path among hubs. Thus, if the hub Thora does not execute but some other Thoras execute (due to leaking the secret) h_k can only stand to lose funds (in the case some other PCN executes their Thora and a sender sends their secret to a recipient in G_k) or at best remain with balance unchanged (if no PCN Thoras execute or there is no execution that triggers G_k to execute). Since the first case happens with non-zero probability, $\alpha > 0$. In each of the leak cases, we use β to denote the expected return of the other hubs. From Table 1, it is clear from that any strategy of h_k that plays $leak$ is dominated by any strategy that does not play $leak$, thus we can eliminate all strategies of h_k involving $leak$ (last 2 rows of Table 1. From the remaining strategies, it is clear that the pure strategy where all hubs play run dominates all other strategies, thus the pure strategy where all hubs play run is the strict NE in this subgame.

	$confirm$	$\neg confirm$
$confirm$	(ϵ, ϵ)	$(0, 0)$
$\neg confirm$	$(0, 0)$	$(0, 0)$
$confirm \wedge leak$	$(-\alpha, \beta)$	$(-\alpha, 0)$
$\neg confirm \wedge leak$	$(0, \beta)$	$(0, 0)$

Table 2. Payoff matrix for the subgames at round 3 of Γ.

We proceed to the subgames at stage 3 of Γ. Table 2 describes the payoff matrix for all subgames in the reduced game at this stage. Note that when h_k does not confirm the Thora at G_k but leaks the secret to some subset of users, the expected payoff of h_k is always 0 as h_k cannot stand to gain or lose funds since the G_k Thora is void and the hub Thora has not been set up yet. We also note that by playing $confirm \wedge leak$ while other hubs play $\neg confirm$, h_k can only be worse off in the case where h_k leaks the secret to a recipient in G_k who would then gladly execute the Thora in G_k to pull funds. As such, a similar analysis to the subgames at stage 4 allows us to eliminate all strategies of h_k that involves $leak$, and from the remaining strategies the pure strategy where all hubs play $confirm$ dominates all other strategies, and forms the strict NE in this subgame.

The analysis of the subgames at stage 2 of Γ proceeds similarly with Table 3 describing the payoffs corresponding to all subgames in the reduced game at this stage. In the case where h_k leaks the secret during this stage and any subset of parties (including h_k) runs the protocol, we use α to denote the expected loss

	run	$\neg run$
run	(ϵ, ϵ)	$(0, 0)$
$\neg run$	$(0, 0)$	$(0, 0)$
$run \wedge leak$	$(-\alpha, \beta)$	$(-\alpha, 0)$
$\neg run \wedge leak$	$(0, \beta)$	$(0, 0)$

Table 3. Payoff matrix for the subgames at round 2 of Γ.

that h_k could encounter. In the case where the other hubs do not run the Thora setup but only h_k does, leaking the secret results in h_k losing nothing at best (if the subset of users h_k leaks the secret to does not trigger the Thora in G_k), or loses funds at worst (if Thora in G_k is triggered, as the hub Thoras will not be set up and h_k being the last hub in the aggregated path among the hubs can only stand to gain funds from setting up and executing the hub Thora). Even if the other hubs run the Thora setup and h_k leaks and runs the setup as well, since the Thora among the hubs is set up after and could still possibly be aborted, leaking the secret at this stage opens up the possibility that G_k's Thora is triggered without the Thora among the hubs setting up, which again causes h_k to lose funds. Thus, since these events occur with non-zero probability, the expected loss to h_k in when leaking the secret and running the protocol is $\alpha > 0$. As such, similar to the analysis of the subgames at stage 3 of Γ, we can eliminate all strategies that involve $leak$. From the remaining strategies, it is clear that the strategy where all hubs play run is the strict NE in this subgame.

Finally, we go to the first stage of Γ. Having eliminated all dominated strategies, the expected payoff of h_k when choosing the action $broadcast$ is $\epsilon > 0$ while the expected payoff of h_k when choosing the action $\neg broadcast$ is 0. This comes from the utility function definition and preference for the honest strategy. Thus it is clear that the pure strategy that always picks $broadcast$ is the strict NE of this subgame.

Therefore, σ is a strict SPE in Γ. □

Remark 1. (Strict SPE and our assumptions on utilities.) Here we make an important observation that relaxing the utility function such as not to give a preference for following the honest strategy as stipulated by X-Transfer would still result in σ being an SPE of Γ. However, σ would not be a *strict SPE*. In particular, relaxing the utility function to not have any preference for honest behaviour could lead to other SPEs with, e.g., senders that do not send the revealed secret to their recipients, or hubs that are in a neutral net flow position but do not execute Thora. Unfortunately, these strategies break the balance security of X-Transfer, and so these assumptions on utilities and preferences are required to enforce the honest strategy and hence balance security. We stress that there might be some wiggle room with regards to these utilities and preferences that could lead to other balance security preserving SPEs (e.g., aborting before any Thora is confirmed would also preserve balance security). As such, we leave the detailed study of necessary conditions of the utility functions and preferences to ensure balance security to further work.

We now have the requisite ingredients to prove Lemma 5.

Proof. (of Lemma 5) From Lemma 6, the honest strategy σ as defined above is the SPE in the game Γ induced by the execution phase of X-Transfer. Now all that remains is to show that σ preserves balance security. In playing σ, the only funds that changes hands are the amounts outputted from the aggregation phase of X-Transfer. Thus, balance security of σ follows from Lemma 4 which shows that the aggregation phase of X-Transfer preserves balance security. □

Theorem 1. *Our protocol satisfies balance security as in Definition 1.*

Proof. Follows directly from Lemmas 4 and 5. □

H.2 Optimality

The analysis of the optimality of X-Transfer is restricted to the aggregation phase which solves the optimization problem of maximising the volume of transactions while minimising the total cross-PCN transfer fees. We first state and prove that the DAG solution G in the aggregation phase incurs minimal fees.

Theorem 2. *The DAG solution G incurs minimal fees as per Definition 2.*

Proof. Follows directly from the statement of the objective of the optimization problem (stated in Equation 1), the correctness of the optimization oracle, and Lemma 1. □

However, the path solution P required by X-Transfer incurs larger fees compared to the optimal solution, as during the DAG to path conversion process (refer to Algorithm 3) a sender s and receiver r that are directly connected in G might now be separated by middle nodes in P. Hence, the funds going from s to r might be transported over multiple edges in P instead of just a single edge in G. In the worst case (depicted in Figure 8), X-Transfer would incur $O(kM)$ more fees compared to the optimal solution, where M is the sum of all outgoing edge weights in G.

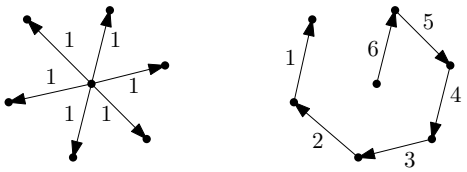

Fig. 8. Comparison of fees incurred in the path solution to the optimal DAG solution.

We note, however, that Algorithm 3 employs a few heuristics to minimize the amount of excess funds transported. Firstly, sending vertices are connected in ascending order, which prevents the largest sending amount to be transported more than once across all sending vertices. Secondly, transporting the funds to the receiving hubs in descending order of total inflow allows the largest amount of funds to be absorbed by the first receiving hub, which again reduces the

excess amount of funds transported over the course of distributing the funds to all receiving hubs. We leave developing more optimal algorithms and better heuristics for future work.

H.3 Computational feasibility

Theorem 3. *The optimization problem in the aggregation phase of* X-Transfer *can be solved in time* $O(n(k\Delta)^{k^2} + k\log(k))$ *where* Δ *is the upper bound on the demand of each user,* k *is the number of hubs, and* n *the number of transactions in the input transaction list.*

Proof. The complexity of solving the ILP as stated in Equation (1) in Section 4.2 is $O(n(k\Delta)^{k^2})$ from the result of [48] (Theorem 1). The remainder of the optimization problem is computing the optimal way to connect the flows between the hubs. In the greedy algorithm we present, the main complexity bottleneck is sorting the list of hubs in terms of inflow and outflow, which gives the $k\log(k)$ term. □

H.4 Privacy

Theorem 4. X-Transfer *satisfies privacy as defined in Definition 4 assuming users running the MPC protocol satisfy the trust assumptions required by the underlying MPC protocol.*

Proof. We will show both phases of our protocol satisfies the definition of privacy as defined in Definition 4.

We first analyse the aggregation phase and consider the indistinguishability game as defined in Definition 4. Suppose there is a TTP that takes in the challenge input $\mathcal{T}^b$ for $b \in \{0, 1\}$, computes the aggregation, and outputs to each node in V' the flow on their incident channels. We further suppose the aggregation output satisfied all conditions as specified in Definition 4, primarily that the flow output restricted to $G^{\mathcal{A}}$ should be the same regardless of whether $\mathcal{T}^0$ or $\mathcal{T}^1$ is given as input. Given that this is so, each corrupted user cannot distinguish the flow output regardless of whether $\mathcal{T}^0$ or $\mathcal{T}^1$ is the input.

Now replace the TTP with an MPC protocol that computes the same functionality. Since we assume that the delegates running the MPC satisfy the trust assumptions of the underlying MPC protocol, we get the same privacy guarantees as in the case with the TTP.

The execution phase of our protocol involves using Thora and HTLCs to update channels. The usage of both Thora and HTLCs in our protocol do not leak the payment values along each channel to any other party apart from the channel owners, and thus does not violate privacy as in Definition 4. Furthermore, the set of involved users is restricted to be the same whether the protocol is run on $\mathcal{T}^0$ or $\mathcal{T}^1$, thus the view of the adversary in the execution phase is also indistinguishable, hence private. □

I Experimental Evaluation

We implemented the aggregation phase of X-Transfer and performed a (proof of concept) experimental evaluation towards answering the research question:

Does X-Transfer *increase the total volume of successful transactions?*

Baseline. To answer this research question, we compare the performance of our implementation of X-Transfer against a baseline which does not implement any transaction aggregation. In this baseline, transactions are processed sequentially and at each step transactions are checked to see if they can be executed.

Setup and methodology. We ran our experimental evaluation [7] with 5 PCNs and 1000 clients per PCN. We created bidirectional channels between each client and hub in all PCNs and randomly sampled the client-to-hub channel capacities from the list of broadcasted channel capacities contained in a recent (Aug 2023) snapshot of the Lightning Network [20]. We used Python 3 for our implementation and will make the code available upon publication. All experiments were run on a laptop with 8 cores (Intel Core i7) and 16GB RAM, running Ubuntu 18.04.1 LTS. We used Gurobi [33] for solving the ILP.

Our experimental parameter sweep is the size of input transactions. We first fix the target number of input transactions per client to 10. Let c denote the capacity utilization of each channel, which we define as the ratio of the sum of the volume of all transactions originating from a client over the client-to-hub channel capacity. Note that c is fixed for all channels. For each client, we then randomly sample 10 transactions going from the client to a recipient chosen uniformly at random across all PCNs, with the restriction that the sum of the volume of the sampled transactions divided by the client-to-hub channel capacity equals c. We then run a sweep over $c \in \{0.5, 1, 2, 4\}$. Figure 9 plots the actual distribution of the sizes of the sampled transactions from our procedure. The y-axis depicts the transaction amount (in satoshis) and the x-axis displays the transactions ordered from smallest to largest. For instance, the element $(1, T)$ denotes that the smallest transaction is of size T. In our sampling procedure, transaction sizes are sampled in proportion to the capacity of the client-to-hub channels. Thus, as expected, most transactions are pretty small as most client-to-hub channels have small capacities.

The statistics we compute are the success volume ratio and the run time of X-Transfer. The success volume ratio is defined as the total volume of successful transactions over the total volume of transactions in the input list. The mean and median of each statistic are computed over 10 runs of both X-Transfer and the baseline for each parameter setting.

Results. As can be seen from Figure 10, the average success volume ratio of both X-Transfer and the baseline decreases as expected as the average size of each transaction (governed by c) increases. However, it is clear that the average success volume ratio of X-Transfer is *consistently and significantly larger* (at least 2 times) than the baseline. This shows that X-Transfer gives a clear boost in transaction throughput over the baseline of no aggregation.

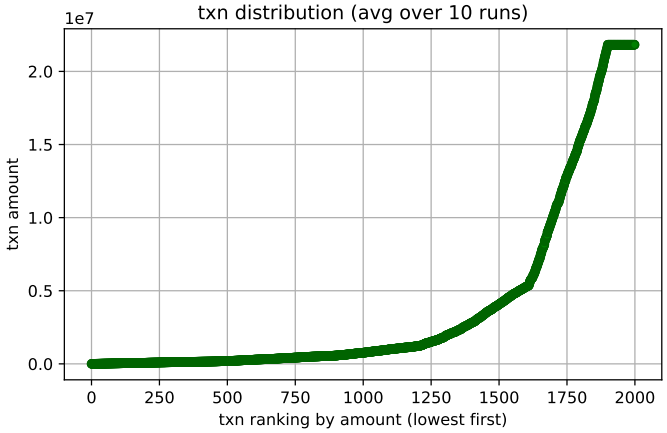

Fig. 9. Distribution of transaction sizes.

Figure 11 shows the mean and median run time of X-Transfer, which includes both solving the ILP as well as computing the optimal hub flows. We note that the mean and median generally correspond closely, with the exception in the case of $c = 4$. This could be explained by variance in the Gurobi solver for the ILP when given larger inputs. Nevertheless, we observe the median runtime for all parameter settings remains around 3 seconds, which indicates that the aggregation segment of X-Transfer incurs only a minimal computational overhead and this overhead also scales well with the size of input transactions.

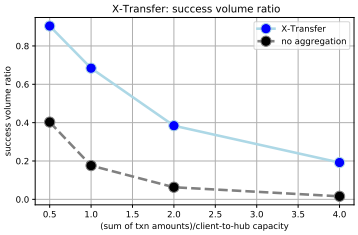

Fig. 10. Mean success volume ratio

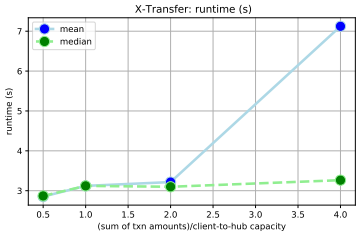

Fig. 11. Runtime of X-Transfer

Evaluation discussion. Next we discuss two directions to optimize the evaluation that could be interesting to explore for future work.

First, we note the tradeoff between efficiency and optimality in the solving of the ILP: we can handle more transactions and larger transaction volume which boosts the efficiency of X-Transfer if we sacrifice the optimality of the ILP solution. This can be done by editing the Gurobi solver we use to solve the ILP to terminate sooner with a nearly optimal solution.

Second, our current implementation of transaction aggregation as specified by X-Transfer works best in the one-shot setting as all transactions are given

equal weight. Indeed the optimization problem as specified in Eq 1 can be rewritten as max $\sum_{i=1}^{n} w_i|t_i|x_i$, where $w_i \in [0,1]$ are some prespecified weights given to each transaction. In X-Transfer we assign all transactions to have equal weight (we can think of w_i to be 1 for all i in Eq 1). However, when X-Transfer is run consistently in reality sometimes it makes sense to prioritize certain transactions over others (for instance, transactions that keep failing to execute in previous rounds of X-Transfer). This can be done by giving these transactions a larger weight in the objective.