SoK: Optimization Tools for the Hydra Multiplayer Channel

Maxim Jourenko[1] and Mario Larangeira[1,2]

[1] Institute of Science Tokyo, Japan
[2] Input Output Global

Abstract. Blockchain technology allows parties to use a consensus protocol to add transactions and data to a decentralized ledger that are immutable. However, the amount of transactions that can be processed within a consensus protocol and thus added to the ledger within a given time frame is limited. This is a scalability issue. Layer-2 protocols are a class of approaches that process transactions off-chain, i.e. without involvement of the consensus protocol, but allow usage of the consensus protocol as a backup for security. This potentially reduces the processing of $n \in \mathbb{N}$ transactions to only $\mathcal{O}(1)$ transactions being committed to the ledger. The basic building blocks of Layer 2 protocols are Payment Channels, whereas newer protocols as Hydra (FC'21) allow for multi-party channels that are not limited to payments but can execute smart contracts, i.e. program code. There has been substantial research proposing tools and improvements to Layer 2 protocols, but most assume the original two party Payment Channel design. This makes it unclear to what degree existing techniques can be utilized with Hydra style multi-party channels. This work (1) presents a survey of such such existing two party channel protocols and tools built on top of them. We use a didactic approach with a focus on explaining their core ideas. (2) This work identifies uses cases for which novel protocols for Hydra should be developed. (3) This work analyzes open problems for Hydra, lists relevant related work and proposes which existing tools can be used to address them or serve as a starting point for further research.

Table of Contents

1 Introduction . . . 2
1.1 Structure of this Work . . . 2
2 Preliminaries . . . 2
2.1 Notation . . . 2
2.2 Channels . . . 2
3 Existing Tools . . . 3
3.1 Channel Networks . . . 3
3.2 Virtual Channels . . . 7
3.3 Routing . . . 8
3.4 Security of Offline Parties . . . 11
3.5 Re-Balancing Channels . . . 15
3.6 Payment Privacy . . . 15
4 Hydra Settings . . . 18
5 Evaluation of Potential Tools for Hydra . . . 18
5.1 Offline Parties . . . 18
5.2 Weaker Security Assumptions . . . 19
5.3 Anonymity / Privacy . . . 21
5.4 Summary . . . 23
6 Evaluation of Potential Tools for Payment Channel Networks over Hydra . . . 23
6.1 Network Payments . . . 23
6.2 Routing . . . 24
6.3 Re-Balancing Channels . . . 25
6.4 Privacy . . . 26
6.5 Summary . . . 26
7 Evaluation of Potential Tools for State Machines across a Hydra Network . . . 26
7.1 Channel Stability and Watchtower . . . 27
7.2 Liquidity . . . 27
8 Conclusion . . . 28

1 Introduction

Blockchain technology enables byzantine consensus that is scalable to a vast amount of parties. As a tradeoff and in comparison to classic protocols, blockchains have limited capacity for the amount of data and transactions that can be added to the blockchain within a given time frame. *Offchain*, also called *Layer 2*, protocols propose a solution: Parties can collaborate with each other to create offchain *channels*, a protocol where they can perform transactions among themselves, however, without having to add all these transactions to the blockchain. Instead, these transactions are *enforceable* on the blockchain, i.e. in case of a dispute (or tear down of the channel) a summary of the transactions performed offchain can be committed to the ledger. This improves scalability: Performing $n \in \mathbb{N}$ transactions offchain results in at most $\mathcal{O}(1)$ transactions being committed to the ledger itself.

Hydra [17] is a channel construction designed for the Cardano blockchain. In contrast to existing protocols, Hydra is *multi-party*, allows for transactions with the same expressiveness as on the regular blockchain, and operates using the Extended Unspent Transaction Output (EUTxO) ledger model. As a result, with a few exceptions, existing Layer 2 protocols that utilize channel constructions are not necessarily directly usable with Hydra or do not fully utilize Hydra's capabilities. Moreover, re-contextualizing relevant problem statements in the area of Layer 2 protocols to Hydra can yield novel insights and protocols.

In this work we review existing problems in the area of Layer 2 protocols, give an overview and compare existing protocols that propose solutions to these problems. We introduce three settings or use cases for which we re-contextualize existing protocols and use it to derive open problems for Hydra. For these open problems we propose existing related work that can be used as a starting point for literature research.

1.1 Structure of this Work

In the following, first we do a survey of existing tools by their use case in Section 3. Then we introduce three settings in which Hydra can be deployed in Section 4 and evaluate existing tools in the context of each setting in Sections 5 - 7.

2 Preliminaries

2.1 Notation

In this work we compare different protocols and point out whether a certain work fulfills some property or it does not. If not noted otherwise, ● denotes that a protocol fulfills a given property and ○ denotes that it does not. Moreover, we denote ◐ in case it is not straightforward whether a protocol fulfills a given property and provide additional information in such a case.

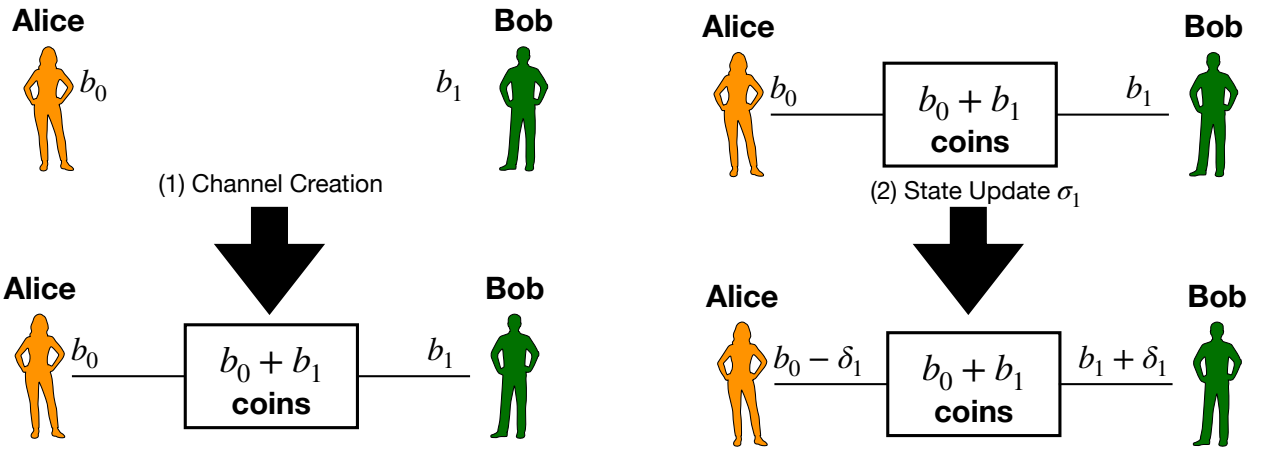
2.2 Channels

We consider a distributed ledger, e.g. a blockchain, as *Layer 1*. A protocol can make use of Layer 1 to create an *offchain*, or *Layer 2*, channel that facilitates interaction, in form of transactions, between all participating parties. Setup, dispute and tear down of the channel can require transactions to be processed by Layer 1, but transactions between the parties do not incur transactions on Layer 1.

We will use Layer 2 and offchain interchangeably in the remainder of this work. The basic building blocks of Layer 2 are *channels*, structures set up on the blockchain that allow participating parties to interact with one another an arbitrary number of times and without having to further interact with the blockchain except in the case of dispute or tear-down of the channel. Transactions between these parties do not incur fees which has two benefits: (1) It enables Microtransactions, i.e. transactions with lower value than submission of a transaction on the ledger. (2) It enables fast finality, i.e. transactions are executed almost instantly through the execution of a payment sub-protocol between both parties and without having to wait for the ledger to confirm it. A potential use case for channels could be a prepaid card for a coffee shop. For instance, opening the channel equals issuing and paying into a prepaid card where one participant is the coffee shop run by Bob and one is the customer Alice. No transaction fees ensure that the price for a coffee does not exceed the price of a coffee paid in cash, and fast finality allows the customer to receive a coffee without unreasonable delay (1 hour in the case of Bitcoin).

An overview of a channel lifecycle is displayed in Figures 1 to 3. Channels maintain some state, e.g. how coins in the channel are distributed between participants in the case of payment channels. The parties process transactions by updating this channel state. Channels allow the latest state to be enforced on Layer 1. This is done through channel closure and can be done in two manners:

- *Unilaterally* by one channel participant which is subject to a *dispute* phase where any counterparty can dispute the channel closure and correct it, if the closing party attempted to close the channel in a state that does not represent the latest one.



(a) Channel Opening

(b) Channel State Update

Fig. 1: Lifecycle of a payment channel. Alice and Bob intend to open a channel where initially Alice holds b_0 and Bob holds b_1 coins. (1) Opening the channel locks $b_0 + b_1$ coins within that channel. (2) Both parties can perform a channel state update to update how the coins in the channel are distributed, e.g. a payment changes the balances by $\delta_1 \in \mathbb{Z}$. In this process, a party's balance of coins cannot be less than 0 at any point.

– *Collaboratively* by all channel participants, where all parties agree to close the channel and cooperate to close it according to the latest state which does not incur a dispute phase.

Dispute requires interaction with the ledger by sending a transaction. Having this transaction included within the ledger takes at most time $\Delta \in \mathbb{N}$ which is the ledger's *liveness* parameter. Lastly, we consider c to be the *capacity* of a channel where c is the total amount of coins available within the channel.

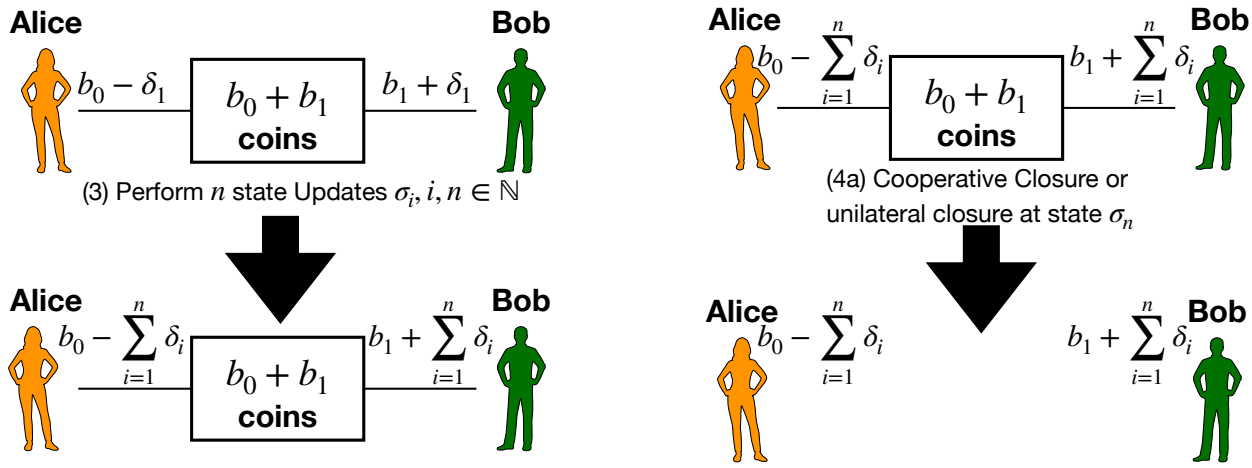
3 Existing Tools

In the following we list use cases for Layer 2 protocols that have been tackled by previous works and survey the tools proposed for these use cases. We stress that we survey protocols for channels, thus individual channel constructions, albeit being thematically close, are outside of the scope of this work.

3.1 Channel Networks

Early channel constructions, and prominently those used in the *Lightning Network* [58] (LN), are two-party channels, however, newer proposals include multi-party channels [21, 17] such as Hydra. Even in the multiparty case, the set of parties participating in the channel is fixed at the point of setup. This renders a channel's use cases quite limited. Protocols such as Hash Time Locked Contracts (HTLC) [58] enable payments across multiple hops of a network of payment channels: Imagine that, in the above example, the coffee shop run by Bob might have an additional channel with a pizza store run by Charlie. In that case, Alice can ask Bob to forward a payment to Charlie through their channel for a small fee, extending the use case of a channel from being a prepaid card to a payment network like the Visa credit network.

Hash Time Locked Contracts. Several protocols have been proposed to overcome the party limitation of channels by enabling transactions across a path of adjacent channels, whereas two channels are adjacent if they share at least one participant called the *intermediary* as illustrated in Figure 4. Doing so forms *networks* of channels. The *Lightning Network* implements HTLCs [58] to create the first large scale payment channel network on Bitcoin. The transactions that are enabled in the LN are limited to payments between two parties connected through the network as shown in Figure 4. In the following we consider a path of nodes $v_0, \ldots, v_{n_{\mathsf{path}}}, n_{\mathsf{path}} \in \mathbb{N}$ where v_0 is sender, v_n is receiver of a payment, $v_1, \ldots, v_{n-1}$ are *intermediaries* and γ_i is the channel between parties v_{i-1} and v_i. Each channel on a path from sender to receiver of a payment sets up a conditional payment: On revealing a secret x such that for a public y holds that $h(x) = y$ where h is a cryptographic hash function



(a) Channel Payments

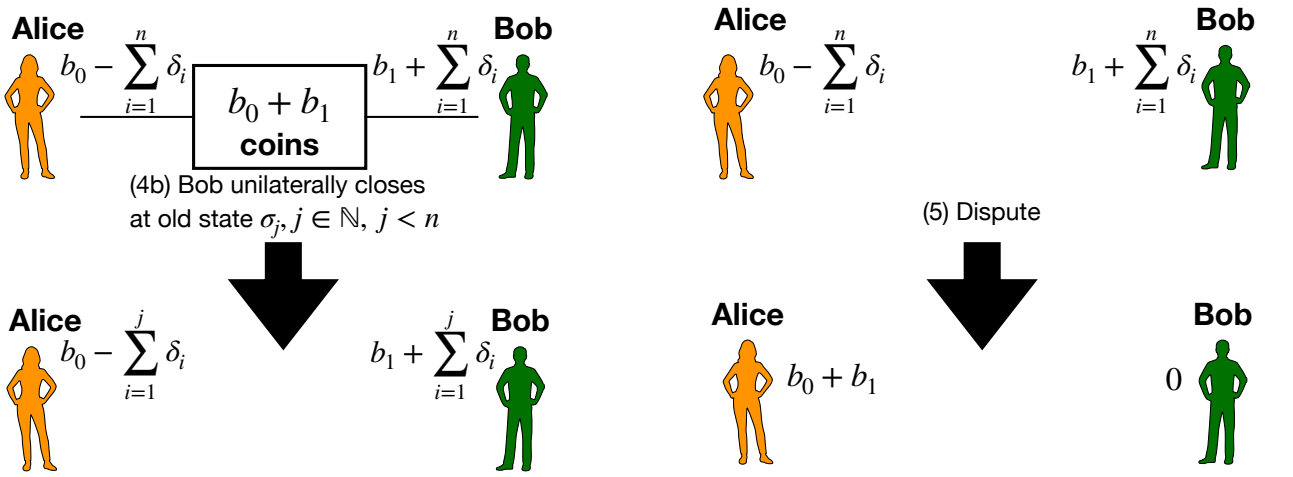
(b) Channel Closure

Fig. 2: Lifecycle of a payment channel. Alice and Bob intend to open a channel where initially Alice holds b_0 and Bob holds b_1 coins. (3) Alice and Bob perform $n \in \mathbb{N}$ further payments in the same manner as in Figure 1b through state updates $\delta_i \in \mathbb{Z}, i \in \mathbb{N}, 1 \leq i \leq n$. At some point, (4a) both parties cooperate to close the channel or one party closes the channel unilaterally at the latest state.

and $x, y \in \mathbb{N}$, then the payment on the channel is executed. As executing a payment on one channel, even if it is done offchain, reveals the secret x to the sender of the payment, that sender will be able to enforce the payment on the previous channel on the payment path to recover the coins they paid. In the example in Figure 4: If Charlie reveals x to Bob, then Charlie proves to Bob that he can enforce the payment on-chain, allowing them to resolve the payment offchain through a channels state update. In this process Bob learns x such that Bob can repeat this process in his channel with Alice and claim the coins he paid to Charlie from Alice. LN channels and HTLCs have been impactful such that some works on Layer 2 protocols attempt to extend or remain compatible with HTLCs and LN Channels [66, 44, 38, 9, 15].

Privacy. Reusing the same value y for each channel has issues: For one, it allows linking payments to another which can leak information on sender and receiver of a payment in Layer 2. Moreover, it gives rise to the wormhole attack [45]. Fulgor & Rayo [44] as well as AMHL [45] improve privacy of HTLCs by unlinking the individual payments of a HTLC payment that happens on each hop in a payment path: Instead of reusing the same public y in each conditional HTLC payment, these public values are replaced with uncorrelated values y_i for channel γ_i where only the intermediary who participates in channels γ_{i-1} and γ_i can derive y_{i-1} from y_i.

Vulnerabilities. To make HTLC payments secure, payments will timeout after some time t_i for channel γ_i where $t_{i-1} - t_i \geq \Delta$. Thus, in the case of HTLC payments the total locktime is in $\mathcal{O}(n)$. This leads to the proposal of several attacks for payment channels utilizing HTLCs, like congestion [50] and lockdown [53] attacks. AMHL [45] proposes to resolve the wormhole attack while improving privacy of network payments whereas Payment Trees [35] and Blitz [6] propose to mitigate Congestion and Lockdown attacks. Sprites [47] are akin to HTLCs but reduce the worst case duration of performing the payment from linear to constant in the number of hops of the path a given payment takes. To achieve this, they require use of Ethereum style on-chain smart contracts, thus are incompatible with less expressive blockchains such as Bitcoin. Payment Trees [35] proposes a means to facilitate payments in a payment channel network not based on HTLCs but instead on virtual channel constructions, reducing the worst case duration of the payment to be logarithmic in the path's length. Blitz [6] reduces the worst case duration of a payment to be independent of the payment path's length. This is achieved by having a payment abort when triggered through a dedicated transaction that acts as a global trigger for abort in all channels on a path. Otherwise, the payment is successful after an absolute timeout $T \in \mathbb{N}$ which is equal on all channels on a payment path. Note that choosing the same absolute timeout T on all channels leaks information about the payment's path. Thora [2] proposes a protocol that allows a set of payments to be executed atomically through the use of a separate *enable* transaction that acts as a trigger for these transactions which can be utilized to create payments across multiple hops of a PCN. Of note is that this enables payment



(a) Channel Incorrect Closure

(b) Channel Dispute

Fig. 3: Lifecycle of a payment channel. Alice and Bob intend to open a channel where initially Alice holds b_0 and Bob holds b_1 coins. At some point, (4b) in contrast to correct closure at the latest channel state as in Figure 2b one party, for instance Bob, closes the channel unilaterally at an older state, e.g. at the state where $j \in \mathbb{N}, j < n$ payments were performed. (5) This is then subject to the counterparties' dispute. The effect of dispute depends on the channel implementation: Some implementations then close the channel at the latest state (Hydra [17]), or punish Bob for misbehaving and awarding all coins in the channel to Alice (Lightning Network [58]).

paths with gaps, however, practically this would require that an intermediary used for the payment has multiple nodes within the network which reduces the feasibility of this approach. AMCU [25] is another protocol that attempts to enable payments that are not restricted to a path of the underlying PCN, however, it has been shown to be broken [35].

Timelock Bribing Attacks. Layer 2 protocols rely on a ledger's Δ-Liveness property to implement dispute mechanisms. This is not only the case for channel constructions, but holds true for most Layer 2 protocols, for instance like HTLC payments. Imagine two conflicting transactions τ_0 and τ_1. Here τ_0 can be included on the ledger immediately, however, τ_1 can only be included on the ledger after at least time Δ. The Δ-Liveness property guarantees that τ_0 is included in the ledger within at most time Δ, thus τ_1 can never be included in the ledger. Bribery Attacks are a distinct type of attack that aims to undermine timelock based dispute mechanisms. This attack manipulates the behavior of miners, i.e. nodes that perform a ledger's consensus mechanism in return for a fee reward. This attack is done by offering a *bribe* to blockchain miners, e.g. through mining fees, conditioned on the inclusion of τ_1 on the ledger. This incentivizes the miners to censor τ_0 and instead include τ_1 as this strategy maximizes their expected fee reward. On top of that, the *Bribe & Fork Attack* [11] is an extension to the Timelock Bribing Attack that combines the Bribing Attack with the threat of a fork to reduce the expected amount of coins required for a bribe, improving the feasibility of this type of attack. SUBORN Channels [12] propose a channel construction resilient to bribing attacks, whereas MAD-HTLC [66] proposes a variation of the HTLC protocol resistant to bribery attacks in the face of a rational adversary. MAD-HTLC utilizes the behavior of miners to enforce punishment of misbehavior. However, Wadhwa et al. [67] shows that MAD-HTLC is vulnerable to counter-bribing attacks and propose the He-HTLC protocol [67] to address this vulnerability.

Network Topologies. While PCNs are commonly created in ad-hoc fashion by all participants where each participant decides by itself which channels it creates and how these are connected to the PCN, Tang et al. [64] propose forming the PCN cooperatively into Ramanujan graphs of Lubotzky–Phillips–Sarnak. Such graphs have beneficial metrics that help with issues such as routing, centralization and the PCN's resilience to some attacks. The authors utilize a smart contract such that each party sets up multiple channels whereas some channel are chosen freely as before and others must be created according to the smart contract's rules to form the desired graph structure. As the coins used to create channels are split up among multiple channels, this results in channels with lower capacity, however, protocols for splitting payments [1, 24] could mitigate this.

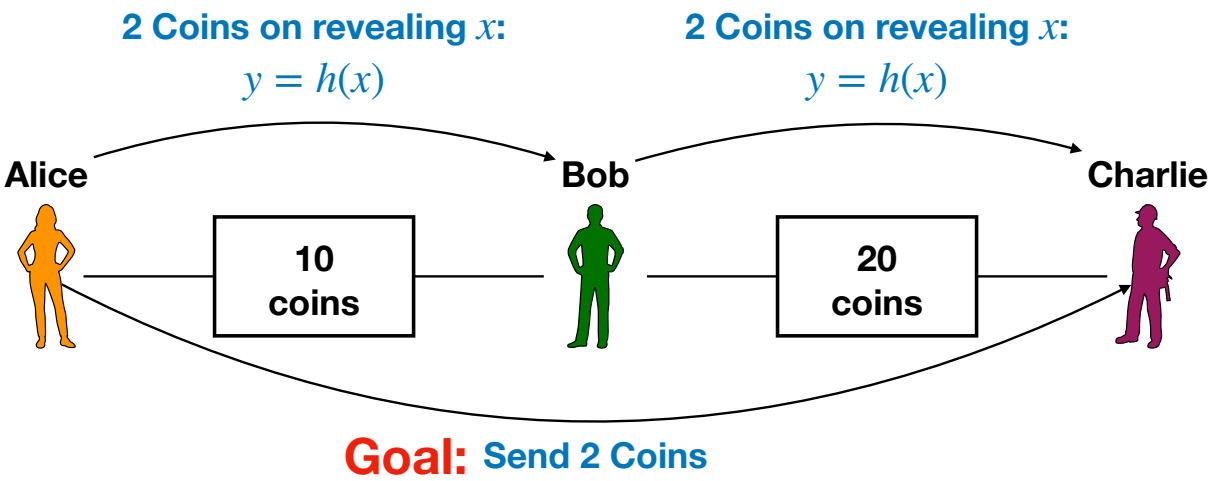
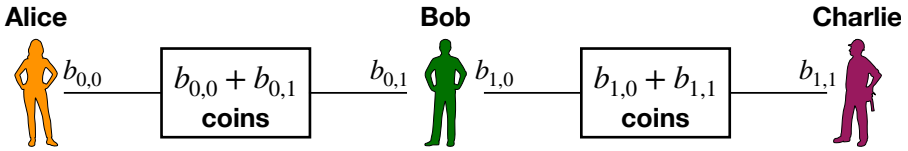


Fig. 4: Illustration of Hash Time Lock Contract based payments. Here, Alice and Bob as well as Bob and Charlie share a channel. Alice wants to send 2 coins to Charlie without sending a transaction to the ledger but reusing the existing channels. This can be done using HTLCs by making conditional payments on each hop on the path from Alice to Charlie: If Charlie reveals a secret x which is the pre-image of $y = h(x)$ where h is a cryptographic hash function, then he will receive 2 coins from Bob on their shared channel. Upon this Bob learns x and can in turn claim 2 coins from Alice on their shared channel. In summary, Alice pays 2 coins, Charlie receives 2 coins and Bob receives but spends 2 coins on each channel resulting in a coin difference of 0.

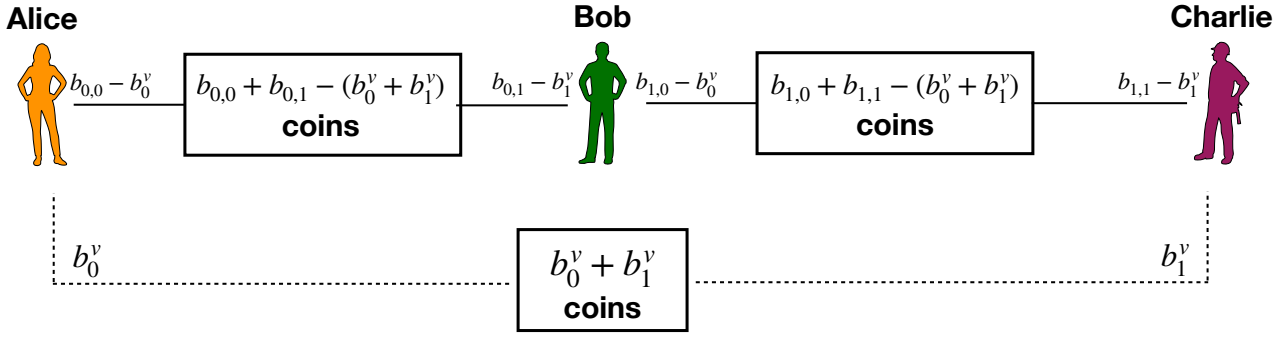
Name	Locktime	Not bound to Path	Privacy Preserving	Balance Security
Lightning and HTLCs [58]	$\mathcal{O}(n)$	◐	○	●
MAD-HTLC [66]	$\mathcal{O}(n)$	○	○	◐
He-HTLC[67]	$\mathcal{O}(n)$	○	○	◐
Payment Trees [35]	$\mathcal{O}(1)$	○	○	●
Sprites [47]	$\mathcal{O}(1)$	● [2]	●	●
AMHL [45]	$\mathcal{O}(n)$	○	●	●
AMCU [25]	$\mathcal{O}(1)$	●	●	○ [35]
Blitz [6]	$\mathcal{O}(1)$	○	○	●
Fulgor & Rayo [44]	$\mathcal{O}(n)$	○	●	●
Thora [2]	$\mathcal{O}(1)$	●	●	●
SyncPCN/PSyncPCN [26]	$\mathcal{O}(n)$	○	○	●

Table 1: Comparison of protocols that enable payments across a path of channels. Here, we compare the protocols for (1) their locktime, i.e. how long the coins used for a payment could be locked up and unavailable in terms of the length of the payment path n, (2) whether the payment must be along a path of the PCN or whether the payment path can have gaps, (3) whether the protocol implements any privacy preserving technique, e.g. to satisfy value or relationship privacy (Figure 13) and (4) whether any honest party is at the risk of losing coins if another party misbehaves. Of note is that the locktime of Payment Trees [35] is $\mathcal{O}(1)$ on average but its worst case locktime is $\mathcal{O}(\log\ n)$ which is for the payer and payee of a given payment. Moreover, while HTLCs require that there exists a path from payer to payee in the underlying PCN this is not strictly the case: The wormhole attack [45] exploits that HTLCs utilize the same condition y on a payment path. If a payment is routed through two colluding nodes in the network, they can make the payment skip the nodes in between them. MAD-HTLC [66] and He-HTLC [67] have balance security only in the presence of a rational adversary. AMCU [25] was shown to be broken [35].

State Channels. Interactions in a channel network are not limited to payments, but can also be extended to execution of smart contracts [23, 42, 33]. Aumayr et al. [4] utilize adaptor signatures to create channels where parties can interact with one another in the same manner as they can on the underlying blockchain. Notably, this allows not only for creation of channels on Bitcoin with scripting capability, but also can be extended to any circuit computed by two parties by executing a fair and secure two-party computation protocol. Relevant to this work is the Interhead [36] and Ad-Hoc Hydra Head constructions [33], which similarly retain the same expressiveness as the underlying ledger and allow for executing smart contracts between the members of adjacent Hydra Heads.



(a) Channel Network



(b) Virtual Channel on Channel Network

Fig. 5: Illustration of Virtual Channel constructions which is constructed across two adjacent channels within a PCN as displayed in Figure 5a. Alice, Bob and Charlie cooperate to create a virtual channel between Alice and Charlie where Alice holds b_0^v and Charlie holds b_1^v coins. To this end, Alice has b_0^v coins removed from her channel with Bob, Charlie has b_1^v coins removed from his channel with Bob and Bob has the same amount of coins removed from each channel as the virtual channel's counterparty, i.e. b_1^v coins in the channel with Alice and b_0^v coins in the channel with Charlie. Upon channel construction Alice and Charlie can perform payments without interacting with Bob, however, channel closure or dispute requires interaction with Bob.

Summary. Table 1 summarizes existing protocols that enable payments across PCNs. For one, we compare each protocol's locktime, i.e. up to how long payments can be locked up, as a high locktime of $\mathcal{O}(n)$ makes the protocol vulnerable to griefing, congestion [50] and lockdown [53] attacks. Several protocols work towards executing payments without requiring a path from payer to payee, however, finding use cases where this can be exploited requires specific PCN topologies [64] and might be impractical in common PCNs. Lastly, of note are SyncPCN/PSyncPCN [26] which allow for secure payments even in the case where a ledger's does not fulfill the Δ-liveness property, or in the case where it is broken through a miner bribing attack. In the miner bribing attack the miners are *bribed* to withhold the inclusion of a transaction on the ledger in favor of a conflicting transaction that pays more miner fees.

3.2 Virtual Channels

Offchain protocols can go beyond facilitating payments through a channel network as discussed in Section 3.1, and even allow to create additional *virtual* channels, i.e. without requiring transactions sent to the ledger except in case of a dispute. A virtual channel is illustrated in Figure 5: Virtual channels are constructed over two hops of an underlying channel network. Assuming three parties, Alice, Bob and Charlie where Alice and Bob as well as Bob and Charlie share a channel γ_0 and γ_1 respectively. Moreover, we consider the two hop path ψ_{ABC} that consists of the nodes Alice, Bob, Charlie in direction from Alice, through Bob to Charlie. We denote with $b_{i,j}, i, j \in \{0, 1\}$ the coins in channel γ_i where $j = 0$ for the first node in ψ_{ABC} and $j = 1$ for its successor in ψ_{ABC}. For example, Alice has $b_{0,0}$ coins in channel γ_0 whereas Bob has $b_{0,1}$ coins, thus the channel contains $b_{0,0} + b_{0,1}$ coins. Analogously Bob has $b_{1,0}$ coins in γ_1 and Charlie has $b_{1,1}$ coins. During setup of the virtual channel, all parties collaborate to open a virtual channel with $b_0^v + b_1^v$ coins. Then, b_0^v coins are removed from

Alice's balance in γ_0 and credited to her in the virtual channel, whereas and b_1^v coins are removed from Charlie's balance in γ_1 and credited to him within the virtual channel. Bob, as the intermediary, mirrors how many coins are removed by the counterparty within each channel, i.e. in γ_0 Bob removes b_1^v coins from his balance and in the channel γ_1 Bob removes b_0^v coins from his balance. After setup, Alice and Charlie can interact with one another without further interaction with Bob, who only acts during virtual channel closure or dispute. Only dispute results in transactions being sent to the Layer 1 ledger.

Virtual Channel Constructions. The first virtual channel constructions [22, 23] were proposed relying on Ethereum style smart contracts, however, further virtual channels are compatible with Bitcoin [34, 40, 5, 7] and instead relying on timelocks and UTxO ledger rules. This previous work was extended to allow creation of multi-party state channels [21] based on virtual channels. Further work [36] extended virtual channel creation to connect multi-party state channels [21, 17]. Of note are Ad-Hoc Ledger [33], which is not a virtual channel construction by itself, but allows execution of regular channel constructions within Layer 2, effectively turning them into virtual channels. While Hydra Heads are formalized within the UC framework [37], the formalization of Ad-Hoc Ledger are under development[3].

A property of most Virtual Channel constructions is their recursive nature: Virtual channels can not only be constructed on top of two regular channels, but also recursively on top of two virtual channels or a combination of a virtual and a regular channel. Donner [7] proposed the Domino Attack that abuses this property where creating a dispute in one virtual channel results in a dispute in all underlying virtual and non-virtual channels. Depending on the virtual channel construction, in the worst case this results in the closure of all underlying channels.

Summary. Table 2 displays an overview of virtual channel constructions. It distinguishes between constructions for payment and state channels, as well as whether the proposed protocol creates a multi-party virtual channel. Moreover, it displays whether a protocol requires a smart contract capable ledger and whether the proposed protocol was formalized within the UC framework. Of note are Multi-Party Virtual State Channels [21], that allows for an underlying network of pairwise state channels to be extended into having multi-party channels and, moreover, is formalized within the UC framework. The Interhead [36] and Ad-Hoc Ledger [33] constructions are designed to be compatible with Hydra.

Name	Type	Req. Smart Contracts	Multi-Party	UC
Bitcoin Compatible Virtual Channels [5]	Payment	○	○	●
Elmo [40]	Payment	○	○	●
Lightweight Virtual Payment Channels [34]	Payment	○	○	○
Donner [7]	Payment	○	○	●
General State Channels [23]	State	●	○	●
Interhead [36]	State	●	●	○
Ad-Hoc Ledger [33]	State	●	●	◐
Multi-Party Virtual State Channels [21]	State	●	●	●

Table 2: Overview of Virtual Channel Constructions displaying (1) whether they are virtual *payment* or *state*, i.e. allow execution of smart contracts, channels, (2) they require the underlying ledger to support smart contracts, (3) they allow construction of channels of an arbitrary amount of parties instead of exactly two, and lastly (4) whether their security is formalized within the UC framework. Here we note that the Ad-Hoc Ledger [33] has not been formalized within the UC framework at this point, however, such a formalization is currently under work (2nd March 2026).

3.3 Routing

A challenge of performing payments in a PCN is the discovery and selection of the path a payment should take. This is akin to routing network communication in mobile ad-hoc networks: Potential challenges are that nodes can randomly join the network or fail and leave and connections can be created or removed (e.g. by depleting the limited capacity of a channel) dynamically. Scalability is a common challenge in creation of routing protocols. Both the memory required by each node and the number of messages sent through the network to find a route should be logarithmic in the size of the network. Otherwise, there are two trivial solutions for routing protocols:

– If each node can store the whole network locally, then they can compute a route to any other node locally, e.g. using Dijkstra's algorithm.

[3] Date: 2026.03.2

– If the number of messages sent may be linear in the size of the network, a sender can flood the network with a route discovery message where a node upon receipt of such a message forwards it to all neighbors, recording the route the message took to reach it. As soon as the recipient of a payment receives such a message, they can use it to reconstruct the shortest path that a routing discovery message took to reach it.

Types of Nodes. In the context of blockchain PCNs, we can distinguish between two types of nodes within a PCN that are relevant to their routing capabilities:

– Full nodes, i.e. nodes that have a full replica of the blockchain stored locally, have knowledge of the full PCN topology, albeit they might not be aware of nodes being offline.
– Light nodes are nodes that do not have a full replica of the ledger stored locally. Devices with limited capabilities, like mobile phones, are most likely only capable to operate light nodes.

This further stresses the importance of scalable routing protocols: Protocols that require the full knowledge of a PCNs topology exclude light nodes from being able to perform payment routing. As the size of networks can become large, for example the LN has about 5,000 nodes and 13,000 channels (January 2026) [4], therefore scalable routing protocols are required [43, 61, 62, 59, 54, 31, 68, 52]. Performing a payment of $b \in \mathbb{N}$ coins in a network requires that there exists a path in the PCN where each channel is able to perform a payment of b coins. This limits the available channels that can be used for routing a payment, especially in the case where b is comparatively large. This can be addressed by protocols that allow payments to be split up and routed through multiple paths. These payments are then executed either atomically [1], i.e. either all or no payments are executed, or non-atomically [1, 24] where a payment can partially fail.

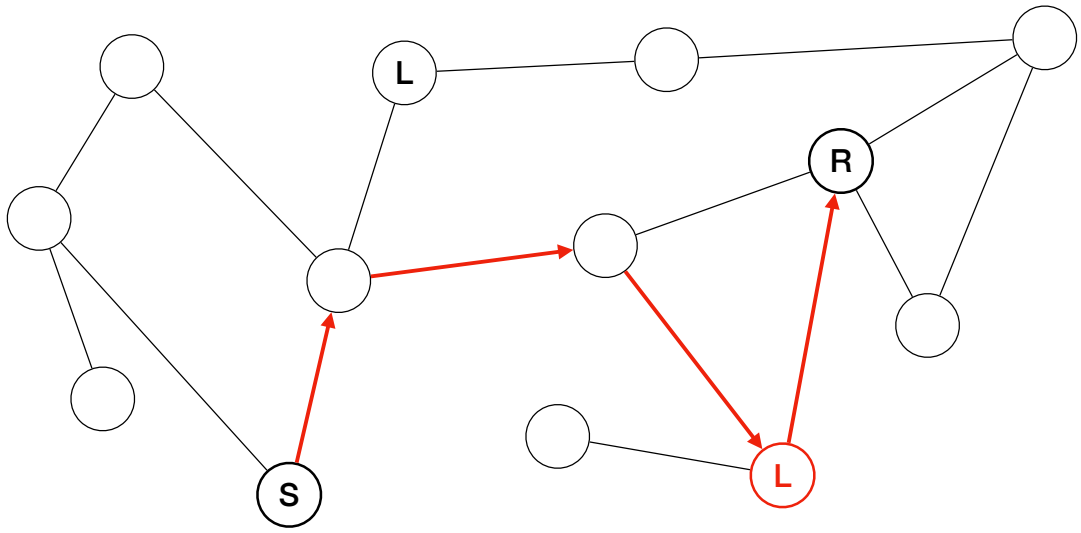


Fig. 6: Illustration of Landmark Routing from sender S and recipient R of a payment. L denotes landmarks which are designated nodes for which each node knows a route to. Assume $L_1, \ldots, L_{n_L}, n_L \in \mathbb{N}$ landmarks. For scalability n_L must be logarithmic in the size of the network. Thus, a route from S to R can be created by picking one landmark $L_i, 1 \leq i \leq n_L, i \in \mathbb{N}$ and concatenating the known route from S to L and the known route from L to R. Commonly, nodes pick a landmark for which the resulting route is shortest. In the above case this is the lower landmark with the resulting route indicated in red.

[4] https://1ml.com/statistics

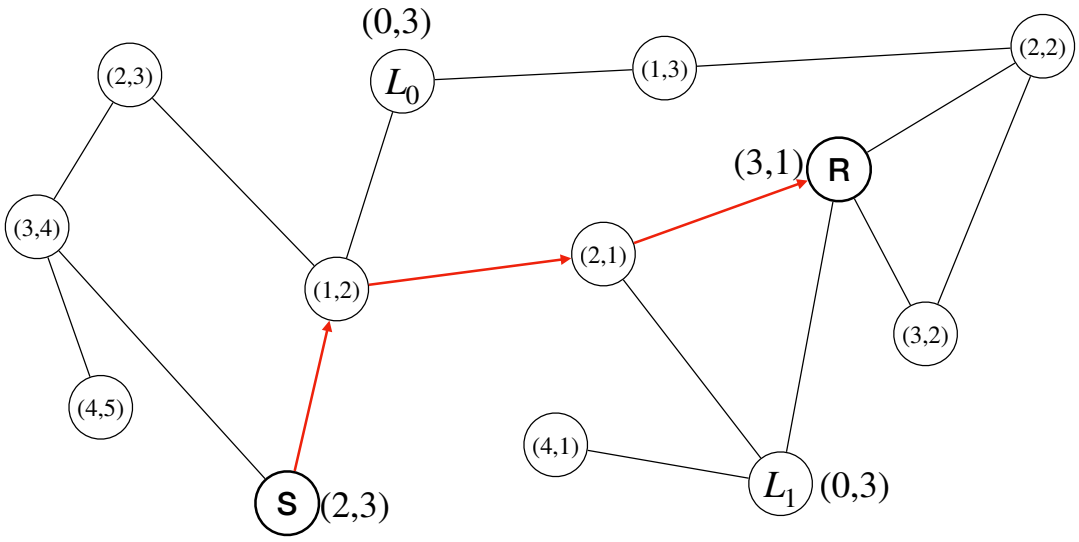


Fig. 7: Illustration of Embedding Based Routing from sender S and recipient R of a payment. An address space is spanned over the graph where the address consists of the distances to designated nodes L_0, L_1. For example, the distance from S to L_0 is 2 hops, whereas the distance of S to L_1 is 3 hops, thus its address is $(2, 3)$. The distance of two nodes with addresses (a_0, b_0) and (a_1, b_1) is $|a_0 - b_0| + |a_1 - b_1|$. Routes are found ad-hoc during routing by having each node attempt to forward to other nodes that are closer to R than themselves within the address space. This can result in shorter routes compared to landmark routing (Figure 6) as it does not force to have to route through a landmark. The above route in red is constructed by selecting the successor node with the lowest distance to R on each hop.

Classic Routing Protocols. PCNs are akin to Mobile Ad-Hoc Networks (MANET) or Wireless Sensor Networks (WSN) as nodes can spontaneously enter and exit the network and link quality can change dynamically. Another challenge is scalability: The amount of memory and routing messages sent should remain scalable, i.e. logarithmic in the network's size. There are different families of routing protocols that can be considered for these networks. For one, there are Re-Active routing protocols, where routes are discovered at the moment they are needed. For another, there are pro-active routing protocols where routing information is prepared in advance before a message is to be routed. Two pro-active routing protocol families are common: For one, there is landmark routing (Figure 6), where each node maintains a route to a set of designated nodes, called landmarks. Then, routes are created by selecting a landmark to route a path through. A variant of Landmark routing is *Hub Labeling* where for each node in the graph a set of hub nodes is maintained. These hub sets are chosen such that for two nodes S, R, the intersection of their hub sets is non-empty and contains a node on the shortest path between them. Embedding based routing (Figure 7) instead spans a dynamically created address space across the network which is then used for routing. For instance, by utilizing a set of designated nodes $v_0, \ldots, v_m, m \in \mathbb{N}$ each node's address is $(d_1, \ldots, d_m)$ where $d_i \in \mathbb{N}$ is the node's distance to v_i. Specific to PCNs is that connections in the network have capacities.

Routing as a Flow Problem. Routing a payment through a PCN, especially when considering payments across multiple paths [1, 24], is akin to computing flows in a Flow Network. In contrast to traditional routing protocols, flow based routing protocols (Figure 8) can find and utilize multiple paths [1, 54, 24] for payments which is highly beneficial: Routes can be found even for payments that are far larger than available channel capacities and, moreover, even channels with low capacities can be utilized for routing. However, the existing flow based routing protocols have critical issues: Both Tang et al. [64] and Robustpay+ [68] require full network knowledge

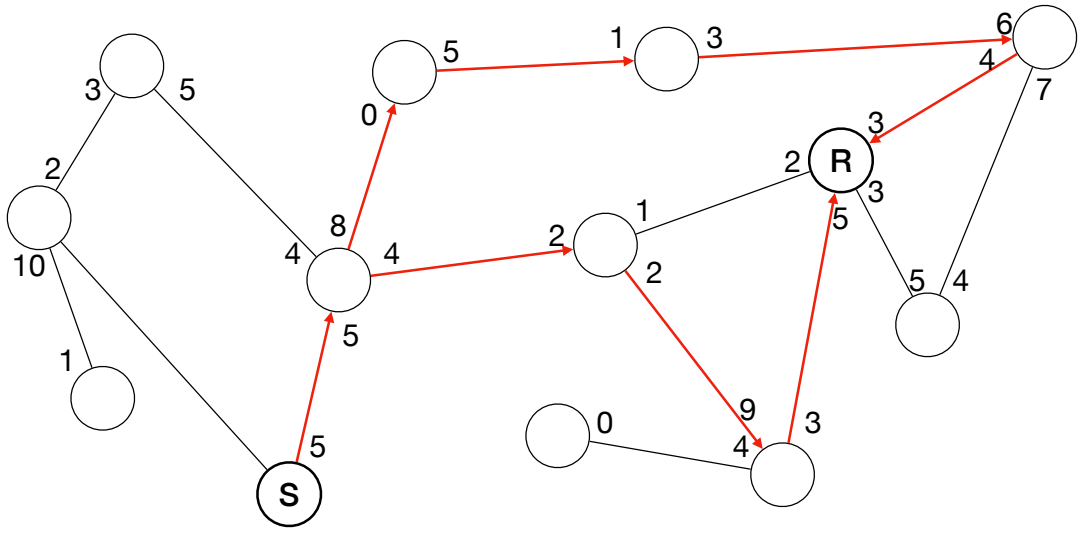


Fig. 8: Illustration of Flow based routing from sender S and recipient R of a payment of 5. Here we consider a directed flow network where we have directed edges and each edge in the network has a capacity and we aim to compute a flow of size b through it. In a PCN the capacities on an edge from node A to B equal to the amount of coins A has in the channel between A and B, whereas the flow we compute is the payment amount. In this Figure we annotate these capacities on the edges with two numbers, whereas the number closer to a node is the amount of coins that node has on the respective edge. Flow based routing utilizes capacities of channels and can help in splitting payments across multiple routes. This figure displays a flow of 5 routed through two routes from S to R.

and thus are unscalable, whereas the Spider Network [62] utilizes designated and trusted nodes, called Spider Hosts, to create a scalable transport layer on top of a PCN, and thus is vulnerable to byzantine Spider Hosts.

Routing Nodes. A common approach to create a scalable routing protocol for PCNs is to select a subset of nodes within the network and elevate them to be designated routing nodes (RN). This approach reduces the problem size as routing does not have to be done over all PCN nodes, but over the RNs instead. This allows the Spider Network [62] to implement a Flow based routing protocol. Other examples of protocols utilizing RNs are SPRITE by Panwar et al. [52] and RACED [41].

Summary. Table 3 provides an overview of routing protocols for PCNs. Flare [59] is the routing protocol that was proposed for the Lightning Network. Further work adapted common routing protocols for PCNs such as Landmark Routing in Silent Whispers [43], Embeddings based routing in Speedy Murmurs [61], and Ad-Hoc On Demand Vector routing by Hoenisch et al. [31]. Other works [62, 64, 68] looked at routing as a Network Flow problem and propose corresponding routing protocols. Of note is that existing flow based protocols are flawed: They either do not consider nodes behaving byzantine [62], or they are unscalable [64, 68] and thus cannot be executed by light nodes. Corcoran at al. [19] propose the only protocol that considers routing in PCNs where channels can have more than two parties, however, the underlying routing algorithm used is Dijkstra's algorithm, thus is unscalable, and no consideration is done regarding privacy.

3.4 Security of Offline Parties

Most channels, and notably Hydra, are designed to be trustless: A participating party does not need to trust any other party to ensure it is not losing their coins. Note that we discuss relaxing this requirement in Section

[5] Requires Onion-Style Messages

Name	Routing Protocol Family	Source Routing	Privacy	Scalable
Flare [59]	Hybrid Landmark	●	◐[5]	●
Silent Whispers [43]	Landmark	○	●	●
Speedy Murmurs [61]	Embedding	○	●	●
Spider Network [62]	Flow (RN)	○	○	●
AODV [31]	Reactive	○	○	●
Tang et al. [64]	Flow	●	○	○
Robustpay+ [68]	Flow	○	○	○
SPRITE (Panwar et al.) [52]	RN	○	●	●
RACED [41]	RN	○	◐	◐
LightPIR [56]	Hub Labeling	●	●	○
Corcoran et al. [19]	Dijkstra	●	○	○

Table 3: Overview of Routing Protocols in Channel Networks. Here we compare existing protocols by (1) their approach, (2) whether they utilize source routing, (3) whether they are privacy preserving and (4) whether the protocol is scalable in the size of the network, i.e. each node has logarithmic storage complexity and the amount of routing messages sent per route discovery is logarithmic. Flare utilizes a hybrid approach of re-active and pro-active routing methods. RACED requires its RNs to leak some information about them publicly. Moreover, it only addresses routing between RNs, but leaves the problem open of finding a route from a payment sender to a RN node and from a RN node to the payment receiver.

3.4. A drawback of this trustless design is that parties must be always online to ensure they can dispute if another party aims to close the channel in a state in which they would lose their coins. Several works attempt to remove the always online requirement by utilizing a third party: (1) Watchtower, (2) Sleepy Channels and BFT based protocols.

Watchtower. One of the earliest proposals was the LN compatible Watchtower construction *Monitor* [6] where a set of channel monitors are employed to dispute a channel for an offline party. However, there are no consequences for a monitoring party if they fail to do so. Figure 9 illustrates the concept of a Watchtower: We consider a payment channel between Alice and Bob, as well as a Watchtower who has some collateral allocated on the ledger. Before Alice goes offline, she appoints the Watchtower to dispute in her stead by submitting the latest channel state σ_n. For this, she receives a receipt ν that she can use as proof of this employment. Then, in case Bob acts maliciously and attempts to close the channel in an older state than σ_n, the Watchtower can dispute in Alice's stead. However, if the Watchtower fails to do so, Alice can use the receipt ν to punish the Watchtower, e.g. by having its on-chain collateral be burnt.

Disclose Cascade Watch Commit [9], compatible with the LN, is considering the set of miners to act as Watchtower. Transactions performed in a channel are first disclosed and cascade through the network of miners and Watchtower. Thereafter the miners watch the blockchain and commit transactions in case of a dispute. Miners are rewarded upon participation of the dispute process. Outpost [38] and Cerberus [15] propose Watchtowers that are paid upon updating the transactions they should enforce and are compatible with the LN.

PISA [46] is a smart contract based Watchtower construction where the Watchtower is accountable for misbehavior or failure to dispute by burning collateral put aside. However, this involves a tradeoff. Under full accountability the Watchtower's collateral is burnt on failure, so the customer cannot recover their lost coins. If instead the collateral is paid out to the customer, a Watchtower can recover part of its own forfeited collateral by registering Sybil customers and compensating itself. Counter-Collusion [69] considers bribery of the Watchtower and its collusion with an adversary. It utilizes a game theory based approach to design a system resilient to such a case in the face of a rational adversary. Garrison [49] and FPPW [48] employ adaptor signatures, a primitive that can be based on Schnorr or ECDSA signatures, to create Watchtowers.

Sleepy Channels. Another line of work by Aumayr et al. [3, 8] proposes alternative approaches to Watchtowers to keep channels secure even if parties go offline. Sleepy Channel [8] is a payment channel construction. It restricts unilateral channel closure to specific time windows where parties must be online in order to dispute and can be offline otherwise. Cooperative channel closure, i.e. when all parties agree to close a channel and cooperate to do so, can be done at any time. A drawback of Sleepy Channels is that they have a fixed lifetime and are not valid arbitrarily long.

[6] Slides at: https://scalingbitcoin.org/milan2016/presentations/D1%20-%208%20-%20Tadge%20Dryja.pdf

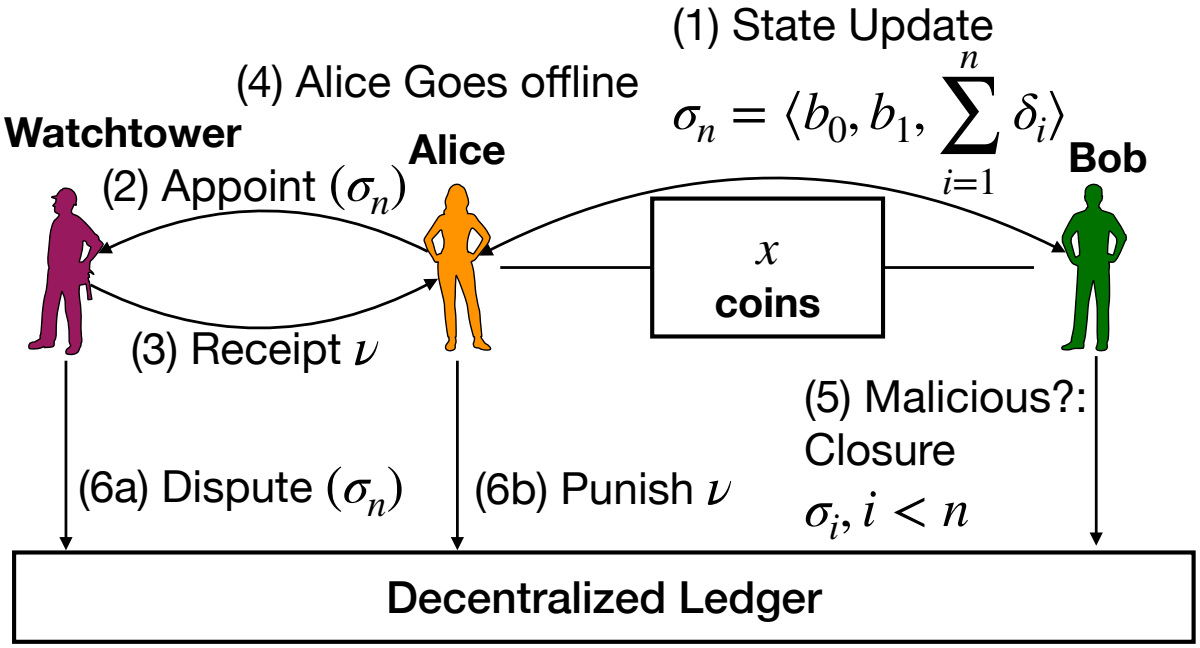


Fig. 9: Illustration of a PISA [46] style Watchtower considering a channel between Alice and Bob, as well as a Watchtower employed by Alice. (1) After any update of the channel's state $\sigma_n = \langle b_0, b_1, \sum_{i=1}^{n} \delta_i \rangle$, which is derived from initial balances b_0, b_1 and n payments δ_i, (2) Alice appoints the Watchtower using σ_n and (3) receiving a receipt ν which holds the Watchtower accountable. (4) If Alice goes offline and Bob is malicious (5) Bob can attempt to close the channel in an old state $\sigma_i, i < n$. Then either (6a) the Watchtower disputes in stead of Alice using σ_n on-chain, or (6b) Alice uses the receipt ν to punish the Watchtower for its failure to act.

CRAB, Sleepy CRAB [3] and SUBORN Channels [12] consider Timelock Bribing Attacks in addition, i.e. miners are bribed to censor transactions that are sent in order to dispute incorrect channel closure or delay them until after the dispute phase expired. This effectively breaks the ledger's liveness parameter Δ. This could lead the channel to be closed in an incorrect state. This is achieved by having channel participants allocate additional collateral γ on channel opening where a successful dispute will award γ coins taken from the cheating party's collateral to the miner that includes the dispute transaction on the ledger. However, this requires $\gamma \geq \frac{c}{2}$ in the case of a rational adversary, i.e. one that does not perform an attack that would result in it losing coins, and $\gamma \geq c$ in the case of an byzantine adversary where c is the channel's capacity.

Sleepy CRAB, illustrated in Figure 10, allows parties to go offline by utilizing a revocation key that allows a party to perform a dispute. Each state update results in a new revocation key, however, it is created using trapdoor one-way functions such that older revocation keys can be derived from newer ones. When a party goes offline, they submit the latest revocation key on a public bulletin board which can be utilized by miners to perform a dispute and receive the allocated collateral as reward. This is of particular interest, as such rewards can yield an additional revenue stream for Mining Pool operators.

BFT. Most Layer 2 protocols attempt to be *trustless*, meaning that even if all but one parties are acting maliciously, the honest party should not be in risk of losing coins. In the setting of two-party protocols like payment channels this is the required setting. However, moving to a multi-party setting raises the question whether security can be relaxed in favour of practicality: E.g. utilizing BFT protocols and corresponding assumptions allows parties to go offline [10]. On top of that and in contrast to Watchtower and Sleepy Channels a BFT based approach does not only allow parties to be offline, but also allows the protocol to proceed even if some parties are offline. Thus, whereas the Hydra Head protocol [17], defined in the trustless setting, can be stalled by only one byzantine party, executing the Hydra Head protocol within a BFT setting would provide resilience against byzantine nodes.

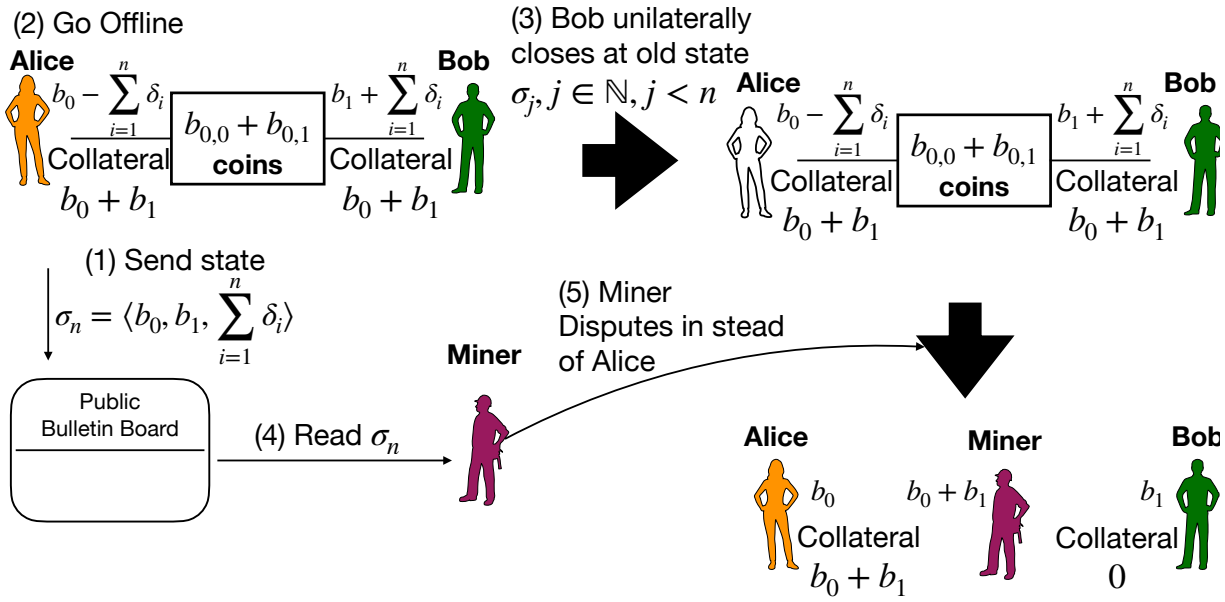


Fig. 10: Utilization of miners to allow channel participants going offline. This might require each party to lock collateral for the duration of the channel, equal to the amount of coins within the channel itself. Before Alice goes offline, (1) she publishes on a public bulletin board information on the latest channel's state sufficient to dispute it. (2) After she goes offline, (3) Bob might attempt to close the channel in an old state. In this (4) a miner retrieves the information published by Alice to (5) dispute the channel on her stead. In the process, the miner receives some rewards, e.g. the collateral of the maliciously behaving party.

Name	Method	Requires Smart Contract	Trustless
PISA [46]	Watchtower	●	◐
DCWC [9]	Watchtower	○	○
Outpost [38]	Watchtower	○	○
Cerberus [15]	Watchtower	○	●
Garrison [49]	Watchtower	○	●
FPPW [48]	Watchtower	○	●
Counter-Collusion [69]	Watchtower	●	○
Sleepy Channel [8]	Sleepy Channel	○	●
Sleepy CRAB [3]	Sleepy Channel	○	●
Brick [10]	BFT	○	●

Table 4: Overview of protocols that allow parties to go offline. This table (1) gives an overview of the approach used, and compares whether (2) the proposed protocol requires a ledger with smart contract capability and (3) whether a protocol is trustless and an honest party has no risk of loss of coins even in the context of a non-rational adversary. PISA [46] proposes two variations of their protocol: In one, the Watchtower is punished when failing to dispute a channel for a customer by loss of all of its collateral, but the protocol has no balance security. In the other, the protocol does have balance security, but the Watchtower can mitigate its punishment to an arbitrary but not full degree.

Brick [10] and SyncPCN/PSyncPCN [26][7] are a line of work that aims to achieve secure channel within a non-synchronous network, i.e. when the ledger does not have any liveness parameter Δ and transactions might take arbitrarily long to be accepted on the ledger. Brick is a two-party channel construction, relying on consistent broadcast. It employs a BFT committee consisting of *wardens* with a rational majority. It remains secure, even if both channel participants are malicious, as long as the richer one of them behaves rational. SyncPCN/PSyncPCN extends brick to allow multi-hop payments, allowing a PCN to be formed even on ledger's without synchronicity, i.e. Δ liveness.

[7] https://dl.acm.org/doi/pdf/10.1145/3558535.3559779

Summary. Table 4 summarizes existing Watchtower protocols. With the exception of PISA [46], one of the first Watchtower constructions developed concurrently to DCWC [9], a lot of work has been done in research for Bitcoin, thus Lightning Network compatible Watchtower. A challenge in the design of Watchtowers is whether a customer of a Watchtower enjoys *balance security*, i.e. whether they can lose coins in case the Watchtower misbehaves and fails to dispute for the customer.

3.5 Re-Balancing Channels

Performing payments in a PCN reduces the amount of coins available within each channel of the payment path on the sender's side and increases the amount of coins available on the receiver's side. If more coins tend to be transferred into one direction within a channel than the other, then the amount of coins in that direction will deplete and make it unusable for further payments into that desired direction. On one side, this can be resolved by interacting with the ledger, e.g. a trivial method is closing and reopening the channel, however, more elaborate methods exist, e.g. Shaduf [27], Cycle [32] and Loop[8]. While Shaduf requires on-chain interaction, the number of transactions sent to the ledger is constant in the amount of re-balance operations, but linear in the amount of channels that re-balance. On the other side there are Layer 2 protocols [39, 13, 55, 14, 65] that mitigate this issue by performing payments into the opposite of the depleted payment direction to make coins available for the preferred payment direction. These payments are done on a circuit, i.e. a path on the PCN where the first and last nodes are equal, such that no node gains or loses coins. This approach is illustrated in Figure 11. Cycle [32] requires on chain interaction by having pre-defined circuits in a PCN register with an on-chain smart contract. Thereafter, the channels within any circuit remain balanced through an asynchronous off-chain protocol.

Re-Balancing. One of the first approaches is REVIVE [39], where linear programming is used to find a suitable payment circuit for re-balancing channels. The work assumes an honest-but-curious adversary. An issue of running an optimization problem on a set of channels is privacy. Proactive Re-balancing [55] proposes an imbalance metric and utilizes this to guide the re-balancing algorithm whereas decisions are made locally by each node, making the protocol privacy-aware. A line of work by Avarikioti et al. [13, 14] further explores re-balancing. Hide & Seek [13] is a privacy preserving approach that utilizes Secure Multiparty Computation (MPC). It proposes an Opt-In protocol where, first, the linear programming algorithm is executed through MPC via k delegates and, second, the payment circuit is executed among the participants. Musketeer [14] further builds up on this by investigating incentives for participation instead of relying on an Opt-In mechanism. It utilizes auctions to bid for and sell re-balancing services.

Netting. An adjacent problem to re-balancing is the problem of bank clearing or netting: We assume a set of nodes or payment hubs that are well connected among one another. Frequent payments between these can result in channel imbalance preventing further payments. Wiser [65] proposes a protocol where transactions are collected and aggregated and then performed atomically among all nodes, thus improving utilization of the capacity of the channels by preventing (temporary) imbalances.

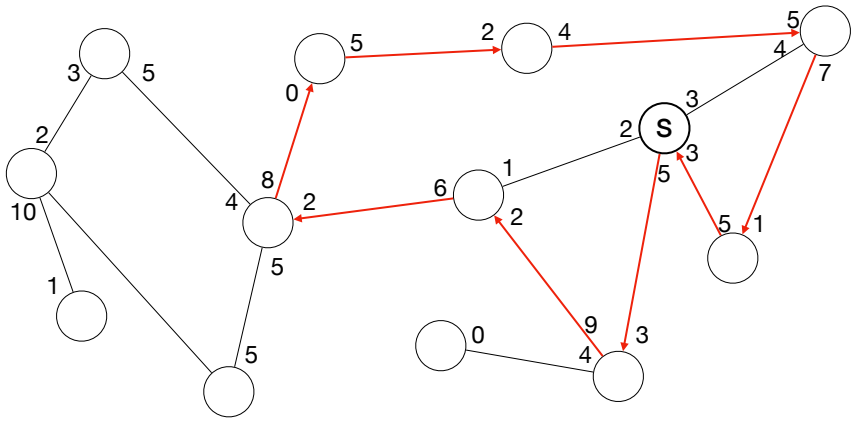
Summary. Table 5 provides an overview and compares network rebalancing protocols. For one, we give an overview of the approach used to find payment circuits for re-balancing. Whereas REVIVE [39] and Hide & Seek [13] utilize Linear Programming (LP) to solve the optimization problem of re-balancing, Proactive Re-balancing [55] and Musketeer [14] make nodes do decisions on re-balancing locally, by either utilizing an imbalance metric that each node computes locally or driving incentives through an auction system respectively.

3.6 Payment Privacy

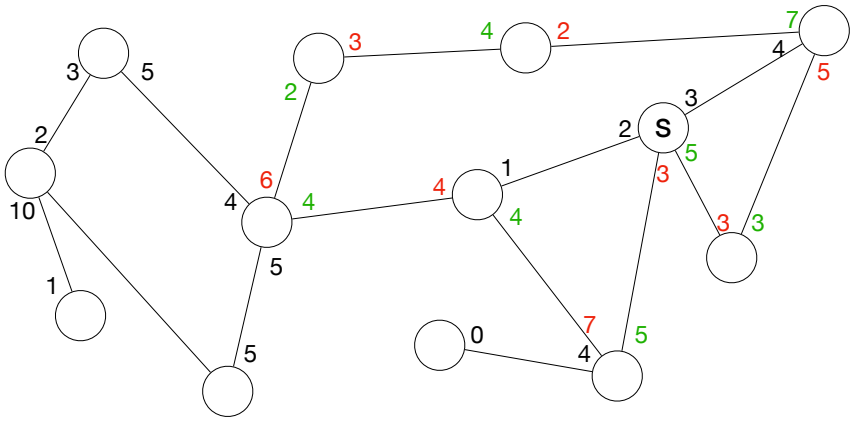
Layer-2 channels enhance privacy to a limited degree by not publicly recording transaction on a distributed ledger, whereas transactions in Layer 1 are readable indefinitely on the public ledger. Instead, only a summary, i.e. an aggregation of all transactions that occurred on a channel will be visible on Layer 1 upon channel closure. Nevertheless, transactions in Layer 2 are not formally private or anonymous. Intermediaries that assist in forwarding a payment within a PCN are able to view payment amounts and can attempt to infer sender and receiver of payments. Moreover, if a HTLC payment fails and has to be resolved on Layer 1, the payment will be public on Layer 1.

Private HTLCs. One approach to improve privacy of payments is to modify the HTLC protocol to be privacy preserving. AMHL [45] as well as Fulgor & Rayo [44] modify HTLC payments to unlink the HTLC contracts on each channel from another, preventing parties to identify sender and receiver of payments, even if the payment is resolved on Layer 1.

[8] https://lightning.engineering/loop/



(a) Imbalanced PCN and Payment Circuit



(b) Balances post balancing

Fig. 11: Execution of a re-balancing circuit. Nodes represent parties whereas edges represent channels between parties. Edges are annotated with two numbers next to each node. The number on a channel next to a node represents how many coins they have within that channel. Figure 11a displays an unbalanced PCN and a payment path in red from and to node S. Figure 11b displays the effect on the channel balances of executing a payment of 2 coins on the red path. Red numbers indicate that a party's balance in that channel was reduced whereas green indicate it increased. Finding appropriate paths for channel re-balancing can be done, for instance, through an optimization problem [39, 13].

Name	Method	Privacy	Requires Smart Contract
REVIVE [39]	LP	○	●
Hide & Seek [13]	LP through MPC	●	○
Musketeer [14]	Auctions	●	○
Proactive Rebalancing [55]	Local imbalance metric	●	○
Shaduf [27]	On-Chain	○	●
Cycle [32]	Asynchronous Messaging	●	●
Wiser [65]	Netting	●	○

Table 5: Overview of re-balancing protocols. REVIVE and Hide & Seek attempt to find a circuit through an optimization problem solved through Linear Programming (LP), whereas Proactive Rebalancing leaves the choice to each node locally through an imbalance metric. Of the above methods Hide & Seek, Musketeer, Proactive Rebalancing, Cycle and Wiser are privacy preserving. Lastly, we indicate whether the given protocol requires a ledger with smart contract capability.

Private Channel Balances. While the balance of a channel is only available to the channel's participants, any adversary can attempt to deduce it by attempting to route payments through it: If routing of a payment of $b_p \in \mathbb{N}$ coins succeeds, the channel capacity in the direction of the payment has has at least b_p coins. Doing such probes can leak information of channel balances across a PCN. Repeating such probes can then leak information on payments performed such as payment amount, sender and receiver. Twilight [20] proposes adding noise to payment route probes to achieve Differential Privacy.

Tumblers and Mixers. Another approach are so called Tumblers, or Mixers, that are inspired by Chaum's ECash [18]. Tumblers [30, 63, 29, 60] as illustrated in Figure 12 are payment hubs that allow adjacent nodes to perform transactions with one another while sender and receiver remain unlinkable. Figure 12 displays a tumbler similar to TumbleBit [30] and $\mathsf{A^2L}$ [63]. TumbleBit [30], one of the first Tumblers for Bitcoin, is inspired by Chaum's ECash [18]: withdrawal and spending of coins should be unlinkable by the bank that issues the coins. In the context of a Tumbler this translates to sender and receiver of coins should be unlinkable even by the Tumbler operator. Of note are $\mathsf{A^2L}$ [63] which introduced the use of *adaptor signatures* for construction of a Tumbler, however, later work [29] identified a vulnerability and proposes a fixed version $\mathsf{A^2L^+}$, as well as a Tumbler formalized within the UC framework $\mathsf{A^2L^{UC}}$. Moreover, Blind Hub [60] introduces *Blind Adaptor Signatures* and a Tumbler formalized in the UC framework where payment amounts do not need to be constant.

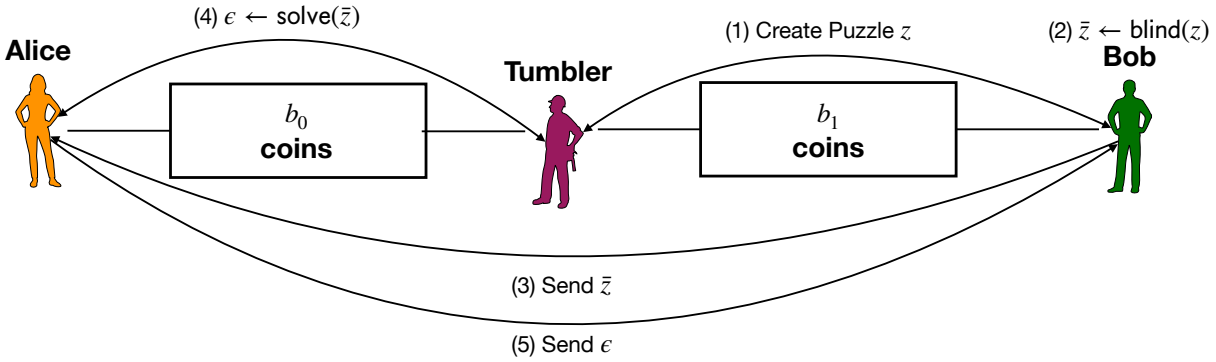


Fig. 12: Illustration of a tumbler implemented through blinded puzzles [30]. Both Alice and Bob are connected with the Tumbler using payment channels. When Bob wants to receive a payment from Alice, (1) they execute a protocol with the Tumbler to receive a puzzle z which they (2) blind, receiving $\bar{z}$, and (3) send to Alice. (4) This blinded puzzle is then used in a protocol between Alice and the tumbler to receive a solution ϵ. In the process, the tumbler receives the respective payment amount through the payment channel from Alice. Lastly, (5) Alice sends ϵ to Bob, who can use it to enforce a payment from the Tumbler to him on their payment channel.

Auditability. While privacy is a critical topic, if private channels can be used to circumvent regulations, it would impede adoption of the technology. However, being able to audit channels appears to be in conflict with channel privacy. To address this, privacy preserving channel audits have been proposed for Hydra [16].

Summary. Table 6 gives an overview and compares tumbler protocols. For one, it compares the protocol's security, i.e. whether it was formalized within the UC framework or even was broken, which is the case for $\mathsf{A^2L}$ [63] which was broken [29]. Moreover, it compares the approaches and cryptographic primitives used to implement the respective tumbler. We note that this line of work requires the Tumbler itself to issue signatures used for both, redeeming and issuing of a payment, but techniques must be utilized to unlink these two parts of each payment, e.g. through Blind Signatures as in TumbleBit [30]. More recent work [29, 60] utilizes adaptor signatures [57], however, we note that even in case of a protocol formalized in the UC framework, at the time of their publication no UC-secure adaptor signature existed; the first was proposed by Gerhart et al. [28].

Name	UC Secure	Cryptographic Primitive
TumbleBit [30]	○	Blind Signatures
$\mathsf{A^2L}$ [63]	Broken [29]	Re-Randomizable Signatures
$\mathsf{A^2L^+}$ [29]	○	Adaptor Signatures [57]
$\mathsf{A^2L^{UC}}$ [29]	●	Adaptor Signatures
Blind Hub [60]	●	Blind Adaptor Signatures [60]

Table 6: Overview of Tumblers utilizing Layer 2 payment channels. We compare (1) whether the work has been formalized in the UC framework and (2) given an overview of the cryptographic primitive used for implementation of the tumbler. While $\mathsf{A^2L^{UC}}$ [29] is formalized in the UC framework, it does not model any adaptor signature as an ideal functionality; instead it invokes a secure adaptor signature scheme directly in the protocol construction. However, since then, an adaptor signature scheme has been formalized within the UC framework [28]. Of note is that $\mathsf{A^2L}$ has been shown to be broken [29].

4 Hydra Settings

When evaluating the applicability of tools for Hydra we consider the following use cases, each having a different setting and set of problems.

Hydra. In the *Hydra* setting we consider one Hydra Head by itself as defined in the original paper [17] or the UC formalization [37].

PCN over Hydra. Here we consider a set of Hydra Heads where a PCN is formed by using HTLCs to enable payments between parties on different Hydra Heads. This is particularly relevant in network topologies in which Hydra Heads are formed with two parties each, e.g. a star topology [9] forming a payment hub.

State Machines across a Hydra Network. Here we consider a set of Hydra Heads connected through either Interhead [36] constructions, forming virtual Hydra Heads, or ad-hoc Hydra Heads [33]. Note that for the latter a formalization in the UC framework is in development. In this setting parties in different Hydra Heads can jointly execute smart contracts in Layer 2 and are not limited to payments as in the previous setting.

5 Evaluation of Potential Tools for Hydra

Here we focus on discussing the applicability of existing tools (Section 3) in a setting where we have exactly one Hydra Head.

5.1 Offline Parties

Hydra requires that all parties participating stay online. This is a difficult to realize requirement, as nodes within a PCN can be unreliable and can go offline even involuntarily, e.g. through software crashes, network issues or power outages. However, staying online is essential as otherwise (1) a party might fail to participate in channel dispute and (2) it will disable transaction and snapshot generation which requires a signature of all participating parties. Accommodating for parties going offline in the context of Hydra offers unique challenges, which we consider in the following as well.

[9] https://hydra.family/head-protocol/topologies/star-shaped

In this section, we consider Δ Liveness assumption as follows:

Definition 1 (Δ Liveness). *A decentralized ledger satisfies Δ-Liveness if, upon submission by an honest party, every transaction becomes confirmed within at most time Δ.*

Thus, we consider the following open problem:

Open Problem 1 (Offline Parties) *Going offline in a Hydra Head exposes a party to two distinct risks: (i) an adversary may close the Head in an outdated state while the offline party cannot dispute, and (ii) the Head protocol stalls because transaction confirmation requires signatures from all participants. Can a Watchtower-like construction for Hydra mitigate risk (i) while preserving balance security, and can it do so without revealing to co-participants whether a given party employs a Watchtower? Separately, can the Head protocol continue to confirm transactions in the absence of one or more parties, and under what trust assumptions?*

A large number of tools support parties being offline, such as Watchtower (Section 3.4) and Sleepy Channels (Section 3.4). Thus, investigating what a Watchtower for Hydra could look like is an open problem.

Properties that such a construction for Hydra should fulfill are

- There should be an incentive to operate a Watchtower
  - The incentive should be proportional to the work done, e.g. the Watchtower receives some coins every time they update the latest transaction
- The Watchtower is incentivized to act honestly and participate in channel disputes whenever necessary through either:
  - Punishment and loss of collateral
  - Reward upon dispute
- An honest customer should enjoy balance security
- Whether a party participating in a Hydra Head employs a Watchtower should be kept private

Additionally, if attempting to implement Sleepy CRAB [3], which co-opts block producers to effectively act as Watchtowers, we face the additional challenge of proper incentivisation under Cardano's fee model. Unlike Bitcoin's fee market, Cardano's fee is a deterministic function of transaction size, so there is no mechanism by which a party can outbid others for inclusion — which removes the standard lever for rewarding a disputing block producer.

Existing watchtower designs of note that can serve as a starting point for this endeavour are Cerberus [15] for its incentive structure and FPPW [48] for its privacy preserving properties. In addition, both designs satisfy balance security.

5.2 Weaker Security Assumptions

Some tools could allow for implementations of Hydra with fewer Security Assumptions. In the following we first consider Non-Synchronous Networks in where Δ-Liveness does not hold (Section 5.2.1). Then we consider the case where parties behave byzantine andstall transactions confirmation (Section 5.2.2). Lastly, we consider the case where all parties are corrupted (Section 5.2.3).

5.2.1 Non-Synchronous Networks. Brick [10], Sleepy Channels [8] and Sleepy CRAB [3] propose channel implementations that remain secure even if Δ Liveness is not guaranteed by the underlying ledger. This is especially notable since it is unclear whether and for what values of Δ any ledger fulfills Δ-Liveness. Most channel implementations do not document how their value of Δ was derived, and different implementations use markedly different values. In the case of the Lightning Network, the CLightning client uses a default of $\Delta = 34$ blocks (340 minutes on average) whereas the default of LND is $\Delta = 80$ blocks (800 minutes). Note that users can change these default settings. Optimism [10] estimates about 10 minutes for a transaction to be included in a block and 2 epochs until it is finalized, resulting in a Δ of about 23 minutes in total in regular network conditions. They admit that Δ might differ when there is network congestion, however, do not attempt to estimate values for Δ in such a case. In the case of Cardano, it appears that with a value of $\Delta = 85$ blocks approximately 0.01% of transactions would fail to be included on the ledger within that time [11]. Note that this is a quantile, not a guarantee: a bound under which 99.99% of transactions confirm is precisely not a Δ-Liveness. In practice, no deployed ledger is known to satisfy Δ-Liveness absolutely; what is achievable is a (Δ, ε)-liveness guarantee in which the failure probability ε is left unquantified. Making ε explicit is part of the next open problem. It is not clear at which network conditions this value has been evaluated. The amount of mining fees provided to a transaction as well as whether the ledger is saturated with transactions or even experiences congestion can affect the time it takes for a transaction to be included on the ledger. Thus we consider the following open problem:

[10] `https://docs.optimism.io/op-stack/transactions/transaction-finality`

[11] `https://aiken-lang.org/fundamentals/what-i-wish-i-knew#transaction-latency-vs-finality`

Open Problem 2 (Cardano Δ) *What methodology can rigorously determine a value of Δ for the Cardano ledger such that Δ-Liveness holds with quantifiable probability, as a function of transaction fees and network congestion? In particular, can such a methodology provide formal statistical or game-theoretic bounds on Δ, rather than empirical point estimates under benign conditions?*

Deriving appropriate values for Δ requires empirical evaluation for each ledger under consideration. Having appropriate values for Δ is critical for the security of Layer 2 protocols, but values currently used in real world protocols (Table 7) lack sufficient justification to be used for this purpose. Thus, evaluating and sufficiently justifying values for Δ in the Cardano blockchain is essential for trusting security of Layer 2 protocols on Cardano. This evaluation must account for adverse network conditions — including congestion — and model Δ as a function of the transaction fee offered relative to the prevailing average fee.

Name	Blockchain	# Blocks	Est. time for Δ
CLightning	Bitcoin	34	340 minutes
LND	Bitcoin	80	800 minutes
Aiken	Cardano	85	28.3 minutes
Optimism	Ethereum	115	23 minutes (avg. network conditions)

Table 7: Overview of ledger Δ values used in different blockchain protocols. CLightning and LND are both clients for the Lightning Network, but utilize significantly different values for Δ. The Cardano figure is derived from the 85-block value at approximately 20 sec per block the source does not address network conditions of transactions fees. Optimism estimates 115 blocks (Ethereum L1 blocks as Optimism is rollup) in total between sending a transaction and having it included on the ledger at average network conditions resulting in a value of Δ being 23 minutes.

However, even if the required work has been done to find values for Δ such that the underlying ledger fulfills Δ Liveness with sufficiently high probability, *Bribing Attacks* can be used to break this liveness assumption. In a *Timelock Bribing Attack* [66, 67, 51, 12] or a *Bribe & Fork Attack* [11] a corrupted party attempts to close a channel in an old state, but attaches a high fee to that transaction, incentivizing block producers to include that transaction on the ledger. Thus, we consider the following open problems:

Open Problem 3 (Timelock Bribing Attack) *Hydra's dispute mechanism relies on Δ-Liveness, which can be undermined by timelock bribing attacks. Cardano's delegated proof-of-stake consensus and fixed fee structure differ fundamentally from Bitcoin's mining incentive model on which existing bribing attack analyses [11, 12, 51] are based. Can these structural differences be exploited to make Hydra inherently more resilient to timelock bribing, or conversely, do they introduce novel attack vectors not present in proof-of-work settings?*

A concrete next step is to model the cost of a timelock bribing attack under Cardano's stake-based consensus and deterministic fee formula, and compare it to the known costs under Bitcoin's PoW model [12, 51], to determine whether the attack economics differ qualitatively.

Eventually, the question arises on whether we can achieve security of Hydra, regardless of whether the assumption of Δ Liveness holds, i.e. in the case of Hydra being executed in a non-synchronous network.

Open Problem 4 (Non-Synchronous Hydra) *Can a Hydra Head be constructed that is secure without assuming Δ-Liveness of the underlying ledger, following approaches such as Brick [10] or Sleepy CRAB [3]? As a weaker but practically important variant: can the existing Hydra Head construction be augmented with a graceful degradation mechanism that preserves safety (no party loses coins) during transient failures of Δ-Liveness, while restoring liveness once synchrony resumes?*

Brick [10] and Sleepy CRAB [3] propose interesting channel constructions for non-synchronous networks.

5.2.2 Fault Tolerant Head Protocol As mentioned in Section 5.1, if any party goes offline, the Hydra Head protocol is halted. In the following we consider the Liveness property as in Hydra [17]:

Definition 2 (Liveness). *Every transaction becomes confirmed at some point.*

This is a weaker version of the Δ Liveness assumption. Hydra fulfills the *Liveness* property only conditionally: it holds if no party is corrupted and the adversary delivers all messages eventually. Open questions for fault tolerance consider whether we can relax these assumptions and still retain *Liveness*.

Thus, we consider the following open problems:

Open Problem 5 (Fault Tolerant Liveness.) *Hydra's current design requires unanimity for transaction confirmation, providing trustless security (one honest party suffices) at the cost of liveness when any party is byzantine. Can a Hydra Head construction achieve Liveness (Definition 2) in the presence of up to f byzantine parties? What is the minimal trust assumption — for instance, an honest majority — required for such a construction, and what is the resulting degradation in security guarantees, particularly the loss of the trustless property?*

Open Problem 6 (Rational Liveness) *A rational adversary will not stall the Hydra Head protocol unless the expected payoff from doing so exceeds the cost. Can an incentive mechanism be designed — such as penalties for non-participation or rewards for timely signing — that makes stalling a strictly dominated strategy for rational parties, thereby achieving Liveness without strengthening trust assumptions beyond a single honest party? This is a strictly weaker requirement than Byzantine fault tolerance (Open Problem 5) and may preserve the trustless property that BFT-based approaches sacrifice.*

Open Problem 5 could be addressed by replacing the unanimity requirement with a BFT protocol for transaction and snapshot confirmation, at the cost of weakening the trust model from one-honest-party to honest-majority. Open Problem 6 takes a different approach: rather than changing the consensus mechanism, it asks whether incentive mechanisms can make non-participation irrational, preserving the existing unanimity structure. Brick [10] is a relevant starting point for both directions, as it demonstrates how incentive compatibility and BFT can coexist in a channel construction.

5.2.3 Corrupting all Participants.

Brick [10] proposes a channel construction where all participants can be corrupted, but remains secure as long as the party holding the majority of coins is rational, even if malicious. A challenge in adapting this approach for Hydra is to extend Brick's approach from two-party channels to Hydra's multi-party channels. Another approach to have security even in the face of corruption of all participating parties is the use of Zero Knowledge Proofs (ZKPs) similar to implementations of ZK-Rollups.

Thus, we consider the following open problem:

Open Problem 7 (Malicious Hydra) *Can a Hydra Head be designed to guarantee correct state machine execution even when all participating parties are corrupted? Specifically, can techniques from zero-knowledge proof systems — as employed in ZK-Rollups to verify off-chain computation — be adapted to produce verifiable proofs of correct Head execution that can be checked on Layer 1? As a weaker variant, if all parties are corrupted but act rationally, can Brick's [10] approach of securing the richer-but-rational party be extended from two-party channels to Hydra's multi-party setting?*

As with the case of a fault tolerance of the Hydra Head protocol, relevant related work that can serve as a starting point is Brick [10].

5.3 Anonymity / Privacy

In the following we assume a payment within a Hydra Head of $b \in \mathbb{N}$ coins from S to R. We furthermore consider privacy of a payment which is illustrated in Figure 13 and more formally as follows:

Definition 3 (Value Privacy). *No party $p \notin \{S, R\}$ can learn the value of b.*

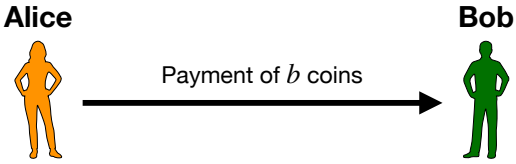
Definition 4 (Relationship Privacy). *When observing a specific payment, no party $p \notin \{S, R\}$ can learn that there is a relation between S and R.*

Definition 5 (Sender Privacy). *When observing a specific payment, no party $p \notin \{S, R\}$ can learn information on the sender S of a payment.*

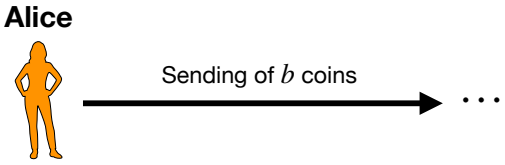
Definition 6 (Receiver Privacy). *When observing a specific payment, no party $p \notin \{S, R\}$ can learn information on the receiver R of a payment.*

Since transactions within a Hydra Head must be confirmed by all participating parties, full privacy against co-participants is structurally impossible: every party observes every transaction. However, Tumbler constructions can provide relationship privacy and value privacy against non-transacting co-participants by routing payments through an intermediary that unlinks sender and receiver. Blind Hub [60], a Tumbler formalized within the UC framework, achieves both Value Privacy and Relationship Privacy, but was designed for account-based ledgers — its applicability to Hydra's EUTxO model is an open question.

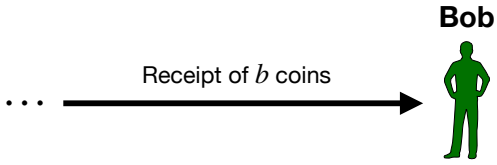
Thus, we consider the following open problems.



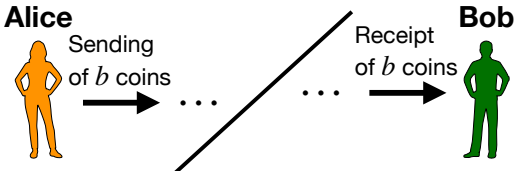
(a) Public Payment



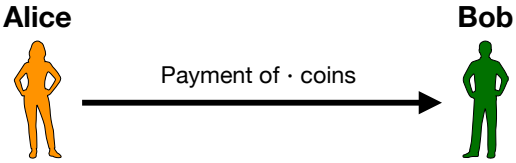
(b) Receiver Privacy



(c) Sender Privacy



(d) Relationship Privacy



(e) Value Privacy

Fig. 13: Illustration of privacy properties in payments. Figure 13a displays the setting we are looking at which is a public payment from Alice to Bob of b coins. Figure 13b displays receiver privacy, i.e. observing a payment does not reveal information on its receiver, whereas Figure 13c displays Sender privacy where the sender of the payment is kept private respectively. Relevant for the implementation of tumblers (Section 3.6) is relationship privacy as illustrated in Figure 13d, where an observer can see Alice and Bob send or receive a payment respectively, but they cannot link these to conclude that Alice performed a payment to Bob. Lastly, there is value privacy as illustrated in Figure 13e where the amount of coins of a payment is private.

Open Problem 8 (Payment Privacy) *In a Hydra Head, all participants co-sign every transaction, making full privacy against co-participants structurally impossible. Can relationship privacy (Definition 4) and value privacy (Definition 3) be achieved against non-transacting co-participants within the Head, for instance through an intra-Head Tumbler construction? What is the minimal set of privacy properties achievable in this constrained setting? In particular, does Hydra's requirement that every participant co-sign every transaction invalidate the unlinkability arguments of existing Tumblers [29, 30, 60], and does the EUTxO model — in which a Tumbler can be expressed as a single validator script carrying its own state — offer any means of restoring them?*

Open Problem 9 (Smart Contract Privacy) *Privacy definitions for payments (Definitions 3, 4, 5, and 6) illustrated by Figure 13 do not directly generalize to arbitrary state machine interactions. A state machine transition involves an input, an output state, and the identity of the invoking party — each of which may independently require privacy. Can formal privacy notions for state machine interactions in a Hydra Head be defined, distinguishing at minimum: (i) input privacy (hiding what was submitted), (ii) transition privacy (hiding which smart contract function was executed), and (iii) invoker privacy (hiding who triggered the transition)? Which of these notions are achievable given that all Head participants observe all transactions?*

When investigating privacy enhancing technologies for Hydra Heads, we consider $\mathsf{A}^2\mathsf{L}^{\mathsf{UC}}$ [29] and Blind Hub [60] to be potential entry points as they are formalized within UC. Utilizing these and the UC formalization of Hydra [37] can serve as a base to create protocols that can be formalized within the UC framework in turn.

5.4 Summary

Here, we give a summary of the open problems we identify for Hydra Heads. For each overarching topic, we sum up the relevant related work.

Table 8: Overview of Open Problems and Related Protocols for the Hydra Head setting.

Open Problem	Existing Tools
Offline Parties **Section 5.1**	Disclose Cascade Watch Commit [9] Outpost [38] Cerberus [15] PISA [46] Garrison [49] FPPW [48] Counter-Collusion [69] Sleepy Channel [8] Sleepy CRAB [3] Brick [10]
Non-Synchronous Networks **Section 5.2.1**	Sleepy Channel [8] Sleepy CRAB [3] SUBORN Channels [12] MAD-HTLC [66] He-HTLC[67] Bribe & Fork [11] Brick [10]
Fault Tolerant Head Protocol **Section 5.2.2**	Brick [10]
Full Corruption **Section 5.2.3**	Brick [10] ZK-Rollups
Privacy **Section 5.3**	TumbleBit [30] $\mathsf{A^2L^+}$ & $\mathsf{A^2L^{UC}}$ [29] Blind Hub [60]

6 Evaluation of Potential Tools for Payment Channel Networks over Hydra

Here we focus on discussing the applicability of existing tools (Section 3) in a setting where we have multiple Hydra Heads $\mathsf{H}_0, \ldots, \mathsf{H}_n, n \in \mathbb{N}$ and two parties p_0, p_1 distributed arbitrarily in two different Hydra Heads. The problem statement is that p_0 wants to perform a payment to p_1 utilizing Layer 2.

6.1 Network Payments

The Lightning Network [58] can likely be implemented on top of Hydra, which would have the immediate benefit of inheriting the large ecosystem of tools already built around it. However, HTLCs as used in LN are vulnerable to congestion [50], lockdown [53] and wormhole [45] attacks and, moreover, lack privacy (Section 3.1).

A key structural difference between a standard PCN deployed directly on a ledger and a PCN deployed across a series of Hydra Heads is that transactions which would ordinarily go on-chain can instead be processed *within* a Hydra Head at no additional Layer 1 cost. This shifts the cost model underlying existing payment protocols in a non-trivial way. For instance, Sprites' [47] on-chain smart contract — which is responsible for reducing the worst-case locktime from $O(n_{\text{path}})$ to $O(1)$ — becomes an intra-Head contract with negligible cost. Similarly, Blitz's [6] global abort transaction, which in a standard PCN must be confirmed on Layer 1, can instead be confirmed atomically within a Head. These reductions are not free. Both Sprites' contract and Blitz's abort transaction are adjudication mechanisms, and their security derives from being enforceable on a layer the counterparty cannot stall. Relocating them into a Head means they inherit the Head's liveness assumption — and, as discussed in Section 5.1, a single offline participant stalls a Head. The shift should therefore be understood as trading a Layer 1 fee for a weaker enforcement layer, rather than as a strict improvement. Quantifying that trade is part of the next problem.

Open Problem 10 (Intra-Head Atomic Payments) *In a PCN over Hydra, and assuming all nodes of a payment on the PCN are present within the same Hydra Head, on-chain operations required by existing atomic payment protocols can be shifted into a Hydra Head at negligible Layer 1 cost. Which existing protocols (Table 1) benefit from this cost reduction, and can the benefit be formally quantified — for instance in terms of reduced locktime, fewer on-chain transactions, or weaker trust assumptions?*

However, while these cost savings can be observed in an Intra-Head setting, i.e. a payment across channels that are all included within one Hydra Head, it is unclear whether this extends to Cross-Head payments where the sender and receiver of a payment are not present within the same Hydra Head but instead are participating in different ones.

Open Problem 11 (Cross-Head Atomic Payments) *When a cross-Head payment traverses multiple Hydra Heads, each Head along the path can potentially absorb on-chain operations that would otherwise go to Layer 1. However, operations that span Heads — such as a global abort transaction in Blitz [6] or a preimage-conditioned contract in HTLCs — may not be localizable to a single Head. Which operations in existing atomic payment protocols (Table 1) can be decomposed into per-Head operations, and which inherently require cross-Head coordination that cannot be absorbed by any single Head?*

The protocols surveyed in Table 1 differ in their on-chain footprint and thus in how much they stand to gain from the intra-Head cost shift described above. As a concrete next step, a systematic comparison of these protocols under the Hydra cost model — quantifying locktime reduction, on-chain transaction count, and trust assumption changes — would directly address both Open Problem 10 and Open Problem 11. SyncPCN/P-SyncPCN [26] deserves particular attention in this comparison, as it is the only protocol in Table 1 designed for settings where Δ-Liveness may fail, making it especially relevant to Open Problem 4.

Orthogonal to the choice of payment protocol, the topology of the underlying PCN significantly affects routing efficiency and resilience. Hydra Heads are multi-party, meaning a single Head embeds a clique in the PCN graph. This creates a distinctive topology in which nodes within the same Head share a correlated failure mode: if one participant goes offline, the Head stalls and all channels within it become temporarily unusable. At the same time, multi-party Heads reduce the total number of channels needed to connect a given set of parties. Tang et al. [64] propose forming a PCN cooperatively into Ramanujan graphs with provably good expansion, diameter, and resilience properties, but such structured topologies are unlikely to emerge naturally — in practice, PCNs converge toward power-law degree distributions. Promoting a more desirable topology may require incentive mechanisms that compensate parties for creating channels in topologically beneficial but individually suboptimal positions. Thus, we consider the following open problem:

Open Problem 12 (Incentivized Network Topology) *What network topologies balance the connectivity benefits of multi-party Hydra Heads against their correlated failure risk? Specifically, which graph-theoretic properties — such as bounded diameter, high expansion, or low clustering — are most beneficial for a PCN over Hydra, and can incentive mechanisms be designed, drawing on the approach of Tang et al. [64] and game-theoretic tools, to promote topologies exhibiting these properties?*

For both the payment protocol and topology questions, protocols for multi-path payments [1, 24, 54] are complementary: they improve PCN utilization regardless of which atomic payment protocol or topology is chosen, and their interaction with Hydra's multi-party structure is itself an open question.

6.2 Routing

When creating a large PCN, it is necessary to find routes for payments in a scalable manner. A PCN tends to be unreliable. At any time nodes and channels can disappear, reappear or be newly formed and channel capacities are ever-changing. In addition, routing in a PCN over Hydra presents a structurally novel challenge absent from standard PCNs: when a party goes offline, all channels of the Hydra Head in which it participates simultaneously become unavailable. This creates correlated failures across multiple edges of the PCN graph — a phenomenon that existing routing protocols, designed for single-channel failure modes, do not handle. Routing protocols must, therefore, be evaluated not only for scalability and privacy but also for resilience to correlated multi-edge failures. Lastly, scalability is required to allow route discovery for light nodes.

We consider the following open problem:

Open Problem 13 (Routing across Hydra) *Is there a routing protocol for PCNs that is (a) scalable to light nodes, (b) secure against byzantine nodes, (c) privacy-preserving, and (d) resilient to the correlated multi-channel failures that arise from Hydra Head node failures?*

If we consider choosing a suitable network topology for Hydra Networks as described in Section 6.1 and Open Problem 12, does the choice in topology affect the choice of routing protocol? More specifically:

Open Problem 14 (Clique-Exploiting Routing) *Can the clique topology of Hydra Heads — where every pair of participants is connected through the shared Head channel — be exploited to design routing protocols with provably better performance than in standard two-party channel PCNs?*

Orthogonal to routing, multi-path payments [1, 54, 24] are technologies that improve utilization of a PCN, enabling routing of payments that are much larger than channel capacities present in PCNs, and moreover, enabling utilization of channels with small capacities. Another orthogonal aspect, as discussed in Section 6.1, is promoting and utilizing network topologies with beneficial properties [64] to promote network utilization and decentralization. While protocols based on traditional routing protocols [59, 43, 61, 31] cannot directly make use of multi-path payments, existing flow based routing protocols [62, 64, 68] that can do so are either not scalable or are insecure as discussed in Section 3.3. Thus, we consider the following open problem:

Open Problem 15 (Flow Routing) *Is there a scalable and secure flow based routing protocol? Is such a protocol suitable to be used with PCNs across Hydra Heads?*

While having multiple nodes within a PCN present within one Hydra Head makes it more vulnerable to a single point of failure in the case just one node goes offline, this structure presents an interesting topology: Hydra's multi-party Heads create a PCN that is naturally modeled as a hypergraph [19], where each hyperedge connects all participants of a Head. Existing hypergraph path-finding [19] is neither scalable nor privacy-aware, while existing scalable and privacy-preserving routing protocols [43, 61] assume pairwise channels. Thus we consider the following open problem:

Open Problem 16 (Routing in Multi-Party Channel Networks) *Can a routing protocol for multi-party channel networks be designed that is simultaneously scalable (logarithmic storage and message complexity), privacy-preserving (hiding sender, receiver and payment amount from intermediaries), and capable of exploiting the multi-party structure of Heads to improve routing efficiency?*

As a starting point, for research into routing protocols we recommend Silent Whispers [43] and Speedy Murmurs [61] as they are both scalable and privacy-preserving. In addition, protocols for multi-path payments [1, 54, 24] improve the usability of PCNs. Lastly, creation of a PCN with a decentralized topology [64] has orthogonal benefits.

6.3 Re-Balancing Channels

Channel imbalance is an inherent consequence of directional payment flows in a PCN. Re-balancing protocols address this by routing circular payments that restore balance without net transfers of value. All existing re-balancing protocols — REVIVE [39], Hide & Seek [13], Musketeer [14], Proactive Rebalancing [55], Wiser [65] — are designed for standard two-party PCNs, and their applicability to a PCN over Hydra requires analysis. Moreover, re-balancing protocols, require computations on a channel's current balance. Privacy preserving techniques should be considered to avoid leakage of this information. Another observation is that we can consider two cases: One in which a re-balancing circuit is fully contained within a Hydra Head, and one case where it is not. Within a Head, all parties co-sign every transaction, which means a re-balancing payment can be confirmed atomically with other Head transactions at no additional on-chain cost. Across Heads, however, a re-balancing circuit must traverse multiple Heads and may require cross-Head atomicity, which reintroduces the standard challenges of atomic multi-hop payments. This distinction motivates two separate problems:

Open Problem 17 (Intra-Head Re-Balancing) *Can the multi-party co-signature structure of Hydra Heads be exploited to perform re-balancing more efficiently — with fewer messages or weaker cryptographic assumptions — than in standard two-party channel PCNs?*

Open Problem 18 (Cross-Head Re-Balancing) *Can privacy-preserving re-balancing circuits be executed across multiple Hydra Heads, and what are the minimal atomicity guarantees required for such circuits to be secure?*

For this problem, Hide & Seek [13] and Musketeer [14] provide starting points for the privacy-preserving component. Shaduf [27] is also relevant: it is the first re-balancing protocol that does not require circular paths, which may be beneficial in sparse inter-Head topologies where cycles are hard to find, at the cost of additional on-chain interaction. On top of that, Shaduf and other Re-Balancing techniques that require on-chain Interaction can benefit if the required on-chain interactions are instead done on Layer 2, making them a viable approach in the Intra-Head Re-Balancing case (Open Problem 17).

6.4 Privacy

While Lightning Network payments by default are not private, several approaches exist to improve privacy [44, 45] within a PCN. Privacy in a PCN over Hydra has two distinct dimensions. The first is payment privacy: hiding the sender, receiver, and amount of payments from intermediary Heads and external observers. The second is audit privacy: allowing authorized parties to verify correctness while preserving privacy against unauthorized observers. First, we consider payment privacy with the following open problem:

Open Problem 19 (Private Payments in PCNs over Hydra) *Can privacy-preserving payment protocols — satisfying at minimum relationship privacy and value privacy — be implemented over a PCN of Hydra Heads, compatible with Hydra's EUTxO model?*

AMHL [45] is the strongest starting point: it provides relationship privacy and resists the wormhole attack. Twilight [20] offers a qualitatively different approach based on differential privacy, providing a rigorous probabilistic guarantee rather than a cryptographic one, at the cost of relying on trusted execution environments.

Second, we consider audit privacy. While privacy preserving audits are available for Hydra Heads [16], it is not clear whether that can be extended to audits over PCNs. Thus we consider the following open problem:

Open Problem 20 (Privacy-Preserving Audits over PCNs) *Can the single-Head audit protocol of [16] be extended to support audits across a PCN of Hydra Heads, where each Head has a different participant set and audit scope?*

This problem is significantly harder than the single-Head case. An auditor must aggregate evidence from multiple Heads without learning information about Heads it is not authorized to audit. The multi-Head setting also raises questions about which party has the authority to authorize an audit, and whether the audit protocol must be agreed upon at channel setup or can be imposed retroactively. These questions have no precedent in the existing literature.

The earlier two problems, namely "privacy-preserving payments" and "privacy-preserving audits" have directly conflicting requirements: the former aims to hide transaction details from all observers, while the latter aims to reveal them to authorized parties. Any solution must address this conflict, likely through selective disclosure mechanisms.

6.5 Summary

Lastly, we summarize and give an overview of the open problems we identify for employing a PCN on top of Hydra. For each open problem we list the existing tools that address such problems in other Layer 2 constructions, thus are a candidate to be extended for the use with Hydra.

7 Evaluation of Potential Tools for State Machines across a Hydra Network

Here we focus on discussing the applicability of existing tools (Section 3) in a setting where we have multiple Hydra Heads $\mathsf{H}_0, \ldots, \mathsf{H}_m$ and n_H parties $p_0, \ldots, p_{n_H}$, $m \in \mathbb{N}, m \geq 2$ arbitrarily distributed among all Hydra Heads with each Hydra Head having at least two parties participating. The problem statement is that $p_0, \ldots, p_{n_H}$ jointly execute a state machine through an Interhead [36] or an Ad-Hoc Hydra Head constructions [33]. In contrast to tools for channels (Section 5) and PCNs (Section 6) there are few existing tools for use cases similar to the Interhead and Ad-Hoc Hydra Head constructions. The closest to this use case are virtual channels. However, we note that virtual channels are akin to regular channels albeit without requiring interaction with Layer 1. Thus, in the case of a virtual Hydra Head, i.e. an Interhead, the same tooling as for Hydra Heads (Section 5) could be used for Interheads.

Thus, we consider the following open problem:

Open Problem 21 (Tool Transfer to Interheads) *Which of the tools developed for single Hydra Heads in Section 5 can be extended to Interheads, and what are the precise structural obstacles to those that cannot?*

Investigating the above problem includes studying the structural properties of Interheads and identifying whether these properties block the use of specific tools, and decide whether those blocks are fundamental. For example, a Watchtower for a single Head monitors one dispute condition; an Interhead Watchtower must monitor dispute conditions that span multiple Heads with different participant sets. This is a fundamental change that requires new protocol design, not just an extension of existing work.

Table 9: Overview of Open Problems and Related Protocols for the PCN over Hydra setting.

Open Problem	Existing Tools
Network Payments **Section 6.1**	Managed Network Topology [64] Lightning and HTLCs [58] MAD-HTLC [66] Payment Trees [35] Sprites [47] Blitz [6] AMHL [45] Thora [2] SyncPCN/PSyncPCN [26]
Routing **Section 6.2**	Silent Whispers [43] Speedy Murmurs [61] Flare [59] Spider Network [62] AODV [31] Tang et al. [64] Robustpay+ [68] SPRITE (Panwar et al.) [52] RACED [41] LightPIR [56] Corcoran et al. [19] Atomic Multi-Path Payments [1] Non-Atomic Multi-Path Payments [54, 24]
Re-balancing Channels **Section 6.3**	REVIVE [39] Proactive Rebalancing [55] Hide & Seek [13] Musketeer [14] Shaduf [27] Cycle [32] Wiser [65]
Privacy **Section 6.4**	Fulgor & Rayo [44] AMHL [45] Twilight [20] Audits [16]

7.1 Channel Stability and Watchtower

While Watchtowers have been implemented for individual channel constructions, we are not aware of any Watchtower construction that can be used for virtual channels such as the Interhead. Moreover, it is unclear how Watchtowers extend to Ad-Hoc Hydra Heads. Thus, let us introduce the following open problem:

Open Problem 22 (Watchtower for Virtual and Interhead Channels) *Can a Watchtower construction be designed for Interheads, that provides balance security to offline parties as well as accounts for the multi-Head dispute logic and the absence of a single on-chain UTXO representing the Interhead's state?*

7.2 Liquidity

Interheads and Ad-Hoc Hydra Heads are distinct from PCNs, but are nonetheless applications on top of a channel network that redistribute coins. Thus, it raises the question of whether individual parties could run into liquidity issues and whether there are tools that could alleviate it. Thus, we consider the following open problem:

Open Problem 23 (Liquidity Management in Interheads) *What liquidity constraints does an Interhead construction impose on the underlying channel network, and can existing re-balancing techniques be adapted to alleviate liquidity bottlenecks in this setting?*

Interheads and Ad-Hoc ledger, like PCNs, require submission of collateral by intermediaries. Whereas in PCNs collateral has to be committed for payments performed, collateral in Ad-Hoc ledger has to be committed for each UTxO moved into the Ad-Hoc ledger. While, for a one-time payment, the required collateral for a payment in a PCN using HTLCs over one hop is equal to the required collateral in an ad-hoc ledger that connects two Hydra Heads, these are significantly different approaches to handling collateral. Thus, existing PCN re-balancing protocols are not directly applicable and must be adapted to address potential liquidity issues or entirely new techniques might need to be found.

On top of that, when considering the setting where the Interhead or Ad-Hoc ledger constructions are done recursively to create an Interhead or Ad-Hoc ledger across non-adjacent Hydra Heads, additional liquidity issues occur: If done naively, then each recursive construction requires committing additional collateral. For example,

in the regular Ad-Hoc ledger construction, each coin in the Ad-Hoc ledger is backed by exactly one coin of collateral per Hydra Head. However, if we use that Ad-Hoc ledger to construct another one recursively on top, then it would require to commit another coin of collateral, i.e. each coin in the Ad-Hoc ledger on top is now backed by two coins of collateral. This raises the question whether it is possible to improve on the efficiency of recursive Interhead and Ad-Hoc ledger construction to ensure that each coin is backed by at most one coin of collateral in each Hydra Head. More concretely we consider the following open problem:

Open Problem 24 (Recursive Collateral) *Can the required collateral for each coin within an Interhead or Ad-Hoc ledger be bound by at most one coin, even in the case of recursive Interhead and Ad-Hoc ledger construction?*

Another issue that can be revisited when considering the Interhead and Ad-Hoc Ledger setting is Re-Balancing of channels in a PCN running on top of Hydra as discussed in Section 6.3. Re-Balancing protocols such as Shaduf [27] avoid the requirement of having a circuit on which Re-Balancing can take place on the PCN. However, as a tradeoff they require some additional on-chain interaction. We note that this additional cost can be mitigated by having it done in Layer 2. Whereas executing a protocol such as Shaduf within a Hydra Head is a potential approach for Intra-Head Re-Balancing (Open Problem 17), executing such a protocol on top of an Ad-Hoc ledger can be a potential approach for Cross-Head Re-Balancing (Open Problem 18):

Open Problem 25 (Cross-Head Re-Balancing using Ad-Hoc Ledgers) *Can the Ad-Hoc ledger construction be used to allow for off-chain channel Re-Balancing, even if there does not exist a valid re-balancing circuit?*

8 Conclusion

In this work we gave an overview of and compared related work, and used these to derive open problems for three settings in which Hydra can be deployed. For each set of open problems within a particular topic we propose related sources as a starting point for further research. Concretely, we surveyed existing tools for Layer 2 payment channel networks and evaluated their applicability to Hydra across three increasingly complex settings:

- The single Hydra Head (Section 5);
- The PCN formed over multiple Hydra Heads (Section 6); and
- The state machine execution across a Hydra network via Interhead and Ad-Hoc Ledger constructions (Section 7).

For each setting we identified open problems and proposed existing work that can serve as a starting point for further research, arriving at 25 open problems in total.

Cross-cutting themes. Several cross-cutting themes emerged from this analysis. First, Hydra's requirement that all parties co-sign every transaction creates a fundamental tension between security and practicality that recurs across settings: it provides trustless security (Open Problem 5) but causes liveness failures when any party goes offline (Open Problem 1), and it makes intra-Head privacy structurally impossible against co-participants (Open Problem 8). Relaxing this unanimity requirement — whether through BFT (Open Problem 5), incentive mechanisms (Open Problem 6), or zero-knowledge techniques (Open Problem 7) — is arguably the most consequential design decision for Hydra's future, as it would simultaneously address offline tolerance, liveness, and potentially privacy.

Hydra-specific structure. Hydra's multi-party structure introduces a distinctive feature absent from standard two-party PCNs: correlated channel failures, where a single offline participant disables all channels in a Head. This affects routing (Open Problem 13), topology design (Open Problem 12), and re-balancing (Open Problems 17 and 18), and no existing protocol accounts for it. Moreover, the on-chain-to-intra-Head cost shift (Open Problems 10 and 11) may alter the practical tradeoffs that have historically favored HTLCs over more sophisticated alternatives. A systematic evaluation of existing atomic payment protocols under the Hydra cost model — quantifying locktime, on-chain footprint, and trust assumption changes — would be a concrete and tractable contribution.

More on the proposed research agenda. Among the 25 problems, we highlight several as particularly tractable or high-impact starting points for further research. Determining a rigorous value for Cardano's Δ (Open Problem 2) is primarily empirical and could be addressed with existing blockchain analysis tools. Designing a Watchtower for Hydra (Open Problem 1) can build directly on Cerberus [15] and FPPW [48], which already satisfy balance security. Evaluating existing payment protocols in the Hydra cost model (Open Problems 10 and 11) requires comparative analysis rather than new protocol design. At the other end of the spectrum, achieving state machine privacy (Open Problem 9), designing routing for multi-party channel hypergraphs (Open Problem 16), and bounding recursive collateral (Open Problem 24) appear to require fundamentally new techniques with limited existing work to build on.

Final remarks. We stress that Sections 5, 6, and 7 of this work focus on identifying problems and surveying relevant tools, but do not propose solutions. The open problems are intended as a research agenda for the community, and we hope that the structured comparison of existing tools provided here — together with the explicit identification of what is and is not directly transferable to Hydra — will accelerate progress on these questions.

References

1. [lightning-dev] amp: Atomic multi-path payments over lightning. https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html. Accessed: 2026-10-20. 5, 9, 10, 24, 25, 27
2. Lukas Aumayr, Kasra Abbaszadeh, and Matteo Maffei. Thora: Atomic and privacy-preserving multi-channel updates. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 165–178, 2022. 4, 6, 27
3. Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and Subhra Mazumdar. Securing lightning channels against rational miners. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 393–407, 2024. 12, 13, 14, 19, 20, 23
4. Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 635–664, Cham, 2021. Springer International Publishing. 6
5. Lukas Aumayr, Matteo Maffei, Oğuzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. Bitcoin-compatible virtual channels. In *2021 IEEE Symposium on Security and Privacy*, pages 901–918. IEEE, 2021. 8
6. Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure {Multi-Hop} payments without {Two-Phase} commits. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4043–4060, 2021. 4, 6, 23, 24, 27
7. Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Breaking and fixing virtual channels: Domino attack and donner. In *Network and Distributed System Security (NDSS) Symposium*, 2023. 8
8. Lukas Aumayr, Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Pedro Moreno-Sanchez, and Matteo Maffei. Sleepy channels: Bi-directional payment channels without watchtowers. In *2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 179–192, 2022. 12, 14, 19, 23
9. Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. Towards secure and efficient payment channels. *arXiv preprint arXiv:1811.12740*, 2018. 4, 12, 14, 15, 23
10. Zeta Avarikioti, Eleftherios Kokoris-Kogias, Roger Wattenhofer, and Dionysis Zindros. Brick: Asynchronous incentive-compatible payment channels. In *International Conference on Financial Cryptography and Data Security*, pages 209–230. Springer, 2021. 13, 14, 19, 20, 21, 23
11. Zeta Avarikioti, Paweł Kędzior, Tomasz Lizurej, and Tomasz Michalak. Bribe & fork: Cheap bribing attacks via forking threat, 2024. 5, 20, 23
12. Zeta Avarikioti and Orfeas Stefanos Thyfronitis Litos. Suborn channels: Incentives against timelock bribes. Cryptology ePrint Archive, Paper 2022/814, 2022. 5, 13, 20, 23
13. Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. Hide & seek: Privacy-preserving rebalancing on payment channel networks. In *Financial Cryptography and Data Security*, pages 358–373. Springer, 2022. 15, 16, 17, 25, 27
14. Zeta Avarikioti, Stefan Schmid, and Samarth Tiwari. Musketeer: Incentive-compatible rebalancing for payment channel networks. In *Advances in Financial Technologies 2024*, pages 13:1–13:22, 2024. 15, 17, 25, 27
15. Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. Cerberus channels: Incentivizing watchtowers for bitcoin. In *Financial Cryptography and Data Security 2020*, pages 346–366. Springer, 2020. 4, 12, 14, 19, 23, 28
16. Christian Badertscher, Maxim Jourenko, Dimitris Karakostas, and Mario Larangeira. Scalable and lightweight state-channel audits. In *International Conference on Cryptology and Network Security*, pages 264–288. Springer, 2024. 18, 26, 27
17. Manuel M. T. Chakravarty, Sandro Coretti, Matthias Fitzi, Peter Gazi, Philipp Kant, Aggelos Kiayias, and Alexander Russell. Fast isomorphic state channels. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part II*, volume 12675 of *LNCS*, pages 339–358. Springer, Berlin, Heidelberg, March 2021. 2, 3, 5, 8, 13, 18, 20
18. David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203. Springer, 1983. 17
19. Padraig Corcoran and Rhyd Lewis. Path planning in payment channel networks with multi-party channels. *Distrib. Ledger Technol.*, 4(4), October 2025. 11, 12, 25, 27
20. Maya Dotan, Saar Tochner, Aviv Zohar, and Yossi Gilad. Twilight: A differentially private payment channel network. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 555–570, Boston, MA, August 2022. USENIX Association. 17, 26, 27
21. Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In *Theory and Applications of Cryptographic Techniques*, pages 625–656. Springer, 2019. 3, 8
22. Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *Perun: Virtual Payment Hubs over Cryptocurrencies*. IEEE, 2017. 8

23. Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 949–966. ACM, 2018. 6, 8
24. Stefan Dziembowski and Paweł Kędzior. Non-atomic payment splitting in channel networks. In *5th Conference on Advances in Financial Technologies (AFT 2023)*, pages 17–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023. 5, 9, 10, 24, 25, 27
25. Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 801–815, 2019. 5, 6
26. Oguzhan Ersoy, Jérémie Decouchant, Satwik Prabhu Kumble, and Stefanie Roos. Syncpcn/psyncpcn: Payment channel networks without blockchain synchrony. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 16–29, 2022. 6, 7, 14, 24, 27
27. Zhonghui Ge, Yi Zhang, Yu Long, and Dawu Gu. Shaduf: Non-cycle payment channel rebalancing. In *NDSS*, 2022. 15, 17, 25, 27, 28
28. Paul Gerhart, Daniel Rausch, and Dominique Schröder. Universally composable adaptor signatures. Cryptology ePrint Archive, Paper 2025/1363, 2025. 18
29. Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. Foundations of coin mixing services. In *Proceedings of the 2022 ACM SIGSAC conference on computer and communications security*, pages 1259–1273, 2022. 17, 18, 22, 23
30. Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017. 17, 18, 22, 23
31. Philipp Hoenisch and Ingo Weber. Aodv–based routing for payment channel networks. In Shiping Chen, Harry Wang, and Liang-Jie Zhang, editors, *Blockchain – ICBC 2018*, pages 107–124, Cham, 2018. Springer International Publishing. 9, 11, 12, 25, 27
32. Zicong Hong, Song Guo, Rui Zhang, Peng Li, Yufen Zhan, and Wuhui Chen. Cycle: Sustainable off-chain payment channel network with asynchronous rebalancing. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 41–53. IEEE, 2022. 15, 17, 27
33. Maxim Jourenko and Mario Larangeira. State machines across isomorphic layer 2 ledgers. In *Financial Cryptography and Data Security*. Springer, 2023. 6, 8, 18, 26
34. Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Lightweight virtual payment channels. In *Cryptology and Network Security*, pages 365–384. Springer, 2020. 8
35. Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Payment trees: Low collateral payments for payment channel networks. In *Financial Cryptography and Data Security 2021*, pages 189–208. Springer, 2021. 4, 5, 6, 27
36. Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Interhead hydra: Two heads are better than one. In *Mathematical Research for Blockchain Economy*, pages 187–212. Springer, 2022. 6, 8, 18, 26
37. Maxim Jourenko, Xiangyu Su, Adam Blatchley Hansen, and Mario Larangeira. Universally composable treatment of multi-party isomorphic state channels. Cryptology ePrint Archive, Paper 2025/1517, 2025. 8, 18, 22
38. Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. Outpost: A responsive lightweight watchtower. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 31–40, 2019. 4, 12, 14, 23
39. Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 439–453. ACM, 2017. 15, 16, 17, 25, 27
40. Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. Elmo: Recursive virtual payment channels for bitcoin. Cryptology ePrint Archive, Paper 2021/747, 2021. 8
41. Kartick Kolachala, Mohammed Ababneh, and Roopa Vishwanathan. Raced: Routing in payment channel networks using distributed hash tables, 2023. 11, 12, 27
42. Ranjit Kumaresan, Duc V. Le, Mohsen Minaei, Srinivasan Raghuraman, Yibin Yang, and Mahdi Zamani. Programmable payment channels. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 51–73, Cham, 2024. Springer Nature Switzerland. 6
43. Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in credit networks. NDSS, 2017. 9, 11, 12, 25, 27
44. Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 455–471. ACM Press, October / November 2017. 4, 6, 15, 26, 27
45. Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019*, 2019. 4, 6, 15, 23, 26, 27
46. Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. Pisa: Arbitration outsourcing for state channels. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 16–30, 2019. 12, 13, 14, 15, 23
47. Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *Financial Cryptography and Data Security*, pages 508–526. Springer, 2019. 4, 6, 23, 27
48. Arash Mirzaei, Amin Sakzad, Jiangshan Yu, and Ron Steinfeld. Fppw: A fair and privacy preserving watchtower for bitcoin. In *Financial Cryptography and Data Security 2021*, pages 151–169. Springer, 2021. 12, 14, 19, 23, 28
49. Arash Mirzaei, Amin Sakzad, Jiangshan Yu, and Ron Steinfeld. Garrison: a novel watchtower scheme for bitcoin. In *Australasian Conference on Information Security and Privacy*, pages 489–508. Springer, 2022. 12, 14, 23

50. Ayelet Mizrahi and Aviv Zohar. Congestion attacks in payment channel networks. In *International conference on financial cryptography and data security*, pages 170–188. Springer, 2021. 4, 7, 23
51. Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Timelocked bribing. In *International Conference on Financial Cryptography and Data Security*, pages 53–72. Springer, 2021. 20
52. Gaurav Panwar, Roopa Vishwanathan, George Torres, and Satyajayant Misra. Sprite: Secure and private routing in payment channel networks. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '24, page 1878–1894, New York, NY, USA, 2024. Association for Computing Machinery. 9, 11, 12, 27
53. Cristina Pérez-Sola, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. Lockdown: Balance availability attack against lightning network channels. In *International conference on financial cryptography and data security*, pages 245–263. Springer, 2020. 4, 7, 23
54. Dmytro Piatkivskyi and Mariusz Nowostawski. Split payments in payment networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 67–75. Springer, 2018. 9, 10, 24, 25, 27
55. Rene Pickhardt and Mariusz Nowostawski. Imbalance measure and proactive channel rebalancing algorithm for the lightning network. In *2020 IEEE International Conference on Blockchain and Cryptocurrency*, pages 1–5. IEEE, 2020. 15, 17, 25, 27
56. Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, and Michelle Yeo. Lightpir: Privacy-preserving route discovery for payment channel networks, 2021. 12, 27
57. Andrew Poelstra. Scriptless script. Presentation slides: https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-05-milan-meetup/slides.pdf. Accessed: 2026-10-20. 18
58. Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *See https://lightning. network/lightning-network-paper. pdf*, 2016. 3, 5, 6, 23, 27
59. Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. *White Paper (bitfury.com/content/5-white-papers-research/whitepaper_flare_ an_ approach_ to_ routing_ in_ lightning_ network_ 7_ 7_ 2016.pdf)*, 2016. 9, 11, 12, 25, 27
60. Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, Oğuzhan Ersoy, Amin Sakzad, Muhammed F. Esgin, Joseph K. Liu, Jiangshan Yu, and Tsz Hon Yuen. Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2462–2480, 2023. 17, 18, 21, 22, 23
61. Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017. 9, 11, 12, 25, 27
62. Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, Giulia Fanti, and Pramod Viswanath. Routing cryptocurrency with the spider network. *arXiv preprint arXiv:1809.05088*, 2018. 9, 11, 12, 25, 27
63. Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A^2L: Anonymous atomic locks for scalability in payment channel hubs. In *2021 IEEE symposium on security and privacy (SP)*, pages 1834–1851. IEEE, 2021. 17, 18
64. Shuyang Tang and Sherman S. M. Chow. Strengthening multi-hop channels via strategic mesh connections. In *Financial Cryptography and Data Security 2025*. Springer, 2025. 5, 7, 10, 11, 12, 24, 25, 27
65. Samarth Tiwari, Michelle Yeo, Zeta Avarikioti, Iosif Salem, Krzysztof Pietrzak, and Stefan Schmid. Wiser: Increasing throughput in payment channel networks with transaction aggregation. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 217–231, 2022. 15, 17, 25, 27
66. Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. Mad-htlc: because htlc is crazy-cheap to attack. In *2021 IEEE symposium on security and privacy (SP)*, pages 1230–1248. IEEE, 2021. 4, 5, 6, 20, 23, 27
67. Sarisht Wadhwa, Jannis Stoeter, Fan Zhang, and Kartik Nayak. He-htlc: Revisiting incentives in htlc. In *NDSS*, 2023. 5, 6, 20, 23
68. Yuhui Zhang and Dejun Yang. Robustpay+: Robust payment routing with approximation guarantee in blockchain-based payment channel networks. *IEEE/ACM Transactions on Networking*, 29(4):1676–1686, 2021. 9, 10, 11, 12, 25, 27
69. Yuhui Zhang, Dejun Yang, Guoliang Xue, and Ruozhou Yu. Counter-collusion smart contracts for watchtowers in payment channel networks. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021. 12, 14, 23