SoK: Off The Chain Transactions

Lewis Gudgeon
Imperial College London
l.gudgeon18@imperial.ac.uk

Pedro Moreno-Sanchez
TU Wien
pedro.sanchez@tuwien.ac.at

Stefanie Roos
TU Delft
s.roos@tudelft.nl

Patrick McCorry
King's College London
patrick.mccorry@kcl.ac.uk

Arthur Gervais
Imperial College London, Liquidity Network,
Lucerne University of Applied Sciences and Arts
a.gervais@imperial.ac.uk

***Abstract*—Blockchains have the potential to revolutionize markets and services, yet, currently exhibit high latencies and fail to handle loads comparable to those managed by traditional custodian financial systems. *Layer-two* protocols, built on top of (*layer-one*) blockchains, avoid disseminating every transaction to the whole network by sending transactions *off-chain* and instead utilize the blockchain only as a recourse for disputes. The promise of layer-two protocols is to complete transactions in sub-seconds, reduce fees, and allow blockchains to scale.**

With this Systematization of Knowledge, we are the first to structure the complete rich and multifaceted body of research on layer-two transactions. Categorizing the research into payment and state channels as well as commit-chains, we provide a comparison of the protocols and their properties. We contribute a systematization of the associated synchronization and routing protocols along with their privacy and security aspects. Contrary to common belief in the blockchain community, we show that layer-two can scale blockchains; that layer-two protocols are secure without full collateralization; that privacy of layer-two transaction is not granted by default; and that fees depend on the transmitted transaction value. The SoK clears the layer-two fog, highlights the potential of layer-two solutions and identifies their unsolved challenges and promising avenues of future work.

I. Introduction

The advent of blockchains over a decade ago [1]–[4] spurred rapid and extensive innovation across different scientific disciplines. Blockchains offer a mechanism through which mutually mistrusting entities can cooperate in the absence of a trusted third party. However, the use of broadcast in those non-custodial protocols limits their scalability to about ten transactions-per-second (tps) [5], [6], compared to custodian payment systems with thousands of tps [7]. Scaling limitations and transaction latencies have led to a rich literature corpus exploring different blockchain scaling solutions: (i) alternative blockchain consensus architectures [8]–[17], (ii) sharding [18]–[20] and (iii) side-chains [21], some of which were systematized in related work [22].

Layer-two protocols are an orthogonal scaling solution. Contrary to the prior-mentioned solutions, layer-two protocols scale blockchains *without* changing the layer-one trust assumptions and they do not introduce additional consensus mechanisms. Backward compatibility is crucial for widely adopted blockchains, because once deployed, a blockchain's consensus mechanism is challenging to modify due to its decentralized structure. Consensus changes might even lead to different, forked systems [23]. Layer-two protocols enable users to perform so-called *off-chain* transactions through private communication, rather than broadcasting the transaction on the (parent) blockchain. This optimization reduces the transaction load on the underlying blockchain and is fully backward compatible. The theoretical transaction throughput is only bounded by the communication bandwidth and latency of the involved parties. Off-chain transaction security can be guaranteed via allocated collateral, e.g. in payment channel designs [24]–[27] or by offering delayed transaction finality in commit-chain proposals [28].

A. This Systematization of Knowledge

A rich body of literature has emerged on off-chain protocols, proposing payment [24]–[27], [29], state [30] and virtual [31] channels, payment channel networks (PCNs) [27], [29] and related routing protocols [32]–[37], channel rebalancing [38] and channel factories [39] constructions, commit-chains [28], [40], channel hubs [41], [42], privacy-enhancing channels [41], [43]–[45]. However, the sources of information about layer-two protocols are highly disparate. Moreover, in part due to the rapid pace of advancement in the blockchain field, we observe, mostly outside academia, a frequent under-specification of constructions and their adversarial assumptions. This makes it exceedingly difficult to discern thought-through concepts from marketing activities. We aim to clear the fog surrounding layer-two protocols, equipping newcomers to this inaccessible field with a concise reference, and inform the directions of future work. This SoK provides a systematic overview of layer-two systems and identifies the complete set of proposed layer-two protocol types. We scrutinize the following myths.

Myth 1: blockchains cannot scale significantly — either in terms of throughput and computational complexity — without advances at *layer-one*, such as through novel or more efficient consensus mechanisms.

Myth 2: layer-two solutions can only be secure if the off-chain transaction volume is fully collateralized.

Myth 3: by default, off-chain transactions offer privacy.

Myth 4: blockchain transaction fees depend on their size or computational complexity, not on the transaction value.

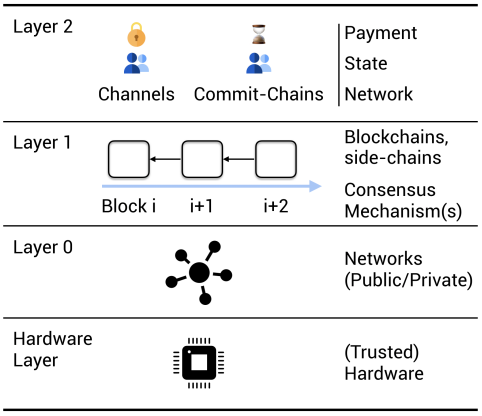

Fig. 1. Suggested blockchain layers. Layer-two channels and commit-chains operate without additional consensus mechanism and transact payments, state, and spawn networks.

This SoK is structured as follows. Section II outlines the necessary background followed by different layer-two design classes, *channels* in Section III and *commit-chains* in Section IV. Section V considers the anonymity and privacy aspects of layer-two protocols, Section VI covers security properties and we conclude the paper in Section VII.

II. Blockchains and Off-Chain Transactions

This section establishes the necessary background and isolates the blockchain components relevant to layer-two. The background presented here is necessarily not a complete overview of blockchain-related concepts, which have been surveyed in others SoK [22], [46]. We distinguish between four different layers within a blockchain system: the *hardware*, *layer-zero*, *layer-one* and *layer-two* (cf. Figure 1).

a) Hardware Layer: Trusted Execution Environments (TEE) substitute the need for a blockchain clock with a trusted hardware assumption, thus enabling efficient protocols at other layers such as off-chain payments [47], [48], the removal of dispute processes and backward compatibility [49]. TEE (e.g. Intel SGX) execute sensitive or security-critical application code within *enclaves* [50], [51], tamper-proof from the operating system or other higher-privileged software.

b) The Network Layer: The network layer, or layer-zero, is typically a peer-to-peer layer on which blockchain nodes exchange information asynchronously [52]. The network layer is of utmost importance to the scalability [53], [54], security [5] and privacy [55] of a blockchain. An efficient layer-zero enables higher transaction throughput and stronger resilience against malicious actors [5]. Blockchain miners, who write transactions to the blockchain, are connected through dedicated *miner* P2P networks (e.g. Fibre [56]), in addition to the public blockchain P2P network.

c) The Blockchain Layer: Layer-one hosts an immutable append-only chain of blocks that accumulates transactions from parties in a network [46]. Each transaction encodes an update of the state of the blockchain. A transaction can exchange digital assets between parties or invoke an application (i.e. smart contract). The integrity of the blockchain is ensured by means of a consensus algorithm executed across participants. Consensus algorithms rely on e.g. the computationally expensive Proof-of-Work (PoW) [1], [17], [57]–[59] or a large number of alternatives [12], [13], [60]–[64]. Blockchains can be permissionless or permissioned depending on whether participation is open or restricted. We focus on permissionless blockchains as permissioned blockchains lack the non-custodial property, but layer-two concepts apply equally to permissioned blockchains. Crucial for the design of layer-two protocols is the scripting language of the underlying blockchain. Bitcoin-like blockchains are based on a restricted Script language [1] and operate via a set of Unspent Transaction Outputs (UTXO), while other blockchains support Turing-complete languages enabling highly expressive smart contracts [2]. Layer-two protocols typically assume two properties from the blockchain layer: *integrity* (i.e. only valid transactions are added to the ledger) and *eventual synchronicity with an upper time-bound* (i.e. a valid transaction is eventually added to the ledger, before a critical timeout).

d) The Off-chain Layer: We define off-chain or layer-two protocols as protocols that i) do not publish every transaction on the blockchain immediately (contrary to *on-chain transactions*) and ii) entirely rely on the consensus algorithm of a *parent-chain*. Off-chain protocols come in two flavors. Channels are formed between n coequal parties whereas commit-chains rely on one central but untrusted intermediary. Side-chains [21] do not classify as layer-two due to having their own consensus algorithm.

III. Channels

A channel establishes a private peer-to-peer medium, governed by pre-set rules, e.g. a smart contract, allowing the involved parties to consent to state updates unanimously by exchanging authenticated state transitions off-chain. We provide an overview of state-of-the-art channel constructions in Table I where we distinguish between two channel techniques:

- Payment channels: Off-chain payment interactions.
- State channel: Off-chain arbitrary interactions.

Payment channels emerged [24] to support rapid one-way payments, then transitioned towards bi-directional channel designs [27], where both parties can issue and receive payments. *State channels* [30] generalize the concept to support the execution of arbitrary state transitions.

A. Channel Overview

A channel allows n parties to agree, via *unanimous* consent, to a new state of a previously agreed smart contract. A channel's lifetime consists of three phases: *(i)* channel establishment, *(ii)* transition and *(iii)* disputes[1].

a) Channel Establishment: All parties cooperatively *open* a channel by locking collateral on the blockchain (cf. Figure 2). The funds can only be released by unanimous agreement or through a pre-defined refund condition.

[1]While earliest channel protocols differ slightly from the above three-part *state replacement* technique, they nonetheless fit within the framework of unanimous consent coupled with the local verification of state transitions.

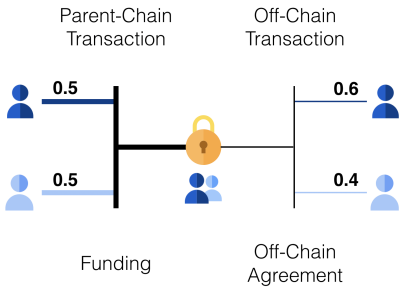

Fig. 2. Payment channel funding (UTXO model) and off-chain transaction.

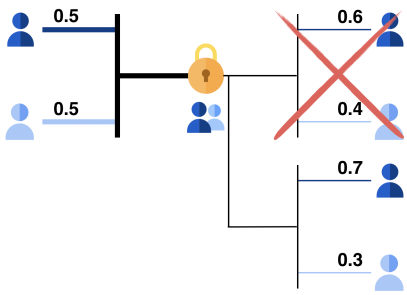

Fig. 3. Payment channel update (UTXO model), invalidate outdated state.

b) Channel Transitions: Once the channel is open, all parties can *update* the channel's state in a two-step process. First, one party proposes a new state transition by sending a signed command and the new $state_i$ to all other parties. Each party computes the state transition as $state_i \leftarrow T_\alpha(state_{i-1}, cmd_\alpha)$, where T_α denotes the transition function for application α and cmd_α denotes a given command relevant to application α. Second, all other parties re-compute the state transition to verify the proposed state before signing it and sending their signature to all other parties.

c) Channel Disputes/Closure: If an honest party does not receive n signatures before a local timeout, it assumes that there has been a disagreement about the proposed state. The honest party may trigger a layer-one *dispute* and enforce a new state transition without the cooperation of the other parties.

We generalize [30], [31] the properties and security guarantees for responsive parties offered by channels:

Unanimous Establishment: A channel is only considered open if all n parties agree to its establishment.

Unanimous Transition: A transition on layer-two, i.e. without an on-chain dispute, requires all n parties to agree.

Balance Security: An honest party can always withdraw the agreed balance from the channel with an on-chain dispute.

State Progression: A party can at anytime enforce an off-chain state transition on-chain, the state machine thus always reaches a terminal state.

TABLE I
OVERVIEW OF DIFFERENT CHANNEL DESIGN PROPOSALS.

	Channel technique	Throughput bottleneck	Dispute mechanism	Watchtower storage	Security proofs
RbI					
Spilman [24], [65]	Payment	Sender deposit	Closure	$O(1)$	×
Raiden [29]	Payment	Network	Closure	$O(1)$	×
RbI & Time Lock					
DMC [26]	Payment	Channel resets	Closure	$O(1)$	×
RbR					
Lightning [27]	Payment	Network	Closure	$O(N)$	×
RbV					
Sprites [30]	State	Network	Command	O(1)	✓
PISA Sprites [66]	State	Network	Command	$O(1)$	×
Perun [31]	State	Network	Closure	$O(1)$	✓
Counterfactual [67]	State	Network	Command	$O(1)$	×
Kitsune [68]	State	Network	Closure	$O(1)$	×

B. State Replacement Overview

Channel constructions are inherently based on state replacement techniques (cf. Figure 3). These techniques assume that participants in a channel are rational and follow the strategy with the highest payoff (e.g. a user publishes an older state if it represents a payment of higher value for this user). To be applicable for the wide range of protocols used to realize channels, the following section discusses generic state transitions. We distinguish four state replacement techniques:

- *Replace by Incentive (RbI).* A sender shares newly authorized states with a receiver. A rational receiver only signs and publishes the state that pays the highest amount.
- *Replace by Time Lock (RbT).* Every state is associated with a time lock[2], which decrements every time the state changes. The state with the lowest time lock is considered the latest state, as it can be accepted into the blockchain before all previously authorized states. Once a channel closes, the state that is included in the blockchain deprecates all other states.
- *Replace by Revocation (RbR).* All parties collectively authorize a new state before revoking the previous state. Upon dispute, the blockchain provides a time period for parties to prove that the published state is a revoked state.
- *Replace by Version (RbV).* States have a monotonic increasing counter representing the state version. Upon dispute, the authorized state with the highest state version is considered the latest state. A new state replaces a previous state if it has a larger version number.

For *RbI* and *RbT*, the latest state can only be written to the blockchain once. *RbR* and *RbV* introduce a dispute process where the counter-party can provide evidence that a state submitted to the blockchain is invalid. After the dispute, the off-chain contract can either be re-deployed to the blockchain (i.e. *closure dispute*, cf. Section III-D1) or a set of commands can be executed via the blockchain (i.e. *command dispute*, cf. Section III-D2). The introduction of a dispute process introduces a new assumption critical to the channel's security; the *always online assumption* [66] (cf. Section VI). Watching services mitigate the assumption by allowing users to delegate their responsibility of raising disputes to a third party.

[2]Time locks define either *absolute* time expressed as a blockchain block height, or *relative* time expressed as the number of blocks that must elapse after a transaction is included in the blockchain.

C. Payment Channels

We here present the evolution of payment channel designs.

1) Replace by Incentive: Spilman [65] presented the first major step towards secure (unidirectional) payment channels based on the RbI mechanism, implemented in Bitcoinj [24], [25]. This channel allows a sender to issue payments to a recipient, but the recipient cannot send the funds back through the same channel. To create a channel, the sender locks a deposit on-chain. The deposit can be refunded (i.e. the channel closed) if one of the following two conditions are met: *(i)* the sender retrieves their deposit after time t or *(ii)* both the sender and receiver authorize the release of the deposit. The channel state is represented as the balance of funds of both parties within the channel. To issue a payment, the sender signs a new state that monotonically decrements the sender's balance and monotonically increments the receiver's balance. The signature and the new state are sent to the receiver who can either *(i)* immediately sign and publish on-chain the new state to claim the payment, or *(ii)* wait for a new state from the sender that pays more coins. For the recipient, it is safe to wait for new states from the sender, because the on-chain deposit cannot be refunded to the sender until time t is reached[3]. The sender can continuously send new payments to the receiver until either the sender's balance is depleted or the receiver decides to close the channel before time t. When closing the channel, a rational receiver publishes the latest received state to settle with the highest amount of coins. To our knowledge, unidirectional payment channels are the only type of channel that allow the sender to remain safely offline, without the risk of losing funds. The throughput (number of transactions) of a unidirectional payment channel is limited by the size of the sender's deposit and the smallest denomination of the cryptocurrency asset. A deposit of e.g. 1 coin allows at most 10^8 transfers assuming a minimum denomination of 10^{-8}.

A *pair* of RbI channels can be combined to support bidirectional payments [29]. Unlike single RbI channels, the sender increments coins owed to the receiver and the value can go beyond the sender's deposit. When the channel is closed, the smart contract computes the offset of the coins owed in both RbI channels before sending each party their final balance.

2) Replace by Time Lock: In a UTXO-based blockchain, RbT allows the construction of bidirectional payment channels. Each state update is associated with a time lock, which prevents the transaction's acceptance into the blockchain, until some predefined time in the future. When the payment direction within the channel changes, the time lock associated to the new state update is decremented by an amount Δ, the *safety time gap* (cf. Figure 4). While RbT enables rapid micropayments, this mechanism suffers from notable limitations. The party receiving the final state update must be online at precisely time t to claim and publish the latest payment on the blockchain. If the latest state does not get accepted within Δ time (i.e. due to blockchain congestion), the counterparty has an opportunity to broadcast an older state update, attempting to reverse the final payment. The choice of Δ and the number of payment direction changes limit the channel's transaction throughput ceiling.

[3]Note that the blockchain acts as coarse time-stamping service.

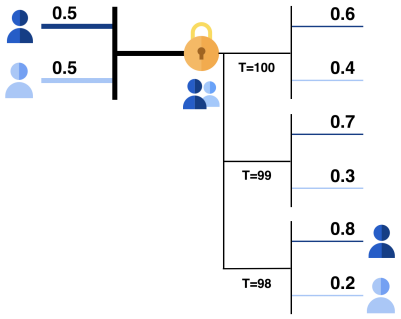

Fig. 4. Time lock-based payment channels in an UTXO model. Lowest time lock transactions are included on-chain first.

To alleviate these concerns, Decker and Wattenhofer [26] propose *Invalidation Trees* combining RbI and RbT, known as Duplex Micropayment Channels (DMC). Bidirectional payments (or duplex payments) are processed via a pair of RbI payment channels. When one channel exhausts its supply of coins, the channels can be *reset* by destroying the current state and re-creating a suitable state update for the pair of one-way payment channels in an off-chain manner via an invalidation tree. Each node in the tree has a time lock, and the branch with the lowest time lock is first accepted into the blockchain before the other branches. An alternative version of DMC [39] proposes to remove the channel's fixed expiry time and support n parties. There exists an inherent trade-off between the number of channel resets and the branch nodes required to broadcast in the event of a dispute. The worst-case dispute requires the entire branch to be published with $n + 2$ states, given n nodes in the invalidation tree and two *RbI* channels. Parties must be online during the safety time gap to ensure the latest branch is written to the blockchain.

3) Replace by Revocation: Poon and Dryja propose Lightning channels to overcome the previous state replacement channel throughput limitations and to remove expiry times [27], [69] (cf. Figure 5). We refer to Lightning channels as RbR, because both parties agree on the channel's new state before revoking the old state. To revoke, both parties exchange revocation secrets (i.e. a preimage of a hash) and retain those during the channel's lifetime. A penalty mechanism discourages parties from broadcasting older states. If one party broadcasts a revoked state, the blockchain accepts within a time-window proofs of maleficence from the other party. A successful dispute grants the winning party *all* coins of the channel. RbR is the first channel design to require both parties to remain online and fully synchronized with the blockchain to observe malicious closure attempts. Unfortunately, *RbR* introduces unfavorable implications for third-party watching services (cf. Section VI). With N being the number of channel updates, *RbR* entails $O(N)$ storage as the watching service must store evidence for every in-channel update to prove an authorized state as revoked.

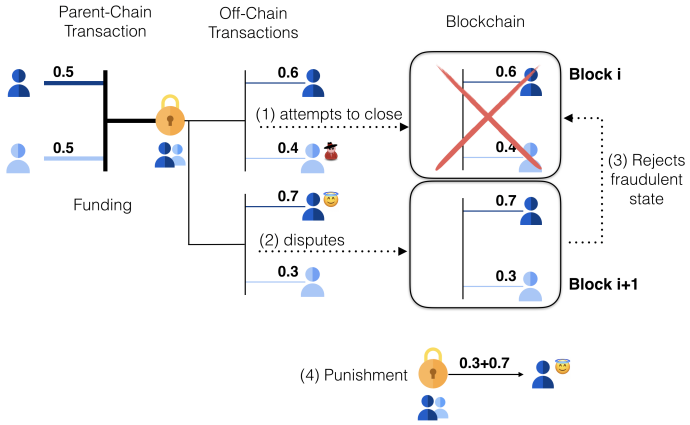

Fig. 5. Payment channel construction based on a punishment scheme [27]. If a malicious channel party (1) attempts to close the channel with an outdated state, the other channel party can (2) dispute the closure, and (3) reject the fraudulent closure. The honest party (4) receives the total channel balance.

4) Replace by Version (UTXO-blockchains): Decker *et al.* propose Eltoo [70] to support *RbV* for UTXO-based blockchains through the use of *floating transactions*, i.e. transactions attachable to an output of any preceding transaction. With the possibility of linking established updates, the Eltoo technique utilizes *state numbers* to impose time ordering on the updates and supports the storing of temporary state. As with Lightning channels, there is no expiry time and no limitation on the channel's throughput. Its closure dispute process is similar to that of state channels (cf. next Section) and there is no penalty for publishing replaced states. Watching services can verify a newly received state with $O(1)$ storage costs, requiring only the state with the largest state number.

D. State Channels

State channels extend the payment channel concept towards the execution of *arbitrary* applications and typically involve two smart contracts: one for the state channel itself and one for the to be executed application. Known state channels rely upon *RbV* as the state replacement technique and thus entail $O(1)$ storage requirements for watching services. To the best of our knowledge, only Sprites [30] and Perun [31] offer formal security proofs. We distinguish state channels among *closure disputes* and *command disputes*.

1) Closure Disputes: In a *closure dispute*, one party triggers a dispute to close the channel and to continue the application's execution exclusively on layer-one. Perun proposes two-party state channels that support installing/uninstalling application smart contracts off-chain [31]. Its dispute process focuses on a single application and enforces a fixed time window for involved parties to submit collectively authorized states for the application with the largest version (i.e. *RbV*). After the time window, any party can resolve the dispute. This re-deploys and continues the application's smart contract with its current state on the blockchain [31]. To install an application, both parties collectively sign the application's new state, the number of coins allocated to it, and its initial version. To uninstall the application, both parties authorize a state that terminates it and de-allocates the coins. The coins are only unlocked based on the outcome of the conditional application smart contract. Kitsune [68] relies on the same closure dispute process but is designed to support n parties and relies on an existing smart contract on the blockchain.

2) Command Disputes: A command dispute aims to execute a specific command on the parent-chain, and then resume execution off-chain. The channel thus does not close and can continue its off-chain execution after the command executes [30]. The blockchain grants a pre-determined time window to collect commands from each party and all commands execute after the dispute process expires. After the dispute, the state version increments, and the new state transition is considered the newest channel state. Note that a state with a newer *RbV* version can cancel the dispute process.

PISA [66] reduces Sprite's [30] dispute costs by allowing parties to submit a hash of the state. Arbitrum [71] removes the overhead for an honest party to send the full state to the blockchain, instead the honest party can assert the hash of a new state alongside the command and its input. Counterfactual's [67] (and others [72]) command dispute process allows the execution of multiple commands by extending the dispute process expiry time depending on the number of commands [67]. While Counterfactual allows parties to install and uninstall multiple applications off the chain, it is restricted to two parties and turn-based applications.

E. Channel Hierarchy

Aiming to reduce the number of required on-chain transactions, the approaches introduced here increase the flexibility of channels with regard to applications and participants.

a) Multiple Applications: Dziembowski *et al.* [73] and Counterfactual [67] explore the possibility of installing and uninstalling applications off-chain (i.e. without on-chain fee). This allows parties to execute multiple concurrent applications (e.g. tic-tac-toe, poker and bi-directional payments). Such modular channels maintain a set of application instances and each instance operates on an individually allocated collateral. Application instances are isolated from each other (even in case of disputes) and based on *RbV*. Collateral is unanimously assigned to one application and cannot be used simultaneously for other applications for security reasons.

b) Channel Factories: Burchert *et al.* [39] propose the concept of a channel factory for Bitcoin, whereby n parties lock coins into a n-party deposit that is then re-allocated to a set of pair-wise payment channels. Each party may maintain one or more pair-wise channels to facilitate transactions. Whenever two parties want to establish a direct channel, all parties cooperatively agree to create a new re-allocation of pair-wise payment channels by jointly updating the n-party deposit. This re-allocation of pair-wise channels can be built using DMC [39], while Ranchal-Pedrosa *et al.* [74] replace DMC with Lightning channels.

F. Channel Synchronization

The channel designs discussed in the previous section are limited to the direct interaction among connected parties. This

brings forth a new question of whether it is possible for two (or more parties) to avoid setting up a new direct channel on the parent blockchain (and thus avoid prohibitive fees) if there is a path of channels that connect them on the network. To facilitate synchronizing a payment (or executing a smart contract) across a path of channels, we present *conditional transfers*. Those allow the sender to lock coins into a transfer such that the receiver can only claim the funds if a condition is satisfied before an expiry time. As we cover in the following, channel synchronization requires every hop along the path to set up conditional transfers with their counterparty. It is crucial that all synchronization techniques guarantee *no counterparty risk* and atomicity of the transaction, such that no honest parties along the path are at risk of coin loss (or theft).

a) Hashed Time-Locked Contracts (HTLC): HTLC [26], [75] enable cross-channel synchronization by allowing A to lock x coins from A and B's channel that are only redeemable if the contract's conditions are fulfilled. The conditions of the contract *HTLC(A, B, y, x, t)* rely on a collision-resistant hash function H, a hash value $y = H(S)$, where S is chosen uniformly at random, the amount of coins x and a timeout t. If B produces a value S such that $H(S) = y$ before timeout t expires, B gets the locked x coins. Otherwise, if timeout t expires, the locked x coins go back to A. Let's assume that A wants to send a payment to C using B as intermediary and there exist a channel between A, B and B, C. The receiver C generates a secret S and provides the initial sender, A, with the hash of the secret $H(S)$. A can then establish a HTLC with B, the next hop, which B may spend under the condition that he can provide A with the preimage of $H(S)$ before a set expiry date, generally expressed in number of blocks. Should B fail to provide the requested input in time, A is able to commit a refund transaction on-chain. B then constructs a similar HTLC with C with two main differences: *(i)* the lock period of the HTLC output is reduced, thereby ensuring that B has enough time to claim the funds from the previous hop; and *(ii)* the amount of coins locked in this HTLC is reduced. The difference in locked coins corresponds to the service fee charged by B for the forwarding of the payment. As C is the final receiver, C can safely reveal S to B, who then has a sufficient period of time. HTLCs can be used for paths of an arbitrary number of channels and are integrated in Lightning [27] and DMCs [26].

A key concern for HTLCs is whether sufficient collateral is available when setting up a path. Every hop along the path includes *an additional* time-delay to ensure the hop can always retrieve their coins from the previous hop after sending their coins to the next hop. The longer the payment path, the more collateral must be reserved. In the worst-case, the collateral cost is $\theta(l^2 X \Delta)$, where l is the number of channels, X is the payment amount and Δ is the time that an on-chain transaction takes longer than an off-chain exchange.

b) Global Preimage Manager: Miller et al. [30] introduce a *PreimageManager* smart contract that allows a single dispute to atomically finalize the synchronized transfer for all hops along the path. This reduces the collateral lock up time for synchronizing a payment across l channels to $\theta(lX\Delta)$. The smart contract accepts the preimage of a hash and stores it alongside a timestamp. All parties along the path introduce a new conditional transfer that asserts the transfer is only considered complete if *the preimage x of hash $h = H(x)$ was published in the PreimageManager before time t*. A single dispute at any position along the path can use the published value x before time t to reach an agreement about the new state. Interestingly, all channels along the payment route can re-use the same condition for their transfer without waiting for each other. Sprites converts disputes as local events in each channel to a single global event, guaranteeing that all channels share the same worst-case expiry time.

c) Scriptless Multi-Hop Locks: HTLC-based synchronization protocols are limited to connect channel constructions with the same hash function (e.g. SHA256). HTLCs also suffer from the severe wormhole attack, which prevents users from successfully executing the synchronization protocol and allows an adversary to steal the synchronization reward [76].

A Multi-Hop Lock (MHL) [76] is an alternative synchronization mechanism that enables cross-channel synchronization. Like a HTLC, a MHL allows A to lock x coins in A and B's channel that can only be released if a set of conditions are fulfilled. The crux of MHL is that the cryptographic hardness condition is no longer encoded in the underlying blockchain's scripting language (and thus it is called *scriptless* in the blockchain folklore). The scriptless locks stem from Poelstra [77] who provided a way to embed certain contracts into Schnorr signatures. Malavolta et al. [76] formalized this construction, proposed alternative constructions relying on ECDSA signatures and one-way homomorphic functions and enabled the combination of locks with different signatures in one payment path. This approach enables interoperable [78] locks across all blockchains that support a digital signature scheme to authorize transactions. Moreover, this approach provides provable security, privacy guarantees and solves the wormhole attack (cf. Section VI). The conditions of *MHL(A, B, x, m, pk, t)* depend on a message m, a public key *pk* of a given signature scheme and a timeout t. If B produces a valid signature σ of m under *pk* before timeout t expires, B receives the locked x coins. Otherwise, if timeout t expires, the locked x coins are returned to A.

d) Virtual channels: Path intermediaries in a channel synchronization are required to remain online and explicitly confirm all mediated transactions. Dziembowski et al. [31], [73] address these shortcomings with the introduction of virtual channels that support payment and state transitions. All intermediaries along the route can lock coins for a fixed period of time and both parties can treat the path as a new *virtual channel* connecting them directly. In this manner, A and B can transact without interacting with intermediaries along the path, thus reducing the transaction latency. Virtual channels are limited by the need to recursively set up a new virtual channel for every intermediary along the path. It is the intermediary's responsibility to ensure the channels close appropriately.

e) Trusted Execution Environments: An alternative layer-two technique is to leverage trusted execution environments (TEE), e.g. Intel SGX [79]. Teechain [47] and Teechan [48] synchronize payments across channels using TEEs. TEE enable expressive and off-chain smart contracts on restricted Bitcoin-based blockchain [49]. Tesseract [80] proposes to construct a scalable TEE based cross-chain cryptocurrency exchange. Trusting a TEE to provide integrity naturally overcomes many obstacles of non-TEE protocols:

No collateral lockup: TEEs absorb the trust requirements, otherwise guaranteed via on-chain collateral.

Interoperability: The computation at the TEE can encode the logic and transaction format required for any blockchain.

Parallelized Disputes: TEEs can emulate the logic of global preimage manager to enable parallel disputes.

No wormhole attacks: TEEs follow the protocol definitions and pay honest users for their synchronization service.

Note that besides the shifted trust assumptions towards the CPU manufacturer, TEEs suffer from their own security concerns such as rollback [51] and side-channel attacks [81].

1) Observations: We outlined several methods for conditional transfers to synchronize transactions across a path of channels with different tradeoffs between complexity and trust assumptions. While TEEs can reduce protocol complexity, they introduce new trust assumptions and attack vectors.

G. Routing

If A wants to pay B using a set of intermediate channels, it is necessary to first find one or several paths of open channels from A to B. If the payment only utilizes a single path, all channels need to have sufficient collateral to conduct the payment. If the payment is split over multiple paths, it is necessary to divide the payment in such a manner that channels on each path can handle the partial payment. In this section, we introduce *routing algorithms*, i.e. algorithms for finding paths in a network of payment or state channels. For simplicity, we use the example of payment channels throughout the section. The protocols, however, do apply to state channels.

Existing network routing algorithms for data transmission experience unique challenges when applied to PCNs. The goal of data routing algorithms is the transfer of data from one node to another, i.e. routing changes the state of nodes by transferring information. Node links and bandwidth capacities in data networks moreover are not considered private information. Retransmission of data is an inherent feature of e.g. TCP, and typically doesn't induce significant economic losses to either sender or receiver.

In contrast, the goal of a payment channel routing algorithm is to change the state of the traversed channels to secure the asset delivery from sender to receiver. Depending on the transaction amount, certain channels may not be suitable to route a payment, and channel balances are thus an obstacle that routing algorithms have to account for. An executed channel transaction permanently alters the state of all channels along the path. Further parameters, such as bandwidth and network latencies moreover influence a channel path's delay characteristics. To protect user privacy, only the total capacity of a channel is disclosed, not the distribution of funds among the two channel participants. Channel transactions might therefore fail and the routing algorithms attempt different execution paths until one succeeds. PCN routing algorithms, therefore, have to account for the unique characteristic of channels to provide satisfactory path recommendations[4]. We summarize five crucial properties routing algorithms for payment channels should satisfy [32]–[34].

Effectiveness: Given a PCN snapshot and the channel balances, the algorithm should find paths which maximize the success probability of a payment. The algorithm should remain effective when channel balances change.

Efficiency: The overhead of path discovery should be low in latency, communication and computation. Changes of the PCN topology should entail a low update overhead cost.

Scalability: The routing algorithm should remain effective and efficient for large-scale PCNs and high transaction rates.

Cost-Effectiveness: The algorithm should find paths with low transaction fees. The fees of a layer-two transaction should be inferior to a layer-one transaction.

Privacy: Routing paths should be found without disclosing transaction values (i.e. value privacy) and the involved parties (i.e. sender and receiver privacy).

We distinguish between two classes of routing algorithms: global routing and local routing. In global, or source routing, each node maintains a local snapshot of the complete PCN topology. In local routing, the algorithm operates on local information, i.e. is only aware of the node neighbors with which it established channels with. We summarize the performance of algorithms presented in related work in Table II.

1) Routing Algorithm Details: In the following, we describe each algorithm in detail and focus on the aspects of effectiveness, efficiency, and scalability. As outlined in Table II, only one algorithm class explicitly considers cost-effectiveness. While others like SpeedyMurmurs [33] implicitly achieve low fees by selecting short paths if fees are homogeneous, the algorithm design and evaluation do not include this aspect. Similarly, only SilentWhispers [32] and SpeedyMurmurs [33] introduce concrete notions of privacy (cf. Section V). Some algorithms involve the use of onion routing [29], [34], [36], which requires the random selection of nodes in a path to achieve its anonymity guarantees [83]. As routing algorithms do not select nodes randomly, it remains unclear if onion routing provides privacy in the context of payment channels (cf. Section V for our privacy observations).

a) Global View: Lightning [27] and Raiden [29] use *Source Routing* [35], in which the source of a payment specifies the complete route for the payment. If the global view of all nodes is accurate, source routing is highly effective because it finds all paths between pairs of nodes. However, by

[4]Note that Tor-like routing is inappropriate, as Tor assumes a random relay selection, which wouldn't account for channel capacities.

TABLE II
ROUTING ALGORITHMS FOR MULTI-HOP PAYMENTS.

		Global View		Local View			
		Lightning [27]/Raiden [29]	SpiderNetwork [34]	Flare [37]	cRoute [82]	SilentWhispers [32]	SpeedyMurmurs [33]
Effectiveness	Snapshot	High	High	High	High	Medium	High
	Dynamic	Medium	High	NA[1]	High	Low	Low
Efficiency	Latency	Low	Low/High[2]	High	High	High	Low
	Communication	Low	Low/High[2]	High	High	High	Low
	Computation	High	High	Low	Low	High	Low
	Update	High	High	NA[1]	Low	High	Low
Scalability	Nodes	Low	Low	NA[1]	?[3]	High	High
	Transactions	High	High	NA[1]	High	Low	Low
Cost-Effectiveness	Considered	✓	×	×	×	×	×
Privacy	Guarantees	×	×	×	×	✓	✓

[1] Dynamic behavior not specified. [2] High for on-chain re-balancing, otherwise low. [3] No evaluation, only tested for 77 nodes.

default source routing does not consider channel balances, and routing decisions might contain channels with low balances or implicitly turn bidirectional channels into unidirectional ones, reducing the available routes over time in a dynamic PCN.

SpiderNetwork [34] improves the effectiveness of source routing in a dynamic PCN by introducing three key modifications: i) the choice of the routes includes a bias towards routes that optimize the balance, ii) routing includes on-chain rebalancing, meaning that nodes deposit additional coins to improve the balance, and iii) routing relies on a packet-switched network, i.e. instead of routing a complete payment, the algorithm splits the payment into constant-size units and routes each of them individually. SpiderNetwork is therefore highly effective even when balances are constantly changing, at the cost of higher latencies if on-chain rebalancing is used.

By pre-computing paths locally, algorithms based on a global view exhibit low latencies and communication overheads. However, the local memory costs to store the ever-complete snapshot and computation costs for finding paths (and solving an optimization problem cf. SpiderNetwork) are high. The same holds for the overhead resulting from opening or closing a channel on-chain, and these overheads increase super-linearly in the number of nodes and limit scalability.

b) Local View: Algorithms based on local information use well studied concepts: *(i)* distributed hash tables (DHTs) [84], *(ii)* flow algorithms [85], *(iii)* landmark routing [86] and *(iv)* network embeddings [87].

Flare [37] leverages the Kademlia DHT [84]. Kademlia in its original form opens new channels between strategically chosen nodes, which is expensive in terms of latency and on-chain fees. Flare, therefore, uses a modified Kademlia version that replaces direct channels with multi-hop paths, which does not require opening new channels. This modification results in longer routes, higher latencies and communication overheads compared to traditional DHTs. The likely most significant limitation of Flare is its inability to support topology changes.

Celer [82]'s routing relies on a flow algorithm, *cRoute*. An optimization problem is formulated based on local congestion with solutions guided by the congestion gradients. The algorithm is effective and keeps channels balanced. As the evaluation of cRoute only considers 77 nodes [82], we cannot make conclusive statements about its scalability, related work indicates problems with efficiency and scalability [33], [85].

SilentWhispers [32] implements landmark routing, where landmarks are dedicated nodes. Each node keeps track of the neighbor to contact to reach all landmarks. A payment between two nodes first traverses the path from the sender to the landmark and then from the landmark to the receiver. Using multiple landmarks in combination with multi-party computation enables payments to be split over multiple paths in a privacy-preserving manner. Each node periodically recomputes how to reach the landmarks to account for topology changes. However, as recomputation is not necessary for every topology change, the costs of updates are lower than source routing. The evaluation of SilentWhispers on a real-world dataset reveals low effectiveness and moderate latencies in comparison to other algorithms [33]. Multi-party computation, required for each transaction, involves computation costs and results in low scalability when increasing the number of transactions.

Aiming to overcome the drawbacks of SilentWhispers, SpeedyMurmurs [33] uses embedding-based routing and a protocol for handling topology updates locally. Nodes express their position in a rooted spanning tree through coordinates and locally choose the next node in a payment path by considering all adjacent channels with sufficient balance. Among these channels, nodes then select the channel to the node with the coordinate closest to the recipient's coordinate. While the coordinate assignment results from the underlying spanning tree, the path can contain channels that are not part of the spanning tree. In this manner, SpeedyMurmurs exhibits high effectiveness and low latencies for a static PCN. As SpeedyMurmurs blocks funds while conducting payments, high transaction frequencies might be affected due to locked funds. If a channel opens or closes, only descendants in the underlying spanning tree have to adjust their coordinates, which typically results in an overhead that is logarithmic in the network size [88]. However, SpeedyMurmurs does not consider balances, which results in low effectiveness due to insufficient balances in a dynamic PCN [34], [82].

c) Deadlocks: Deadlocks may arise on concurrent payments that proactively block deposits on channels [89]. For instance, assume that A and B conduct concurrent payments

and both choose paths that contain the channels c_1 and c_2. Furthermore, c_1 and c_2 have sufficient collateral to complete either but not both payments. Now, if A's payment blocks funds of c_1 before B's payment does and B's blocks funds of c_2 first, both payments fail. In the context of source routing, a suggested solution [89] is to design a global partial order $<$ on the set of channels, while it is unclear how to adapt the proposal to routing protocols relying on a local view.

d) Multi-path Routing: Networks typically spawn multiple paths from a node A to a node B that routing may find. Existing algorithms fall under the following categories: *(i)* single-path routing algorithm [27], *(ii)* multi-path routing that explicitly split the payment over several paths [32], [33]) and *(iii)* packet-switched routing that routes each unit of payment individually [34]. A partial evaluation indicates that packet-switched networks provide the best performance with regard to effectiveness [90]. The evaluation is limited to source routing and does not evaluate packet-switching for alternative routing algorithms. These results indicate that packet-switched routing algorithms is a promising direction of future research.

To our knowledge, no routing algorithm fulfills all desired criteria. Algorithms requiring a global view have inherent scalability issues. Algorithms based on a local view are scalable but are bound to provide lower effectiveness and efficiency. While the existing algorithms exhibit low performance or lack in-depth evaluations, they represent the first application of key routing concepts to payment and state channel. In particular, there is no inherent reason why coordinate-based routing algorithms cannot achieve high effectiveness in dynamic PCN settings. Future research accounting for channel balances may have the potential to overcome such issues. Rebalancing algorithms that transfer funds along circular paths are also an approach worth further investigation [38].

H. Watching Services

To alleviate the online assumption of PCN users, related work proposes to outsource the responsibility of issuing challenges to third-party watching services [66], [91], [92]. Users outsource their latest state to the watching service before parting offline. Watching services then act on behalf of the users to secure their funds. Users can still verify the correct behavior of watching services and punish them (e.g. by keeping pre-allocated collateral) in case of non-compliance. Monitor [91] provides watching services within the Lightning Network. WatchTower [92] is designed for Eltoo and requires $O(1)$ storage but is currently not compatible with Bitcoin's consensus rules. PISA [66] provides watching services for state channels, requires $O(1)$ storage and is the only proposal suitable for a *financially accountable* third party. PISA instances provide receipts to offline users, who can burn their security deposit if they misbehave. DCWC [93] enables users to engage multiple watching services, increasing the probability of at least one honest watcher protecting the offline user's interests.

I. Payment Channel Hubs

Related work [94] observes that layer-two systems benefit from centralized (but non-custodial) star-topologies to reduce *(i)* collateral lockup costs and to *(ii)* simplify routing complexities. A payment channel hub (PCH) is essentially a node in a PCN that maintains many channels with different peers. Having a network with multiple interconnected PCHs should result in a lower average path length. A reduced path length implies a reduction in collateral cost and route discovery complexity. Still PCHs face significant locked capital requirements for each channel. For example, a PCN node with $1M$ channels, each channel sending on average $\$1000$ of transaction volume, requires the hub to lock up a total of $\$1B$. Rebalancing operations are only possible via costly and slow parent-chain transactions. Moreover, user-onboarding is a costly process, a PCN node with $1M$ users would require $1M$ parent-chain setup transactions (costing more than $\$100k$ on Ethereum).

J. Observations

Myth 1 insights: The mentioned results suggest that blockchains can scale further by leveraging layer-two technologies and thus without changes to the underlying layer one. However, PCNs experience limitations and their scalability properties have not yet been quantified appropriately.

All layer-two protocols should consider scalability in terms of the number of nodes, the total number of channels of the network, the distribution of the number of channels per node, the cumulative number of transactions in the network and the distribution of values of those transactions.

Myth 4 insights: While layer-one transaction costs are quantified by their *size* (on UTXO blockchains), or computational complexity (on smart contract blockchains), the transaction costs on layer-two are primarily correlated to the transaction value (in \$). The higher a layer-two transaction value, the more on-chain collateral needs to be reserved, locking up potentially considerable amounts of funds *in advance*.

IV. Commit-chains

In contrast to channels, commit-chains are maintained by one single party that acts as an intermediary for transactions. Hence, commit-chains serve a similar purpose as payment channel hubs but with protocols specifically optimized for this scenario. In the following, we provide an overview of commit-chains before describing two pioneer commit-chains proposals: NOCUST [28], an account-based commit-chain, and Plasma [40], a UTXO-based commit-chain. In Table III, we provide an overview of the properties of NOCUST and Plasma Cash [95] (a simplified Plasma variant).

A. Overview

Similar to PCHs, a commit-chain relies on a centralized but untrusted intermediary that facilitates the communication among transacting parties. The operator is responsible for collecting commit-chain transactions from the users and periodically submits a commitment to all collected transactions

to the parent-blockchain. Unlike channels, commit-chains do not rely on a three-state model (opening, live, dispute/closure phase), but rather on an *always ongoing state* once launched. After an operator has launched a commit-chain, users can join by contacting the operator. They can then conduct transactions that are recorded on the commit-chain. Users can anytime withdraw or exit their assets to the parent chain.

a) Periodic Checkpoint Commitments: Commit-chain users may need to periodically return online to observe the on-chain checkpoint commitment, which can be instantiated as a Merkle tree root or a Zero Knowledge Proof (ZKP) [28], [96]. Merkle root commitments do not self-enforce valid state transitions and therefore require users to participate in challenge-response protocols if a commitment is invalid. In contrast, ZKPs enforce consistent state transitions on-chain, thus reducing potential operator misbehaviour. Currently, there exists no efficient method to instantiate commit-chains on blockchains without highly expressive scripting languages.

b) Data Availability: As commit-chain data is not broadcasted for efficiency reasons, users must retrieve/maintain data required to (partially) exit a commit-chain, commonly referred to as the data availability requirement. Data availability challenges may challenge a commit-chain operator to provide the necessary data or halt the operator upon misbehavior [28], allowing users to exit with their last confirmed balance.

c) Centralized but Untrusted Intermediary: The centralized operator may become a point of availability failure, but not of custody of funds. The operator may thus censor commit-chain transactions, encouraging mistreated users to exit anytime and move towards another commit-chain.

d) Eventual Transaction Finality: Unlike previously discussed layer-two protocols, the intermediary commit-chain operator does not require on-chain collateral to securely route a payment between two commit-chain users. In this setting, commit-chain transactions do not offer instant transaction finality (as in channels), but eventual finality after commit-chain transactions are recorded securely in an on-chain checkpoint.

e) Reduced Routing Requirements: Because a commit-chain can potentially host millions of users, few statically connected commit-chains are envisioned to spawn stable networks with low routing complexity. However, we are not aware of any proposals for atomic cross commit-chain transactions.

We generalize the security properties for users as follows:

Free Establishment: Users join a commit-chain without on-chain transaction by requesting an operator signature [28].

Agreed Transition: A commit-chain transaction is agreed by at least the sender and the commit-chain operator.

Balance Security: Honest users can always withdraw agreed balances from the commit-chain with an on-chain dispute.

State Progression: User can anytime enforce an off-chain state transition on-chain.

Commitment Integrity: Users may verify the integrity for commitments and force the commit-chain operator to seize operation (and rollback to the latest commitment).

Unlike channels, *state progression* is not a default security property for commit-chains, because they only offer *eventual finality*, unless off-chain transactions are secured by additional collateral [28]. In the worst case, transactions remain unconfirmed if the next commitment is invalid or not provided.

B. NOCUST

NOCUST [28] is an account-based commit-chain where an on-chain address is associated to a commit-chain account. The NOCUST on-chain contract expects to periodically receive a constant-sized commitment to the state of the commit-chain ledger from the operator, containing each user's account in the collateral pool. The commitment to this (potentially) large state is constructed such that it is efficient to prove and verify in the smart contract that a user's commit-chain account was updated correctly by the operator, such that transfers, withdrawals and deposits can be securely enacted. Users can deposit any amount of coins within the contract, and perform commit-chain payments of any denomination towards other users. TEX extends NOCUST to support atomic commit-chain swaps [97].

NOCUST proposes *free establishment*, wherein a user can join the commit-chain without on-chain transaction and immediately receive commit-chain transactions. Regarding *agreed transition*, a transaction within NOCUST is enacted with the signature of the sender and the operator to deter potential double-spend scenarios. NOCUST provides *balance security* towards honest users, even if the operator and all other commit-chain users collude. A transaction is considered final, when the sender and operator agree to the payment, and the payment is committed within the periodic on-chain checkpoint. NOCUST only offers *state progression*, if the operator stakes collateral towards the recipient. To this end, NOCUST specifies a mechanism to allocate collateral towards all commit-chain users within a constant-sized on-chain commitment, enabling instant transaction finality for the specified amounts. The allocated collateral is *re-usable* after each checkpoint. The transaction throughput is only limited by the operator's

TABLE III
COMMIT-CHAIN PROPERTIES AND OPERATIONAL COSTS. PLASMA DATA FROM DISCUSSIONS WITH KONSTANTOPOULOS [95].

General properties	Plasma Cash [95]	NOCUST [28]
Security proofs	×	✓
Offline transaction reception	✓	✓
Fungible payments	×	✓
Clients can remain offline	×	×(online each eon)
Safe mass exit	✓	✓
Instant transaction finality	×	✓(with collateral)
Token support	✓	✓
Non-Fungible tokens	✓	×
Provably Consistent State (ZKP)	×	✓
Commit-Chain Swaps	×	✓ [97]
Costs		
Parent-chain commit	Low	Low
Deposit (parent → commit-chain)	Low	Low
Withdraw (commit → parent-chain)	Low	Low
Dispute initiation	Low	Low
Dispute answer	Low	Low
User storage	High	Low
User verification	High	Low
User bandwidth	Low	Low

bandwidth and computational throughput, and independent of the checkpoint commitment interval. While NOCUST users are not required to be constantly online, they are expected to monitor the blockchain at regular time intervals to observe the checkpoint commitments for *commitment integrity*. Each user is only required to verify their respective balance proof by requesting data directly from the operator and comparing it to the locally stored state. In the case of any misbehavior, a user can always issue a challenge using the NOCUST smart contract to force the operator to promptly answer this challenge with valid information. If the operator responds to the challenge with invalid information (or does not respond), users have an accountable proof of misbehavior. To strengthen the operator's integrity, NOCUST supports a provably consistent mode of operation through the use of zkSNARKS. As such, the underlying smart contract validates layer-two state transitions and the operator is not able to commit to invalid state transitions, without being halted by the smart contract.

C. Plasma

Plasma [40] is a high-level specification of a UTXO-based commit-chain. Following the initial proposal, a variety of alternatives are informally discussed [98]–[102]. We only discuss Plasma Cash [95] as it is the most comprehensive working draft. In this system, all coins are represented as serial numbers and every transfer allocates a new owner for the respective coin. A coin is minted with an on-chain deposit and cannot be merged or splitted with another coin on the commit-chain. This limitation reduces the practical applicability as a payment system (but is helpful for non-fungible tokens) and several coins may be required to facilitate a single transfer. Plasma Cash therefore resembles classical e-cash protocols [103], [104] with fixed coin denomination.

In terms of *agreed transition*, a transfer is incomplete until the recipient has verified the *entire* coin transaction history (which needs to be transmitted off-chain), the transaction is included in a hash commitment in the parent-chain and the hash commitment's pre-image is shared with the user. Plasma Cash doesn't specify a mechanism for *free establishment*, but should support this feature similar to NOCUST. While there is no mechanism to challenge the integrity of a hash commitment by the operator to achieve *commitment integrity*, all users can detect invalid commitments and it is expected that they eventually withdraw their coins from the commit-chain. If an operator commits to an invalid coin transfer and tries to withdraw it, the coin's owner can also issue a withdrawal for the same serial number based on a previous hash commitment. If the operator cannot prove to the parent-chain that the coin was spent, then their invalid transfer withdrawal is cancelled, and the rightful owner receives the coins. In addition, if a party tries to withdraw an already spent coin, then the coin's current owner can prove to the blockchain that it was already spent. Thus, it appears that Plasma Cash achieves *balance security* as an honest party can always withdraw their coins from the commit-chain, even if an invalid commitment is posted. In addition, the owner must keep the entire transaction history for each coin and confirm all transactions are confirmed in the commit-chain. Finally, each commitment can only include a single transfer per coin as the operator is not trusted to prevent double-spends and thus the transaction throughput relies on the on-chain commitment frequency. Plasma Cash does not specify a method to provide *state progression*.

D. Observations

Contrary to channels, commit-chains allow transaction recipients to remain offline at the time of payment, approaching similar usability properties as layer-one transactions. Their properties allow for a reduction in the required layer-two collateral, however, require smart contract enabled blockchains.

Myth 1 insights: Commit-chains have shown to be able to scale PoW blockchains by several orders of magnitude [28]. They clearly trade decentralization for a more centralized (but non-custodial) architecture.

Myth 2 insights: Due to periodic checkpoints in commit-chains, delayed transaction finality is secure without collateral of the intermediate operator [28]. Operator collateral is "re-usable" [28] after each checkpoint, potentially reducing the locked capital and on-chain costs of PCHs.

V. Anonymity and Privacy

Layer-one transaction anonymity and privacy is extensively studied [105]–[108], uncovering that blockchain pseudonymity does not entail strong privacy guarantees. A public blockchain allows an adversary to link sender and receiver of payments as well as trace back the origin of coins, breaking the *unlinkability* and *untraceability* properties. Privacy-focused blockchains [109], [110] build upon cryptographic techniques [111]–[113] to obfuscate on-chain information. Unfortunately, side-channel information (e.g. usage patterns) enable linkability and traceability attacks [109], [114]–[116]. As off-chain transactions only have a minimal blockchain footprint, one might believe they provide privacy-by-design.

Achieving privacy and unlinkability of layer-two transactions is not trivial [43], [117]. The creation of a channel associates a permanent pseudonym (e.g. public keys), while synchronization among channels (cf. Section III-G) may reveal the pseudonym of the cooperating parties. Furthermore, naive route discovery among two channels with a disjoint set of participants might require the knowledge of the (partial) topology for the channel network. In HTLC payments (cf. Section III-F), the intermediate channels on the path use the same cryptographic condition $y = H(R)$. An adversary on the path can observe the channel updates (i.e. share the same condition y) and can deduce who is paying to whom.

A. Layer-Two Privacy Notions

We differentiate between *(i)* an *off-path* adversary, which only has access to the blockchain; and *(ii)* an *on-path* adversary, which additionally participates in the layer-two protocol.

1) Payment Hub Privacy: A PCH (cf. Section III-I) or commit-chain (cf. Section IV) operator may have access to mediated transaction amounts and sender/receiver pseudonyms. In such a setting, we consider the following privacy notions.

Payment Anonymity [43]: In the absence of side channels, the receiver of a payment, possibly in collusion with a set of malicious senders, learns nothing about an honest sender's spending pattern.

Unlinkability [41]: The operator cannot link the sender and the receiver of a given payment among the set of all feasible sender-receiver pairs.

2) Multi-Hop Privacy: We consider the following privacy properties for routed payments (cf. Section III-G).

(Off-path) Value Privacy [117]: An adversary not involved in a payment does not learn the transacted value.

(On-path) Relationship Anonymity [117]: Given two payments between two pairs of honest sender and receiver, the adversary (who might control some of the intermediate channels) cannot tell which sender paid to which receiver with a probability higher than 50%.

Unlike other payment networks such as credit networks [32], [118], existing privacy notions in PCNs do not consider link privacy (e.g. whether an adversary can determine the existence of a payment channel between two users) or whether it is possible to infer the (partial) topology of a PCN.

B. Privacy Enhancing Protocols

While related work covers layer-one transaction privacy extensively [110], [119]–[126], it is as yet unclear if such techniques are applicable to layer-two protocols. Instead, the literature proposes layer-two tailored privacy proposals. We distinguish among *(i)* hub-based and *(ii)* multi-hop payment protocols.

1) Privacy Enhancing Payment Channel Hubs:

a) TumbleBit: TumbleBit [41] is a unidirectional PCH relying on an untrusted intermediary, a *Tumbler*. The Tumbler issues anonymous vouchers that users can exchange for coins. A key aim of TumbleBit is to prevent an adversary from linking a payment from a particular payer to a particular payee (an unlinkable payment hub). However, the collusion between the Tumbler and the payee in combination with timing analysis can considerably reduce the number of potential payers. Similarly, side channels (e.g. a unique product price) allows to narrow the set of feasible sender-privacy pairs. Particular threats that are not addressed by the current design are: *(i) intersection attacks* [127] correlate information across different time periods, *(ii) abort attacks* gain information about other parties through abort of transactions, and *(iii)* $n-1$ *attacks* where the tumbler refuses all but one payment.

b) Bolt: Bolt [43] aims to offer privacy-preserving payment channels such that multiple payments on a single channel are unlinkable to each other. Assuming the channels are indeed funded with anonymized capital (e.g. using anonymized assets [110]), Bolt payments are anonymous.

2) Privacy Enhancing Multi-Hop Payment Protocols:

a) Rayo and Fulgor [117]: : *Unlinkable HTLC* [117], is a cryptographic primitive that ensures that a HTLC in a payment path is built upon a different and unrelated hash value. Each intermediate user is provided with two hash values $y_i := H(x_i)$ and $y_{i+1} := H(x_i \oplus x_{i+1})$ and the value x_{i+1}. The intermediate user is also provided with a ZKP that the preimage of y_{i+1} is the same as the one of y_i, just skewed by the value x_{i+1}. Rayo and Fulgor achieve the same functionality as HTLC but prevent linkability of payments.

b) Anonymous MHL (AMHL): : Unlike Rayo and Fulgor, AMHL protocol embeds the synchronization condition within a public key, improving upon Rayo and Fulgor on efficiency and privacy (i.e. the synchronization condition does not explictly appears on the transaction).

C. Observations

Myth 3 insights: While the default transaction privacy on layer-two is likely better than on layer-one, layer-two transactions cannot by default be considered private.

TumbleBit achieves unlinkability but not payment anonymity. BOLT does not support Bitcoin but offers stronger privacy guarantees. Even in the simplified PCH setting, it seems that tradeoffs between privacy and compatibility are required. Multi-hop payment protocols do not enforce single hop privacy guarantees (e.g a user learns predecessor and successor in a payment path) at the gain of global privacy guarantees such as value privacy and relationship anonymity. As demonstrated in AMHL, it is possible to achieve privacy guarantees and backwards compatibility with most existing blockchains. State channels and commit-chains demonstrate interesting functionalities based on the expressiveness of rich scripting languages. These protocols, however, to date do not aim at providing anonymity and privacy guarantees from the commit-chain operator. Instead, privacy is considered an orthogonal and open research problem.

VI. Security

Blockchains experienced the thorough study of their consensus [5], [128] and network [129] security. Security is fundamental to distributed ledgers, as the shift of trust assumptions from a single custodian to a decentralized non-custodial network only prevents the loss of funds if the system's security properties are sound. Layer-two research undoubtedly benefits from this body of literature but also requires the introduction of new requirements, trust assumptions and adversarial models.

A. Security Threats

a) Hot Wallets: Channels require unanimous agreement for state updates and thus need all involved parties to be online with access to their signing keys. Keeping keys online in a *hot wallet*, i.e. a list of private signing keys, is critical — parties become prime targets for adversaries. This may potentially limit the capacity of PCN as channel operators must exercise caution about the number of coins they are willing to risk. While parties in commit-chains face similar challenges, the commit-chain operator is not required to stake assets

to facilitate payments (when providing delayed transaction finality) — and receivers in e.g. NOCUST can moreover remain offline at the time of payment.

b) Online Assumption: One concern for layer-two protocols is the assumption that parties remain online and fully synchronized with the PCN and blockchain. With the exception of RbI and RbT, channel designs require parties to watch for malice closures with outdated states. For commit-chains, users are required to either surface online in periodic intervals (i.e. each *eon* in NOCUST) or to watch the blockchain continuously for malicious exits [98]. If parties fail to monitor the layer-two protocol, the commit-chain operator can perform *execution forks* [66] to steal the offline user's assets. Watching services alleviate the online assumption (cf. Section III-H).

c) Blockchain Reliability and Mass Exits: Layer-two designs assume that the underlying blockchain accepts transactions eventually. Miners include transactions based on fees and under network congestion, transaction fees can grow from several cents to $50 [130]. Under congestion, parties may fail to meet deadlines to settle disputes and for PCN/PCH this might result in unfairly closed channel states. Under a mass exit (e.g. when many users close channels), blockchain users might enter in a bidding-war for their on-chain exit transactions to confirm. Commit-chain operators are single points of availability failure and require, if halted, all users to withdraw their assets. Contrary to PCN, commit-chains do not require a deadline for users to withdraw their coins, mitigating the transaction fee bidding war. A NOCUST operator is forced by the smart contract to halt given one successful dispute by a user, allowing all users to exit fairly. A Plasma Cash operator is not halted, even if operator's misbehavior is provably reported.

d) Consistency Proofs: Many layer-two protocols rely on challenge-response protocols to detect and prove misbehavior using the blockchain as a recourse for disputes. An alternative strategy to enforce consistency of an off-chain protocol is to let the blockchain verify a succinct proof attesting to consistency of the second layer's state. While ZKPs [131] suffer from expensive on-chain verification costs (approximately 650k gas on Ethereum) per proof [132], they can attest to potentially large state transitions which otherwise would require significant on-chain resources. For commit-chains, zkSNARKS were shown to enforce consistent checkpoints [28], leaving data availability of the external ledger as the remaining challenge vector.

e) Security of Synchronizing Protocols: One concern with HTLCs-based protocols is the *wormhole attack* [76] that allows an adversary situated in a multi-hop payment path to steal transaction fees by excluding the honest users from the successful completion of a payment. The adversary thereby forces the honest user to lock coins during the payment commit and bypasses the user during the release phase of a payment. The Lightning Network is currently vulnerable and the AMHL protocol is being considered to mitigate this issue [133].

Another concern with synchronizing payments, *the American Call option*, is that an adversary can set up a multi-hop payment and not release the trigger to finalize the payment. As a result, the coins are locked up until the transfer's expiry time and the adversary can perform this lock-up for free as all coins are refunded. Thus there is a loss of opportunity cost as the coins are locked up.

B. Layer-Two Security Notions

Formal security studies focus on the notion of *balance security*, both in the payment-hub [41], [43] and multi-hop [117] settings. This notion intuitively defines that layer-two protocols must achieve two properties: (i) the adversary cannot extract more funds than previously allocated in the channel's funding; (ii) honest users do not lose funds even when other parties collude. As with privacy, this security notion has been formalized in both paradigms: cryptographic games and the UC framework. BOLT and TumbleBit are the two payment-hub systems with formal security guarantees, while Rayo & Fulgor, AMHL and Perun provide formal security guarantees in the multi-hop setting. NOCUST provides a thorough study of *balance security* for commit-chains.

C. Observations

The security guarantees of layer-two transactions rely not only on the parent chain's consensus guarantees and on-chain security collateral. Data availability concerns and blockchain congestion threats introduce a new dimension of game-theoretic challenges that are not considered by current formal definitions. For instance, current UC definitions consider the blockchain as ideal components, which disregards the mass-exit concern.

Myth 4 insights: Pre-allocated on-chain collateral enables layer-two protocols to offer near instant finality for off-chain transactions. Note that the more intermediate nodes a payment path entertains, the more collateral needs to be locked for atomic payment execution. Once a payment path is discovered, the dominant transaction fee costs are thus the collateral interest rates and associated security risks — not the transaction size (in bytes).

VII. Conclusion

This SoK discusses that, contrary to common beliefs, layer-two protocols enable blockchains to scale without modification on the base layer (cf. *Myth 1*). Those performance improvements, however, result in different security guarantees for off-chain payments than on-chain transactions. Interestingly, commit-chains enable secure off-chain transaction without

TABLE IV
Comparison of layer-two transaction designs.

	Channel	Channel Hub	Commit-Chain
Topology	Mesh	Star	Star
Lifecycle	3-phase	3-phase	Periodic commit
Compatibility	Any chain	Any chain	Smart Contract chain
Privacy	value privacy, relationship anonymity	payment anonymity, unlinkability	×
Offline TX Reception	×	×	✓
Mass-Exit Security	×	×	✓(payments)
TX Finality	Instant	Instant	Delayed or Instant
Instant TX Collateral	Full	Full	Reusable [28]
Delayed TX Collateral	NA	NA	0
Collateral Allocation	$O(n)$ on-chain	$O(n)$ on-chain	$O(1)$ on-chain [28]
User On-Boarding	On-chain TX	On-chain TX	Off-chain [28]

collateralizing the full off-chain transaction volume (cf. *Myth 2*). Although off-chain transactions are not recorded on the public blockchain, they cannot be considered private by default and entail a number of unsolved privacy issues (cf. *Myth 3*). We explicitly lay out the shift in transaction costs from transaction size (in bytes) to transaction value (cf. *Myth 4*). In this paper, we lower the *barrier to entry* to study layer-two protocols and objectively compare the three major design categories channels, channel hubs and commit-chains in Table IV. Finally, we derive open challenges, summarized in the Appendix, that would greatly enrich the state-of-the-art knowledge.

ACKNOWLEDGMENTS

The authors would like to thank Alexei Zamyatin and Sam Werner for their valuable feedback on earlier paper versions.

This work has been partially supported by the Austrian Research Promotion Agency through the Bridge-1 project PR4DLT (grant agreement 1380869); by EPSRC Standard Research Studentship (DTP) (EP/R513052/1); by COMET K1 SBA, ABC; by Chaincode Labs; by the Austrian Science Fund (FWF) through the Lisa Meitner program; by the Ethereum Foundation, Ethereum Community Fund and Research Institute.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, available at: https://bitcoin.org/bitcoin.pdf.

[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[3] "Zcash," available at: https://z.cash/.

[4] "Litecoin," available at: https://litecoin.org/.

[5] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Conference on Computer and Communications Security*. ACM, 2016, pp. 3–16.

[6] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[7] VISA, "Visa inc. at a glance," 2015, available at: https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf.

[8] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[9] V. Buterin, "Slasher: A punitive proof-of-stake algorithm," 2014, available at: https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/.

[10] Anon., "Casper," 2018, available at: https://github.com/ethereum/casper.

[11] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Symposium on Security and Privacy*, 2014, pp. 475–490.

[12] T. Hønsi, "Spacemint-a cryptocurrency based on proofs of space," *IACR Cryptology ePrint Archive*, 2017.

[13] "Sawtooth," 2019, available at: https://intelledger.github.io/introduction.html.

[14] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *USENIX Security Symposium*, 2016, pp. 279–296.

[15] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and P. Saxena, "Scp: A computationally-scalable byzantine consensus protocol for blockchains," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1168, 2015.

[16] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *31 International Symposium on Distributed Computing*, 2017, p. 6.

[17] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *Symposium on Networked Systems Design and Implementation)*, 2016, pp. 45–59.

[18] Anon., "Sharding roadmap," 2019, available at: https://github.com/ethereum/wiki/wiki/Sharding-roadmap.

[19] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Conference on Computer and Communications Security*. ACM, 2016, pp. 17–30.

[20] A. E. Gencer, R. van Renesse, and E. G. Sirer, "Service-oriented sharding with aspen," *arXiv preprint arXiv:1611.06816*, 2016.

[21] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," *URL: http://www.opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 2014.

[22] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "Consensus in the age of blockchains," *arXiv preprint arXiv:1711.03936*, 2017.

[23] "Bitcoin cash," 2008, available at: https://www.bitcoincash.org.

[24] M. Hearn, "Micro-payment channels implementation now in bitcoinj," 2013, available at: https://bitcointalk.org/index.php?topic=244656.0.

[25] Anon., "bitcoinj," 2019, available at: https://bitcoinj.github.io/.

[26] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[27] J. Poon and T. Dryja, "The bitcoin lightning network: scalable off-chain instant payments," 2016, available at: https://lightning.network/lightning-network-paper.pdf.

[28] R. Khalil, A. Gervais, and G. Felley, "Nocust–a securely scalable commit-chain," 2018, available at: https://eprint.iacr.org/2018/642.pdf.

[29] B. T. AG, "Raiden network," 2019, available at: https://raiden.network/.

[30] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," *arXiv preprint arXiv:1702.05812*, 2017.

[31] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment channels over cryptographic currencies," in *Symposium on Security and Privacy*, 2019.

[32] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in credit networks," in *Network and Distributed System Security Symposium*, 2017.

[33] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," 2018.

[34] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing cryptocurrency with the spider network," *arXiv preprint arXiv:1809.05088*, 2018.

[35] C. A. Sunshine, "Source routing in computer networks," *SIGCOMM Computer Communication Review*, vol. 7, no. 1, pp. 29–33, 1977.

[36] Anon., "Lightning-onion," 2018, available at: https://github.com/lightningnetwork/lightning-onion.

[37] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," 2016, available at: https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf.

[38] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Conference on Computer and Communications Security*. ACM, 2017, pp. 439–453.

[39] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks," *Royal Society open science*, vol. 5, no. 8, p. 180089, 2018.

[40] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," 2017, available at: https://plasma.io/plasma.pdf.

[41] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," 2017.

[42] E. Heilman, S. Lipmann, and S. Goldberg, "The arwen trading protocols."

[43] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Conference on Computer and Communications Security*. ACM, 2017, pp. 473–489.

[44] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 43–60.

[45] K. Atlas, "The inevitability of privacy in lightning networks," 2017, available at: https://www.kristovatlas.com/the-inevitability-of-privacy-in-lightning-networks/.

[46] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *Symposium on Security and Privacy*. IEEE, 2015, pp. 104–121.

[47] J. Lind, I. Eyal, F. Kelbert, O. Naor, P. Pietzuch, and E. G. Sirer, "Teechain: Scalable blockchain payments using trusted execution environments," *arXiv preprint arXiv:1707.05454*, 2017.

[48] J. Lind, I. Eyal, P. Pietzuch, and E. G. Sirer, "Teechan: Payment channels using trusted execution environments," *arXiv preprint arXiv:1612.07766*, 2016.

[49] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "Fastkitten: Practical smart contracts on bitcoin."

[50] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[51] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "{ROTE}: Rollback protection for trusted execution," in {*USENIX*} *Security Symposium*, 2017, pp. 1289–1306.

[52] T. Neudecker and H. Hartenstein, "Network layer aspects of permissionless blockchains," *IEEE Communications Surveys & Tutorials*, 2018.

[53] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Conference on Peer-to-Peer Computing*, 2013, pp. 1–10.

[54] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer, "bloxroute: A scalable trustless blockchain distribution network," 2018, available at: https://bloxroute.com/wp-content/uploads/2018/03/bloXroute-whitepaper.pdf.

[55] A. Gervais, S. Capkun, G. O. Karame, and D. Gruber, "On the privacy provisions of bloom filters in lightweight bitcoin clients," in *Computer Security Applications Conference*, 2014, pp. 326–335.

[56] "Bitcoin fibre," 2019, available at: http://www.bitcoinfibre.org/.

[57] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing. fast money grows on trees, not chains." *IACR Cryptology ePrint Archive*, vol. 2013, no. 881, 2013.

[58] S. D. Lerner, "Decor + hop: A scalable blockchain protocol," available at: https://scalingbitcoin.org/papers/DECOR-HOP.pdf.

[59] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol." *IACR Cryptology ePrint Archive*, vol. 2016, p. 1159, 2016.

[60] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. Van Renesse, "{REM}: Resource-efficient mining for blockchains," in {*USENIX*} *Security Symposium*, 2017, pp. 1427–1444.

[61] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[62] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake." *IACR Cryptology ePrint Archive*, vol. 2016, p. 919, 2016.

[63] M. Milutinovic, W. He, H. Wu, and M. Kanwal, "Proof of luck: an efficient blockchain consensus protocol," in *Proceedings of the 1st Workshop on System Software for Trusted Execution*. ACM, 2016, p. 2.

[64] M. Borge, E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Proof-of-personhood: Redemocratizing permissionless cryptocurrencies," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2017, pp. 23–26.

[65] J. Spilman, "[bitcoin-development] anti dos for tx replacement," 2013, available at: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html.

[66] P. McCorry, S. Bakshi, I. Bentov, A. Miller, and S. Meiklejohn, "Pisa: Arbitration outsourcing for state channels." *IACR Cryptology ePrint Archive*, vol. 2018, p. 582, 2018.

[67] J. Joleman, L. Horne, and L. Xuanji, "Counterfactual: Generalized state channels," 2018, available at: https://l4.ventures/papers/statechannels.pdf.

[68] P. McCorry, C. Buckland, S. Bakshi, K. Wüst, and A. Miller, "You sank my battleship! a case study to evaluate state channels as a scaling solution for cryptocurrencies," 2019, available at: https://pdfs.semanticscholar.org/284e/2003a93836ae70c1af0ae922bd9d62473f75.pdf.

[69] P. McCorry, M. Möser, S. F. Shahandasti, and F. Hao, "Towards bitcoin payment networks," in *Australasian Conference on Information Security and Privacy*. Springer, 2016, pp. 57–76.

[70] C. Decker, R. Russell, and O. Osuntokun, "Eltoo: A simple layer2 protocol for bitcoin," 2018, available at: https://blockstream.com/eltoo.pdf.

[71] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in {*USENIX*} *Security Symposium*, 2018, pp. 1353–1370.

[72] T. Close and A. Stewart, "Forcemove: an n-party state channel protocol," 2018.

[73] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Conference on Computer and Communications Security*. ACM, 2018, pp. 949–966.

[74] A. R. Pedrosa, M. Potop-Butucaru, and S. Tucci-Piergiovanni, "Lightning factories," 2019.

[75] B. Wiki, "Hashed timelock contracts," 2019, available at: https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts.

[76] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *Network and Distributed System Security Symposium*, 2019.

[77] A. Poelstra, "Lightning in scriptless scripts," Mailing list post, https://lists.launchpad.net/mimblewimble/msg00086.html.

[78] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *IEEE Security and Privacy*. IEEE, 2019.

[79] Intel, "Intel software guard extensions (intel sgx)," 2019, available at: https://software.intel.com/en-us/sgx.

[80] I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware." *IACR Cryptology ePrint Archive*, vol. 2017, p. 1153, 2017.

[81] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure:{SGX} cache attacks are practical," in *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[82] M. Dong, Q. Liang, X. Li, and J. Liu, "Celer network: Bring internet scale to every blockchain," *arXiv preprint arXiv:1810.00037*, 2018.

[83] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing for anonymous and private internet connections," *Communications of the ACM*, vol. 42, no. 2, pp. 39–40, 1999.

[84] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[85] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows*. Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts, 1988.

[86] P. F. Tsuchiya, "The landmark hierarchy: a new hierarchy for routing in very large networks," in *SIGCOMM Computer Communication Review*, vol. 18. ACM, 1988, pp. 35–42.

[87] C. H. Papadimitriou and D. Ratajczak, "On a conjecture related to geometric routing," *Theoretical Computer Science*, vol. 344, no. 1, 2005.

[88] S. Roos, M. Beck, and T. Strufe, "Anonymous addresses for efficient and resilient routing in f2f overlays," in *Conference on Computer Communications*, 2016, pp. 1–9.

[89] S. Werman and A. Zohar, "Avoiding deadlocks in payment channel networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, 2018.

[90] D. Piatkivskyi and M. Nowostawski, "Split payments in payment networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, J. Garcia-Alfaro, J. Herrera-Joancomartí, G. Livraga, and R. Rios, Eds. Cham: Springer International Publishing, 2018, pp. 67–75.

[91] T. Dryja, "Unlinkable outsourced channel monitoring," 2016, available at: https://scalingbitcoin.org/transcript/milan2016/unlinkable-outsourced-channel-monitoring.

[92] O. Osuntokun, "Hardening lightning, harder, better, faster stronger," 2015, available at: https://cyber.stanford.edu/sites/g/files/sbiybj9936/f/hardening_lightning_updated.pdf.

[93] G. Avarikioti, F. Laufenberg, J. Sliwinski, Y. Wang, and R. Wattenhofer, "Towards secure and efficient payment channels," *arXiv preprint arXiv:1811.12740*, 2018.
[94] G. Avarikioti, G. Janssen, Y. Wang, and R. Wattenhofer, "Payment network design with fees," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 76–84.
[95] G. Konstantopoulos, "Plasma cash: Towards more efficient plasma constructions," 2019, available at: https://github.com/loomnetwork/plasma-paper/blob/master/plasma_cash.pdf.
[96] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*. Springer, 1987, pp. 369–378.
[97] R. Khalil, A. Gervais, and G. Felley, "Tex–a securely scalable trustless exchange."
[98] "Plasma cash: Plasma with much less per-user data checking," 2018, available at: https://ethresear.ch/t/plasma-cash-plasma-with-much-less-per-user-data-checking/1298.
[99] D. Robinson, "Plasma debit: Arbitrary-denomination payments in plasma cash," 2018, available at: https://ethresear.ch/t/plasma-debit-arbitrary-denomination-payments-in-plasma-cash/2198.
[100] B. Rao, "Gluon plasma: a plasma variant for non-custodial exchanges," 2018, available at: https://leverj.io/GluonPlasma.pdf.
[101] B. Jones and K. Fichter, "More viable plasma," 2018, available at: https://ethresear.ch/t/more-viable-plasma/2160.
[102] "Plasma snapp - fully verified plasma chain," 2018, available at: https://ethresear.ch/t/plasma-snapp-fully-verified-plasma-chain/3391.
[103] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*. Springer, 1983, pp. 199–203.
[104] B. Yang and H. Garcia-Molina, "Ppay: micropayments for peer-to-peer systems," in *Proceedings of the 10th ACM conference on Computer and communications security*. ACM, 2003, pp. 300–310.
[105] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun, "Misbehavior in bitcoin: A study of double-spending and accountability," *ACM Transactions on Information and System Security (TISSEC)*, vol. 18, no. 1, p. 2, 2015.
[106] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, "Evaluating user privacy in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 34–51.
[107] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 127–140.
[108] R. Böhme, N. Christin, B. Edelman, and T. Moore, "Bitcoin: Economics, technology, and governance," *Journal of Economic Perspectives*, vol. 29, no. 2, pp. 213–38, 2015.
[109] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan *et al.*, "An empirical analysis of traceability in the monero blockchain," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 143–163, 2018.
[110] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Symposium on Security and Privacy*. IEEE, 2014, pp. 459–474.
[111] N. T. Courtois and R. Mercer, "Stealth address and key management techniques in blockchain systems." in *ICISSP*, 2017, pp. 559–566.
[112] A. Bender, J. Katz, and R. Morselli, "Ring signatures: Stronger definitions, and constructions without random oracles," in *Theory of Cryptography Conference*. Springer, 2006, pp. 60–79.
[113] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *Journal of cryptology*, vol. 1, no. 2, pp. 77–94, 1988.
[114] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, "An empirical analysis of anonymity in zcash," in *USENIX Security Symposium*, 2018, pp. 463–477.
[115] A. Hinteregger and B. Haslhofer, "An empirical analysis of monero cross-chain traceability," *CoRR*, vol. abs/1812.02808, 2018. [Online]. Available: http://arxiv.org/abs/1812.02808
[116] A. Kumar, C. Fischer, S. Tople, and P. Saxena, "A traceability analysis of monero's blockchain," in *ESORICS*, 2017, pp. 153–173.
[117] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 455–471. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134096
[118] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina, "Privacy preserving payments in credit networks," in *Network and Distributed Security Symposium*, 2015.
[119] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Symposium on Security and Privacy*, 2013, pp. 397–411.
[120] B. Wiki, "Bitcoin mixing," 2018, available at: https://en.bitcoin.it/wiki/Mixing_service.
[121] ——, "Coin join," 2019, available at: https://en.bitcoin.it/wiki/CoinJoin.
[122] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 345–364.
[123] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "P2P mixing and unlinkable bitcoin transactions," in *Network and Distributed System Security Symposium*, 2017.
[124] T. Ruffing and P. Moreno-Sanchez, "Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin," in *BITCOIN Workshop*, 2017, pp. 133–154.
[125] P. Moreno-Sanchez, T. Ruffing, and A. Kate, "Pathshuffle: Credit mixing and anonymous payments for ripple," *PoPETs*, vol. 2017, no. 3, p. 110, 2017.
[126] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "Coinparty: Secure multi-party mixing of bitcoins," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY 2015, San Antonio, TX, USA, March 2-4, 2015*, 2015, pp. 75–86. [Online]. Available: https://doi.org/10.1145/2699026.2699100
[127] G. Danezis and A. Serjantov, "Statistical disclosure or intersection attacks on anonymity systems," in *International Workshop on Information Hiding*, 2004.
[128] K. Wüst and A. Gervais, "Ethereum eclipse attacks," ETH Zurich, Tech. Rep., 2016.
[129] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun, "Tampering with the delivery of blocks and transactions in bitcoin," in *Conference on Computer and Communications Security*. ACM, 2015, pp. 692–705.
[130] R. Browne, "Big transaction fees are a problem for bitcoin but there could be a solution," 2017, available at: https://www.cnbc.com/2017/12/19/big-transactions-fees-are-a-problem-for-bitcoin.html.
[131] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 781–796.
[132] V. Buterin, "On-chain scaling to potentially 500 tx/sec through mass tx validation," 2018, available at: https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477.
[133] C. Fromknecht, "2p-ecdsa: Two-party ecdsa multisignatures," Github project, https://github.com/cfromknecht/tpec,.
[134] A. Gervais, G. O. Karame, V. Capkun, and S. Capkun, "Is bitcoin a decentralized currency?" *IEEE security & privacy*, vol. 12, no. 3, pp. 54–60, 2014.
[135] A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. G. Sirer, "Decentralization in bitcoin and ethereum networks," *arXiv preprint arXiv:1801.03998*, 2018.
[136] "Roll-up," available at: https://github.com/barryWhiteHat/roll_up.

APPENDIX

A. Open Challenges

Layer-Two Cost Quantification: A comprehensive study of the real economic costs of layer-two transactions, ideally comparing different channel, synchronization and commit-chain proposals. Only if the layer-two transaction fees and security concerns are inferior to the offered on-chain, then it is rational to perform layer-two transactions.

Layer-One Congestion: Existing work mostly ignores the threat of blockchain congestion. One future avenue would be to design congestion-aware [68] layer-two protocols.

Cross Commit-Chain Payments and Routing: We are not aware of work covering atomic payments across a more decentralized network of commit-chains.

Private Commit-Chain: Contrary to selected payment channel hubs [41], [43], existing commit-chains do not provide any privacy guarantees from the commit-chain operator.

Quantification of Layer-Two Decentralization: While related work discusses layer-one decentralization [134], [135], no work has yet covered layer-two decentralization.

Channel Factories on Commit-Chains: Commit-chains might enable to spawn payment channels among their users (similar to the idea of virtual channels) potentially foregoing costly channel initialization costs.

Compression-Chains: Compression-chain techniques such as Roll-up [136] aim to reduce on-chain transaction footprint. A transaction only requires 9 bytes on-chain, while a ZKP certifies the validity of signatures. While they might not be considered layer-two protocols and may not scale to the same extent, they solve data-availability concerns and strengthen the security properties. A thorough analysis of compression-chains is missing.

Formal Security/Privacy: A systematic method to develop security and privacy notions for layer-two protocols, faithfully including their interaction with layer-one, constitutes an interesting direction for future research.