Strengthening Multi-hop Channels via Strategic Mesh Connections

Shuyang Tang[1] and Sherman S. M. Chow[2]

[1] Dept. of Computer Science & Engineering, Shanghai Jiao Tong University
[2] Department of Information Engineering, Chinese University of Hong Kong

Abstract. Scalable multi-hop cryptocurrency transactions are enabled by payment channel networks (PCNs) like the Lightning Network, which execute most transactions off-chain, greatly reducing on-chain activity. However, unrestricted channel formation often leads to centralization, creating a natural oligarchy where a few nodes gain disproportionate influence, weakening decentralization and raising risks of targeted attacks. Establishing a structured network in PCNs mitigates this issue by encouraging users to allocate channels strategically, resisting partitions.
Reframing PCN connectivity, we introduce a "strategic mesh channels" model that strengthens multi-hop connectivity by creating virtual channels that connect individual channels while retaining user autonomy. Melding theoretical optimization with practical implementation, our simulations based on our smart-contract implementation show that, with an appropriate participation rate, our design significantly enhances decentralization, connectivity, and attack resilience over standard PCNs. Additionally, average users receive economic incentives to allocate channels strategically, driving broader adoption of this structured approach. Network resilience and fairness thus improve, challenging the assumption that decentralization requires unregulated, anarchic formation.

1 Introduction

Scalability challenges in blockchains, *e.g.*, constrained transaction throughput and long confirmation times, have spurred the creation of off-chain solutions. High-throughput off-chain transactions are made possible by *payment channels* and *payment channel networks* (PC and PCN) [23,44], reducing on-chain load. Establishing a payment channel requires a deposit of money m as *custody*, which determines the maximum amount transferable within the channel at any time. Rather than executing every transaction on-chain, PCs facilitate direct off-chain asset transfers with instant confirmations but with fairness and reduced fees. Money held in custody serves as both the deposit for final on-chain settlements and a safeguard, reinforcing fair dispute resolution for off-chain transactions. As multiple PCs interconnect, they form a PCN that enables multi-hop payments, allowing transactions to be routed between users without direct links. Network-wide payments in a PCN incur costs proportional to the number of channels traversed, yet remain lower than on-chain fees because transactions require only local channel updates rather than global consensus.

Standard PCN schemes allow users to independently determine their network topology, often selecting adjacent nodes (*i.e.*, nodes sharing a channel) based on individual preferences. Ideally, a large-scale PCN would emerge, supporting practical (*i.e.*, well-connected) and decentralized multi-hop payments between any pair of users, ensuring strong and robust network connectivity. Given the current decentralized approach, the "price of anarchy" arises as users are incentivized to connect with well-connected peers to optimize their own connectivity. More often than not, networks evolve into suboptimal structures as a result. As users increasingly form connections with a limited set of well-connected nodes, these hubs gain disproportionate influence, culminating in a natural oligarchy that consolidates control and renders the network vulnerable to adaptive attacks.

Having these observations, we introduce *strategic mediated connections*—a structured approach in which users establish a portion of their channels based on a predefined network design. As part of our model, the PCN is represented as an undirected graph, with edges corresponding to either *prescribed* channels or *open-choice* channels. To ensure a well-connected topology, the network is initialized at genesis with a (γm)-regular *basis graph*, in which each vertex connects to γm others. Compliance with this structure is enforced through a *participation rate* γ $(0 \leq \gamma \leq 1)$, requiring each node to establish the prescribed channels according to this basis graph before freely creating the remaining $(1-\gamma)m$ open-choice channels. Handling new nodes, the system requires that they deposit custody of their channels in a smart contract that assigns them a vertex and automatically establishes their prescribed channels. A node can establish open-choice channels with others after fulfilling the compliance.

Expecting all nodes to occupy only one vertex in the predefined network structure is impractical given varied client demands. Consequently, we allow users to occupy multiple nodes (vertices) and establish *super channels*—ideal channels with infinite *capacity* (the maximum admissible amount transferable through them) and zero fees—between them. Here, we set a parameter m for the number of channels; it is only a minimum requirement as participants may form multiple sets of m channels.[3] Overall, this structure imposes constraints that enhance the network topology, decentralization, and resilience against attacks.

1.1 Motivation from the Lightning Network Topology in Practice

Reviewing Findings on LN. The Lightning Network (LN) is currently the most widely used PCN. Tochner *et al.* [51] found that 60% (80%) of multi-hop channels pass through only 5 (15) nodes, highlighting a significant concentration of traffic through a few central nodes. Similarly, Lisi *et al.* [29] revealed that 37.7% of nodes[4] have a degree of 1, whereas only 92 nodes exhibit a degree exceeding 100.

[3] Security concerns generally advise against a single participant spawning multiple nodes due to potential liveness and safety risks, but in our system, occupying each node requires significant custody (redeemable) and fees (non-redeemable), thereby encouraging users with higher transfer demands to occupy multiple nodes.

[4] This paper, *e.g.*, protocol specifications, may use vertex and node interchangeably.

Table 1. A snapshot of LN on September 26, 2024

# Nodes	16,810	Average Distance	9.1
# Channels	54,432	Diameter	13
Channels per Node	6.9	Cut Channels (%)	49.7%
Clustering Coefficient (%)	12.3%	Cut Nodes (%)	12.8%
Transitivity (%)	2.7%		

Most peripheral nodes connect to only a limited set of nodes, forming *bouquet* patterns. These "bouquet roots" constitute only 1% of the network yet connect to 2,285 peripheral nodes ("roses"), representing 29% of the network. The removal of as few as 10 bouquet roots disconnects 10.4% of nodes and 6.4% of channels, underscoring the high centralization and suboptimal connectivity and clustering.

A further analysis of the latest LN topology is conducted using third-party data[5] (Table 1). We evaluate connectivity and local clustering via several metrics. First, we assess connectivity by counting the *cut channels* and *cut nodes* whose removal partitions the PCN into separate components. Fewer cut channels and nodes indicate improved resilience against network splits, which enhances resistance to censorship and network partitioning.

Next, we consider clustering metrics, which capture how interconnected a node's neighbors are. The *clustering coefficient* is defined as the average of the local clustering coefficients across all nodes, where a node's *local coefficient* is the fraction of connections among its neighbors. The clustering coefficient quantifies the likelihood that two nodes connected to the same node are also directly connected to each other, forming a triangle. A high coefficient indicates that transactions tend to concentrate within tightly-knit groups rather than being uniformly spread across the network, leading to liquidity centralization and routing inefficiencies. In contrast, a low coefficient suggests a more evenly connected network, reducing reliance on specific hubs and improving decentralization. Unlike cut nodes, which indicate the centralization of individual nodes, these metrics reflect clique-based centralization. *Transitivity* quantifies the proportion of actual triangles relative to potential triangles, where a value of 1 signifies that every path of length 2 connects to a neighbor of the starting node.

Finally, the *average distance* and *diameter* (*i.e.*, the worst-case distance) measure how efficiently transactions can be routed (see §2.2). Shorter routes generally reduce fees and increase success rates, thereby making the network more suitable for practical payments. Together, these metrics indicate significant centralization within the LN, a trend that continues to intensify [56].

Comparing LN with Ramanujan Graphs of Lubotzky–Phillips–Sarnak (LPS). To address issues reflected in these metrics, we consider LPS graphs [32], a family of Ramanujan expander graphs known for their high regularity and favorable spectral properties. Their relevance to PCNs is discussed in §2.3. Their construction relies on quotient spaces of the modular group and related groups such as PSL

[5] from `https://bitcoinvisuals.com/lightning` on September 26, 2024

Table 2. Properties of LPS Graphs for $(p, q) = (5, 23)$ (left) and $(7, 23)$ (right)

# Nodes	$12,167$	Av. Distance	4.31
# Channels	$36,501$	Diameter	7
Channels/Node	6	Cut Channels	0%
Clust. Coeff.	0.27%	Cut Nodes	0%
Transitivity	0.48%		

# Nodes	$12,167$	Av. Distance	3.95
# Channels	$48,668$	Diameter	6
Channels/Node	8	Cut Channels	0%
Clust. Coeff.	0.39%	Cut Nodes	0%
Transitivity	0.53%		

and PGL (*i.e.*, projective special and general linear groups). See §1.2 and §4 for more details. Table 2 shows the same metrics for a comparable number of nodes. By our design, the average number of channels per node must be even (equal to $p + 1$) and cannot be 7, as in Table 1. So, we present the cases for $p = 5$ and $p = 7$. Our analysis indicates a substantial improvement in topology metrics. We will later explore how to integrate these prescribed expander graphs with open-choice channels in a decentralized manner.

1.2 Technical Roadmap: High-Level Design and Research Methods

Smart Contract for Prescribed and Open-Choice Channels. Our smart contract Π_{sc} assigns a vertex $u \in \mathcal{S}$ from the prescribed graph $\mathcal{S}$ to user P_u upon request, provided the user deposits sufficient custody amount mQ, where m is the number of channels per vertex and Q is the deposit per channel. It allows users to establish $(1 - \gamma)m$ open-choice channels between this vertex and others. Counterintuitively, when the vertex is released to P_u, the γm prescribed channels are already established. The node-join subroutine operates similarly to channel setups in typical PCNs. Users can conduct off-chain transfers and update channel states, as described in §3.1. A key role of Π_{sc} is to secure channel withdrawals. As with bilateral PCs [44,26] (see §2.2), a dispute resolution mechanism is necessary, which we discuss in Appendix B. Multi-hop payments are handled using two routing methods, one for light nodes and one that optimizes canonical MF-based routing, detailed in §3.3.

Expander-based Network Structure. Our prescribed structure $\mathcal{S}$ is configurable. We select the LPS expander graph for several reasons: (1) Expander graphs exhibit strong connectivity, yielding a robust topology that resists bisecting attacks and enhances decentralization. (2) They feature theoretically bounded and empirically small average distances and diameters, minimizing the number of hops required for multi-hop payments. (3) They often yield highly symmetric graphs, promoting fairness among vertices. (4) Their large expansion degrees (see §4) hinder clique formation and further boost decentralization. The LPS graph, defined by two odd primes p and q, results in a $(p+1)$-regular graph with $\frac{q^3-q}{2}$ (or $q^3 - q$) nodes if p is (not) a quadratic residue modulo q. More details are in §4.

Prototype, Evaluations, and Incentives. Our proposed Light Tower Protocol provides a concrete framework for implementing this structured approach to effectively establish strategic mediated connections. We implemented a prototype in Solidity (§5.1) and assessed its impact on decentralization (§5.2). Specifically, our experiments demonstrate that average-sized users are likely to generate higher revenues (Appendix G), thereby incentivizing them to join our network.

Contributions. We summarize our contributions as follows.

- *Investigating Centralization in PCNs:* We empirically analyze the inherent centralization tendencies in the LN, illustrating how autonomous channel formation contributes to vulnerabilities and inefficiencies in existing PCNs.
- *Proposing a Structured, Modular Approach:* We introduce "strategic mediated connections," a pioneering methodology that integrates a prescribed network structure based on expander graphs into PCNs. This approach balances user freedom with network-wide benefits, enhancing decentralization and robustness without imposing excessive constraints. We instantiate this using LPS graphs, provide a detailed protocol for bilateral PCNs, explore alternative multi-hop routing and payment methods, and analyze its security.
- *Developing a Practical Implementation:* We design and implement a smart contract-based protocol for bilateral PCNs that shows the simplicity and practicality of our approach. It includes secure channel establishment, multi-hop routing, and dispute resolution mechanisms that support deployment.
- *Showing Empirical Improvement via Simulation:* Extensive simulations validate our approach, showing marked improvements in network metrics like connectivity, resilience, and decentralization. Our results show that average users are economically incentivized to participate, which promotes adoption.
- *Bridging Theory and Practice in PCN Design:* Our work bridges the gap between theoretical constructs of optimal network topologies and their practical applications in real-world cryptocurrency networks, offering a scalable solution to centralization that can be readily implemented.

1.3 Related Works

Since their inception, various PCs have been proposed [23,43,14,44,13,26,4,20]. Multi-hop payments in PCNs [44] prompted research on routing and bottlenecks [33,46,47,53]. Censorship [41] and distributed denial-of-service [31,51,39] attacks have threatened PCNs (notably in 2018 [52]), and network centralization amplifies the impact and success of these attacks. Splitting attacks [27], which partition a PCN by removing minor nodes, have sparked heated debates [45,24].

State channels [38] have been proposed to extend PC functionality, while virtual channels [16,17,3,15] enable multi-hop payments even when intermediary nodes are offline. Many other techniques, *e.g.*, rebalancing [25], atomic multichannel updating [18], and trusted hardware [28], have also been discussed.

Many layer-two blockchain protocols were surveyed by Gudgeon *et al.* [22]. Sprites [38] coordinated multi-hop payments by smart contracts, reducing collateral overhead with partial withdrawals and deposits. Avarikioti *et al.* [6] first modeled PCN channel creation games to explore network structures and fee constraints for Nash equilibria. Later, Avarikioti *et al.* [8] defined a utility function for new LN users, provided an approximation algorithm for optimal solutions, and examined the parameter space for Nash equilibrium. These studies offer insight into user behavior and motivation under different network topologies.

Currently, no existing work has investigated how smart contract coordination can improve overall PCN topology. However, broader efforts to optimize cryptocurrency ecosystems while preserving decentralization and user autonomy have been explored, such as systematic market control of inflation via auction-based liquidity mechanisms [48]. Studies on PCN topology are linked to transaction structures, which adversaries can exploit to trace transactions in anonymous systems. Beyond unlinkability, maintaining the long-term sustainability of decentralized anonymity systems poses additional challenges [10].

2 Preliminaries

2.1 Notations and Assumptions

$A := B$ assigns B to A. m is the number of channels of a node, $[n]$ stands for $\{1, 2, \ldots, n\}$. N is the upper bound on the total number of nodes in the PCN, and Q is the custody of each payment channel. The power set of set S is $\mathcal{P}(S)$.

We use cryptographic hash function $H(\cdot)$ and (additively) homomorphic one-way function (HOWF) $g(\cdot)$. $(a_0, a_1, \ldots)_P$ denotes the message $\{a_i\}$ and P's signature on it. For off-chain messages, we may denote by $\langle a_0, a_1, \ldots \rangle$. Each public key serves for signatures and encryption for simplicity (while using two keys is safer in practice), as well as the address of a user, often its sole identifier. A vertex in our PCN topology $\mathcal{S}$ represents either a user or an empty slot to be filled in the future. A user can occupy more than one vertex.

Off-chain communications occur via private channels without updating the blockchain. We suppose nodes can communicate privately and efficiently if they know the recipient's physical address. Alternatively, knowing the public key enables delivery via a less efficient public P2P network, assumed to be asynchronous (with delayed, out-of-order, or lost messages). Channels between Π_{sc} and any party are assumed to be partially synchronized, with possible delays or out-of-order messages, but without losses. Since Π_{sc} is realized via smart contracts, interactions with Π_{sc} involve reading on-chain data and submitting transactions authenticated by blockchain consensus, so authentication between Π_{sc} is omitted.

2.2 PC, PCN, and Multi-hop Payments

Payment Channels (PCs). Off-chain PCs improve the scalability of cryptocurrencies by enabling instant off-chain transfers without incurring on-chain costs for each transfer. In unilateral channels, the sender deposits custody funds into a contract address, typically equal to the channel's maximum transferable amount. Funds can then be transferred off-chain through messages acting as checks, allowing the receiver to claim the accumulated transfers from the sender's custody.

A basic unilateral channel construction is outlined in Appendix A.1 for illustration. However, this paper focuses on the more general bilateral channels.

Bilateral channels, unlike unilateral ones, support transfers in both directions, with balances represented as pairs of values that sum to a constant total. Both participants must deposit funds and jointly sign any channel state transitions. This complicates channel withdrawal, as both users must confirm them, while one user may intentionally fail to respond, affecting the channel's liveness. Therefore, when either user issues a withdrawal request, the other has to issue disputes within a specific time if necessary. If no dispute is raised, the withdrawal can be completed unilaterally. This creates demand for *watchtower services* [5,36,7], which address data availability issues of node withdrawal in bilateral channels.

Payment Channel Networks (PCNs). A PCN comprises all payment channels, often modeled as a graph where nodes represent participants and edges represent PCs. Nodes (u, v) are *adjacent* if a channel exists between them. The function $\mathsf{N}\colon \mathcal{S} \to \mathcal{P}(\mathcal{S})$ maps each node u to its adjacent nodes (∂u). Two nodes are *connected* if a path of channels exists between them.

Routing and Multi-hop Locks. Routing in a multi-hop payment involves finding payment paths from payer A to payee B. Several methods for routing are discussed in Appendix E. Once a route is determined, multi-hop locks freeze the selected channels before the transfer occurs. Various schemes implement multi-hop locks, such as *hashed time-lock contracts* (HTLC) [34], which lock passing channels that are unlocked upon either timeout or hash preimage revelation. *Anonymous multi-hop locks* (AMHL) [35] improve security through the use of HOWFs. A brief review of HTLC and AMHL is provided in Appendix A.

2.3 Expander Graphs

Expander graphs ensure robust connectivity even when multiple channels fail, as their uniform structure prevents reliance on a few highly connected hubs, mitigating network splits. Unlike scale-free networks, where liquidity and routing capacity cluster around dominant nodes, expander graphs distribute connectivity more evenly, improving decentralization and making the network resistant to censorship and targeted attacks. Their high expansion factor minimizes clustering and mitigates partitions, reducing bottlenecks that could otherwise constrain transaction flow. Additionally, their bounded diameter ensures that multi-hop payments require fewer intermediate nodes, reducing transaction fees and increasing payment success rates. These structural advantages are particularly relevant in PCNs, where liquidity distribution and routing efficiency are critical. By integrating expander-based channels with user-defined open-choice connections, our model balances decentralization, efficiency, and user autonomy, enhancing the resilience and economic sustainability of the network.

3 Our Proposed Protocol

We now present our Light Tower Protocol, designed to streamline understanding and implementation through a modular approach, with a basis graph detailed in §4. It consists of two parts: off-chain protocols, specifying private interactions between two nodes of a channel, and on-chain protocols, managing public user interactions and the smart contract Π_{sc}. Appendix C presents the full pseudocode.

3.1 Off-Chain Protocols

Off-chain protocols manage payment channels between nodes u and v. Vertex u (v) is occupied by the node with address (*i.e.*, public key) A (B). We refer to a user by their vertex u or address A interchangeably, depending on the context. The channel state is $\mathsf{rec} = (\mathsf{state}_0, \dots, \mathsf{state}_\ell)$. The initial state (version $k = 0$) is $\langle u, v, 0, Q, 0, \_, \_, \_\rangle$, where $\_$ denotes empty. In general, a state has the form:

$$\langle u, v, k, R, \mathsf{frz}, I, \mathsf{sig}_A, \mathsf{sig}_B\rangle \ .$$

Here, R is the balance of node A, and $2Q - R$ is the balance of node B, where $2Q$ is the total capacity of the channel. frz is the amount of frozen funds (if any). I is a hash image, relevant only if $\mathsf{frz} \neq 0$. sig_A (sig_B) is a signature from A (B).

This tuple is symmetric and equivalent to $\langle v, u, k, 2Q - R, -\mathsf{frz}, I, \mathsf{sig}_B, \mathsf{sig}_A\rangle$ when Q is explicitly specified. We may refer to the same tuple using either form.

Initialization. The protocol exchanges the *physical* network addresses of u and v to establish off-chain communications. To initiate a handshake with B, A acquires the public key B from Π_{sc} and sends a handshake request in the form of $\langle \texttt{handshake-req}, A, B, u, v, c = \mathsf{Enc}(B, \mathsf{addr}_A)\rangle$.

Upon receipt, B verifies via Π_{sc} that A is associated with u, then decrypts to recover $\mathsf{addr}_A = \mathsf{Dec}(\mathsf{sk}_B, c)$. B then reciprocates by sending its encrypted network address to A, completing the setup of their off-chain communication.

Payment. A payment of amount a from A to B causes the state transition of

$$\langle u, v, k, R, 0, \_, \mathsf{sig}_A, \mathsf{sig}_B\rangle \mapsto \left\langle u, v, k+1, R-a, 0, \_, \mathsf{sig}'_A, \mathsf{sig}'_B\right\rangle ,$$

where sig'_A (sig'_B) is the signature on the new state $\langle u, v, k+1, R-a, \mathsf{frz}, I\rangle$ of A (B). The reverse process applies to payments from B to A. Noticeably, the action could not be completed if the channel has frozen funds ($\mathsf{frz} \neq 0$)[6].

Freezing. For multi-hop payments, either HTLC or AMHL applies, both requiring a hash preimage to unlock funds. These mechanisms apply to states where $\mathsf{frz} = 0$. To freeze an amount a (*e.g.*, from A), the state transition is

$$\langle u, v, k, R, 0, \_, \mathsf{sig}_A, \mathsf{sig}_B\rangle \mapsto \left\langle u, v, k+1, R-a, a, I, \mathsf{sig}'_A, \mathsf{sig}'_B\right\rangle ,$$

where I is the hash/HOWF value of a uniformly random preimage, $\mathsf{frz} = a$ is frozen. In the symmetric case, the transition is $\langle u, v, k, R, 0, \_, \mathsf{sig}_A, \mathsf{sig}_B\rangle \mapsto \left\langle u, v, k+1, R+a, -a, I, \mathsf{sig}'_A, \mathsf{sig}'_B\right\rangle$.

[6] States consist of eight fixed-length items for a simpler discussion. Hence, channels are modeled to hold at most one HTLC/AMHL at the same time. Yet, by extending the state, enriching dispute-resolving logic, and using threshold signatures (*e.g.*, [55]), we can leverage Turing-complete smart contracts to support more functionalities [37], *e.g.*, freezing multiple sums.

Unfreezing. Without loss of generality, the frozen funds from A to B[7] should be released to B if B provides r such that $H(r) = I$. This corresponds to the state transition syntax

$$\langle u, v, k, R, \mathsf{frz}, I, \mathsf{sig}_A, \mathsf{sig}_B \rangle \mapsto \langle u, v, k+1, R, 0, r, \_, \mathsf{sig}'_B \rangle \,.$$

As will be shown in the dispute resolution phase, even if A is uncooperative, B can redeem the frozen funds using r. Furthermore, if a multi-hop payment fails and B refuses to revoke the state, B cannot claim the funds, and A is allowed to unilaterally withdraw.

Revoked Freezing. If multi-hop payment fails, A and B can collaboratively revoke the latest state by replacing the state $\langle u, v, k+1, R+a, a, I, \mathsf{sig}'_A, \mathsf{sig}'_B \rangle$ with $\langle u, v, k+1, R, 0, \_, \mathsf{sig}''_A, \mathsf{sig}''_B \rangle$. The two signatures on the new state of version number $k+1$ (rather than k) prohibit unnecessary disputes during withdrawal.

A transition from state r_1 to state r_2 is *legal* if it complies with the transition syntax of any action above. We emphasize that a(n) *freezing (unfreezing) state* refers to a channel state triggered by a freezing (unfreezing) action, possibly in the past. It does not imply that the channel is currently frozen or unfrozen.

3.2 On-Chain Protocols

On-chain protocols manage public user interactions and the smart contract Π_{sc}, which maintains the global state of the network and facilitates various on-chain operations. We detail the functionalities of Π_{sc}, integrating necessary definitions of the structure on which it operates and other functions/mappings.

- **Vertices**: A countable set $\mathcal{S} = \{s_0, s_1, \ldots\}$ represents potential nodes in the network topology. An iterator function $\mathsf{nxt}\colon \mathcal{S} \to \mathcal{S}$ enumerates the set, where each node s_{i+1} is obtained by applying nxt to s_i without repetitions.
- **Occupation Mapping**: A function $\mathsf{occupied}\colon \mathcal{S} \to \{0, 1\}$ indicates whether a vertex $u \in \mathcal{S}$ is occupied ($\mathsf{occupied}(u) = 1$) or not (0). Each occupied vertex u is associated with a user public key/address denoted by P_u.
- **Neighbor Function**: A mapping $\mathsf{N}\colon \mathcal{S} \to \mathcal{P}(\mathcal{S})$ assigns to each vertex u a set of neighboring vertices $\mathsf{N}(u)$. Adjacency is a symmetric relation, *i.e.*, for any $u, v \in \mathcal{S}$, if $v \in \mathsf{N}(u)$, then $u \in \mathsf{N}(v)$.
- **Hyperconnections**: A mapping $\mathsf{hyperConn}(u) = \{v \in \mathcal{S}\colon P_u = P_v\}$ returns all vertices occupied by the same user P_u, representing the concept of super-channels—ideal channels with infinite capacity and zero fees between a user's own vertices. These do not require actual storage or maintenance.

Π_{sc} consists of the following subroutines. Appendix D provides a straw-man instance of the prescribed structure for illustration.

Initiation. Π_{sc} is initiated with a tuple $(\mathcal{S}, s_0, \gamma, \mathsf{N}_0, \widetilde{\mathsf{nxt}}, m, Q)$.

[7] For dispute resolution, the direction is traceable from the previous state.

- γ is a parameter defining the network's regularity.
- $\mathsf{N}_0 \colon \mathcal{S} \to \mathcal{P}(\mathcal{S})$ then defines the topology as a (γm)-regular graph.
- $\mathsf{N}(u)$ is initialized with $\mathsf{N}_0(u)$ for each $u \in \mathcal{S}$.
- m is the maximum number of neighbors (channels) per vertex.
- Q is the base capacity assigned to each channel.
- $\mathcal{S}_q \subseteq \mathcal{S}$ is the set of withdrawn vertices, initially $\mathcal{S}_q := \emptyset$.
- The iterator is initialized with $\mathsf{it} := s_0$, the starting vertex. Its iterating function $\mathsf{nxt}(u)$ returns $v \leftarrow \widetilde{\mathsf{nxt}}(u)$ if $v \neq \perp$ (vacancies exist); else: it returns (and removes) a random element of $\mathcal{S}_q$ if $\mathcal{S}_q \neq \emptyset$, or $\perp$ otherwise.

Node join. A user P joins by submitting $(\texttt{join}, P)_P$ and transferring a deposit mQ (to cover the maximum number of channels) to Π_{sc}.

1. Π_{sc} assigns the next unoccupied vertex it to P and updates as follows.
   - marks the vertex as occupied via $\mathsf{occupied}(\mathsf{it}) := 1$
   - associates the vertex with the user's address by $P_{\mathsf{it}} := P$
   - updates $\mathsf{hyperConn}(\cdot)$ accordingly: $\mathsf{hyperConn}(\mathsf{it}) := \{v \in \mathcal{S} \colon P_v = P\}$ and $\mathsf{hyperConn}(v) := \mathsf{hyperConn}(v) \cup \{u\}$ for each $v \in \mathsf{hyperConn}(\mathsf{it})$.
2. Moves to the next vertex: $\mathsf{it} := \mathsf{nxt}(\mathsf{it})$.
3. Confirmation: Sends a confirmation $(\texttt{join-confirm}, P, \mathsf{it}, \mathsf{N}(\mathsf{it}))$ to P, where $\mathsf{N}(\mathsf{it})$ is the set of neighbors for the assigned vertex $\mathsf{N}_0(\mathsf{it})$.

Channel establishment. After joining, P_u can establish additional channels:

1. P submits $(\texttt{addEdge}, P_u, P_v, u, v, \sigma_v)_{P_u}$ to Π_{sc}, where u is a vertex occupied by P and v is the vertex with which P wants to establish a channel.
2. Π_{sc} checks: P_u and P_v occupy u and v, $|\mathsf{N}(u)|$ and $|\mathsf{N}(v)|$ do not exceed the maximum neighbors m, and $(\texttt{addEdge}, P_u, P_v, u, v, \sigma_v)_{P_u}$ is signed by P_u and P_v via σ_v.
3. If all pass, Π_{sc}, updates $\mathsf{N}(u) \leftarrow \mathsf{N}(u) \cup \{v\}$ and $\mathsf{N}(v) \leftarrow \mathsf{N}(v) \cup \{u\}$.

Route query. To facilitate routing and connectivity, Π_{sc} allows users to query the network. Upon receiving $(\texttt{routeQ}, u)$, Π_{sc} returns P_u, the address occupying vertex u if $\mathsf{occupied}(u) = 1$; $\perp$ otherwise. This query functions as a lookup of on-chain data, implemented as a view function and costs no gas.

Channel withdrawal. To withdraw a channel between u and v, P_u submits $(\texttt{withdrawal}, P_u, u, v, \mathsf{state}_p, \mathsf{state})$ to Π_{sc}, where state_p is the second-latest state and state is the latest channel state.

1. Π_{sc} parses the submitted states and verifies their integrity. Note that state_p can be empty when withdrawing from the initial state. Its inclusion ensures the signature of the counterparty is present, as the latest state might be a unilateral unfreezing lacking mutual agreement.
2. Π_{sc} forwards the withdrawal request to v and enters a dispute resolution period lasting Δ time units. During this period, Π_{sc} listens for additional messages and responds accordingly, as both parties may present further information or challenge the withdrawal (see Appendix B).
3. After Δ time units (*i.e.*, after the generation of a certain number of blocks), it removes v from $\mathsf{N}(u)$ and u from $\mathsf{N}(v)$, effectively nullifying the channel balances, and distributes the deposits back to u and v based on the agreed-upon final state state. Finally, if $\mathsf{N}(u) = \emptyset$, then $\mathcal{S}_q$ is updated to $\mathcal{S}_q \cup \{u\}$; similarly, if $\mathsf{N}(v) = \emptyset$, then $\mathcal{S}_q \leftarrow \mathcal{S}_q \cup \{v\}$.

3.3 Payments and Routings

Intra-channel transfers suffice for one-hop payments, whereas non-adjacent nodes require multi-hop payments for off-chain transfers. This involves routing and channel locks. Either HTLC or AMHL locks may be used, although the pseudocode in Appendix C assumes AMHL. For routing, in addition to the canonical schemes, we overview two alternative approaches (detailed in Appendix E).

The first approach finds routes within the basis graph (assuming $\gamma > 0$). Nodes, especially lightweight ones, can compute the route locally using only a few parameters, without having to maintain the entire PCN structure by storing or monitoring numerous channels. This approach reduces both storage and communication overhead. The second approach mirrors existing flow-based routing. If the sender maintains a local copy of the global network structure, it can view it as a flow graph and apply max-flow (MF) algorithms based on augmenting paths. By slightly modifying the algorithm, the sender can identify routes with minimal fees if the network does not alter too rapidly. This optimization, possibly implemented without the implementer's explicit awareness, taps into the *min-cost max-flow* (MCMF) concept. Compared with the first approach, this method utilizes both types of channels and trades higher routing overhead for improved quality of the probed paths.

4 Our Instantiation of the Prescribed Structure

An expander graph is a finite, undirected, and sparse graph with strong connectivity. Here, we assume an m-regular graph $G = (V, E)$, meaning that every vertex in V has m edges. The *adjacency matrix* of G is denoted by A_G. The *kernel metric* for expanders is the largest eigenvalue of A_G that is different from m, denoted λ_G. The *expansion degree*, defined as $h_G = \min_{X \subseteq V, 2|X|<|V|} \frac{|\partial X|}{|X|}$, is (tightly) bounded from below [49] and above [2] by λ_G. Thus, λ_G effectively quantifies the expansion. Also, the diameter D is bounded by $D \leq \log \frac{|V|}{\log(m/\lambda_G)}$ [11]. Smaller λ_G values yield better expansion, tighter upper bounds on the diameter, and improved connectivity.

We use an LPS-type Ramanujan graph for the prescribed channel structure. To construct a Ramanujan graph[8], we follow the method of Morgenstern [40], which extends the work of Lubotzky, Phillips, and Sarnak [32], *i.e.*, *LPS graphs*. We denote the resulting graph by LPS(p, q), where p and q are prime parameters.

[8] Ramanujan graphs are a kind of m-regular expander graphs with $\lambda_G \leq 2\sqrt{m-1}$. Asymptotically, a family of Ramanujan graphs is optimal since the lower bound for λ_G, given by $2\sqrt{m-1}(1-\frac{2}{D})-\frac{2}{D}$ [1], approaches $2\sqrt{m-1}$ as the graph grows. This implies nearly optimized bisection bandwidth. Even better, due to their *discrepancy inequality* [12], Ramanujan graphs have a well-distributed number of edges between any collection of vertices, not just across bisections, further optimizing connectivity.

For LPS graphs, the number of vertices is $\frac{q^3-q}{2}$ if $(p|q) = 1$ and $q^3 - q$ if $(p|q) = -1$. LPS(p, q) is a $(p + 1)$-regular Ramanujan graph if $q > 2\sqrt{p}$. Here, $(p|q)$ denotes the Legendre symbol. Appendix H formally defines LPS graphs.[9]

5 Implementation and Experimental Analysis

5.1 Implementation

We implemented a smart contract to realize $\Pi_{\mathtt{sc}}$, handling LPS-graph generation, vertex allocation, withdrawal, and dispute resolution in just 566 lines of *Solidity*.[10] This highlights the simplicity of our design. The main cost is in graph setup. When deployed on *Ethereum*[11], for $(p, q) = (3, 5)$, generating a 4-regular graph with 120 vertices costs a total of $17,417,151$ `gas`, broken down as follows:

– $8,158,006$ `gas` (46.8%) for contract construction,
– $1,049,921$ `gas` (6.0%) for generating the symmetric group, and
– $8,209,224$ `gas` (47.1%) for generating the LPS graph.

After graph generation, enrolling a vertex requires $703,515$ `gas`.[12] These are smart-contract computations of graphs and assignments. Notably, most costs arise from graph generation. These can be reduced by executing outside the contract and then forwarding the computation results with SNARK proofs, gossip protocols, or a data availability layer.

5.2 Experimental Analysis

We compute the following metrics for all relevant values of γ, ranging from 0 (open-choice PCN structure) to 1 (LPS graph), in increments of $1/(p + 1)$.

Separation Metrics. We study the average and longest distances between two distinct nodes. The latter is also called diameter. These metrics capture the average and worst-case numbers of hops for multi-hop payments.

Minimal Cuts. Centralization leads to network bisection [27] after disabling only a few nodes. Thus, a long bisection width implies decentralization. We measured the number of cut nodes/edges as defined in §1.1 using Tarjan's linear-time algorithm [50] to assess what it takes to bisect a network. Yet, our construction, by design, yields no cut node/edge for all pairs (p, q) and almost all γ. To better analyze connectivity trends as γ varies, we use *the average minimal cuts*, defined as the expected fewest disabled channels disconnecting two nodes. Higher minimal cuts imply better connectivity and decentralization: Networks dominated

[9] Vertices in LPS graphs often correspond to elements of $\mathrm{PGL}(2, \mathbb{F}_q)$ or $\mathrm{PGL}(2, \mathbb{F}_q)$, congruence subgroups of $\mathrm{GL}(2, \mathbb{F}_q)$ / $\mathrm{SL}(2, \mathbb{F}_q)$. Edges correspond to actions by elements in the group or certain generating sets within the group.

[10] `https://github.com/htftsy/PcnTopology/blob/main/contract/PCN.sol`

[11] Raiden (`https://raiden.network`) is one of the PCNs built for Ethereum.

[12] At exchange rates of 2:10 PST, October 10, 2024, graph generation needs 0.152 `ETH` or 362.859 `USD` (3.02 `USD` per vertex). Afterward, enrolling a vertex takes 14.657 `USD`.

by large bouquets of channels (§1.1) rooted in a few nodes exhibit lower minimal cuts, while larger cuts suggest smaller, more distributed bouquets.

Clustering. We also measure transitivity and the clustering coefficient (see §1.1). For a fixed number of channels, lower values of these metrics indicate fewer nodes forming cliques, implying better decentralization and global connectivity. These serve as auxiliary supports for decentralization.[13]

Evaluation Methods. We pick the following Ramanujan graphs. To simulate PCN with $\sim 10^4$ or 10^5 nodes, we set $q = 23$ or 47 and vary p over $\{5, 7, 11, 17, 19\}$ or $\{5, 11, 13, 19, 23\}$. This results in networks with $12,167$ or $103,823$ nodes, forming m-regular graphs with $m \in \{6, 8, 12, 18, 20\}$ or $\{6, 12, 14, 20, 24\}$. Recall that the graph size is $q^3 - q$ if $(p|q) = -1$, and half that if $(p|q) = 1$. We omit the PSL cases like $(p, q) = (13, 23)$ to avoid sudden variations as p grows.

Each $\gamma < 1$ instance involves open-choice channels, but real-world PCN data like LN is not fully applicable for modeling them. First, LN itself is small ($\sim 10^4$ nodes), limiting any analysis with scaling. Second, existing PCNs mostly involve developers and researchers whose behavior differs from larger user-oriented networks. Lastly, most LN nodes have similar forwarding tables, with 37.7% having a degree of 1, contrasting our case of $\gamma = 0$, where each node establishes $p + 1$ channels. Thus, we simulate open-choice channels using random graphs.

We simulate PCNs by Barabási-Albert's algorithm [9], which follows power-law distributions, modeling new nodes linking to existing ones with probabilities proportional to their degrees, simulating the "rich-get-richer" effect.[14] We first generate a complete Ramanujan graph, retain a random γ-fraction of edges, and replace the rest using random sampling above.

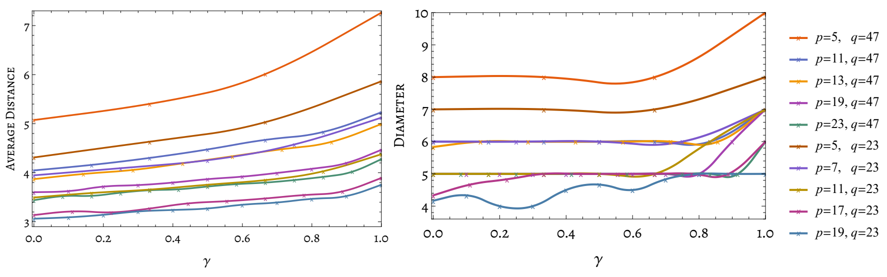
Calculating average and longest distances takes $\mathcal{O}(N^2 \log N)$ time. To manage this for $N = 10^5$, we sample 30 nodes and calculate their shortest paths to all others. The average distance is the mean of these, while the longest is measured as the maximum observed. For the average min-cut, we sample 30 node pairs, convert the graph into a flow network between each pair, and compute the max-flow (min-cut) using the *max-flow min-cut theorem.* Transitivity is measured by sampling 10^5 random pairs of adjacent channels, and clustering coefficients are computed directly. Since our networks lack large bouquets, we can safely enumerate adjacent pairs without hitting $\Theta(N^2)$ complexity.

Finally, we sample 6 graphs for each (p, q), evaluate the metrics independently, and report the averages. Our open-source codes are available.[15]

[13] Clustering metrics alone do not fully capture decentralization, *e.g.*, the highly centralized star-shaped network has zero transitivity and clustering coefficient.

[14] Some methods commonly used to model social networks are unsuitable. The Erdős-Rényi model [19], which uses uniform edge distribution (edges exist with a fixed probability), does not reflect typical PCN behavior. The Watts-Strogatz model [54] simulates "small-world phenomenon" of social networks. However, real-world PCNs resemble *scale-free graphs* [9], where node degrees follow power-law distributions. A node's degree k occurs with a probability proportional to $k^{-\eta}$ for some parameter η.

[15] `https://github.com/htftsy/PcnTopology`



(a) Average distance (b) Diameter

Fig. 1. Evaluations of distance measures

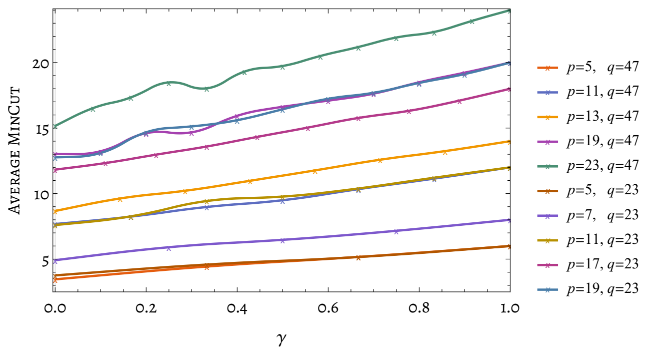
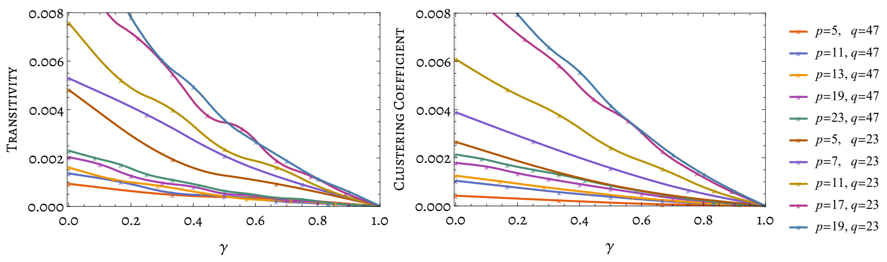


Fig. 2. Evaluation of minimal cuts



(a) Transitivity (b) Clustering coefficient

Fig. 3. Evaluation of clustering measures

Evaluation Results. We assess separation metrics, minimal cuts, and clustering.

– *Separation Metrics.* Fig. 1 shows that both average distance and diameter increase with γ, indicating that the average and worst-case hops increase as more prescribed channels are added. Open-choice channels, which follow power-law distributions, offer slightly better distance metrics but tend to introduce significant centralization. Interestingly, sharp phase transitions in the diameter occur as γ grows. For instance, the curve for $(p, q) = (13, 47)$

holds steady at a diameter of 6 until γ reaches 1, at which point it jumps to 7. Such transitions are common in random graphs (and have broad implications for social networks).

- *Minimal Cuts.* Fig. 2 shows that the average minimal cut increases with γ, indicating improved connectivity and decentralization as more prescribed channels are added. The expander ($\gamma = 1$) yields the optimal minimal cut, where the most efficient partitioning involves cutting $p + 1$ adjacency nodes. Interestingly, the curves for $q \in \{23, 47\}$ almost overlap for the same p, suggesting that network connectivity stays nearly constant as the graph extends. This underscores the high cost of censorship and network bisection.
- *Clustering.* Fig. 3 reveals a consistent decrease in transitivity and clustering coefficient as γ increases, further illustrating the enhanced decentralization.

Appendix G analyzes user incentives through simulations, indicating the system is friendly to average-sized users.

6 Conclusion

We have proposed a systematic methodology for improving connectivity and decentralization in PCNs through the integration of strategically designed network structures. Our approach demonstrates clear advantages in terms of network resilience and user incentivization, as validated through extensive simulations.

This framework offers a practical and scalable solution to the centralization issues in PCNs, making it well-suited for real-world deployment. The use of expander graphs to guide network structure balances user autonomy with enhanced network-wide benefits, significantly improving the robustness and long-term sustainability of PCNs. Our study advances the development of decentralized, resilient PCNs, bridging the gap between theoretical network design and practical implementation. Still, several challenges remain, opening avenues for future exploration in compatibility, security, scalability, and interoperability.

Bitcoin Script Compatibility. Currently, LN remains the most impactful PCN solution in the Bitcoin ecosystem. Nevertheless, our methodology can be emulated and adapted by UTXO-based systems for broader real-world applications.

Other Basis Graphs. The adaptability of PCNs to varying network sizes depends heavily on their underlying graph structure. A desirable solution should
(1) extend the graph infinitely,
(2) refine the topology and ensure fairness, and
(3) mitigate dormant rates (*e.g.*, when a node joins much later than the other).

While instantiating the graph via a hypercube and using a high-dimensional Hilbert curve as an iterator could satisfy conditions (1) and (3), it falls short on condition (2) due to fairness constraints. Conversely, our LPS instantiation fixes the degree at genesis and imposes an upper bound on the total number of nodes. Once this limit is reached, new users must wait for vacancies created by departing nodes. Future research could explore alternative basis graphs that meet all three criteria while also reducing gas costs and improving implementation efficiency.

Other Integrations. The principle of strategic mesh connection is likely to apply to commit chains and payment hubs, shaping the network as symmetric and well-connected hypergraphs with long bisection widths and hence improved topology. Future work could explore hybrid approaches, such as integrating our routing techniques into leader-based layer-2 retail payment systems [42].

Target Censoring Prevention. A predictable order of vertex assignments has security implications. Adversaries might corrupt miners to disrupt transaction orders, enabling them to spawn and position vertices adjacent to a target node. This attack vector allows censorship or blocking of the target. One potential mitigation strategy involves introducing randomness (*e.g.*, [30]) into the iterator to make such attacks more difficult. Further research is needed to assess the effectiveness of this approach and its impact on network performance.

User Behavior. A game-theoretic analysis of user behavior can provide deeper insights into the incentive mechanisms that drive user interactions within the network. We highlight two challenges.

- *Penalty or Reputation.* Non-compliance—whether by attempting to settle with outdated states or ignoring the prescribed channel formation—should trigger a penalty mechanism that deters misbehavior and safeguards honest users from false accusations. Additionally, a decentralized reputation system with dynamic credentials [10] could further promote adherence to the prescribed channel formation while reinforcing fairness.
- *Foreign PCNs Interoperability.* Channels have to be established by the smart contract, which controls the number of open channels, preventing users from spawning more than prescribed. However, protocols can be devised to operate on foreign PCNs while leveraging the same multi-hop logic. This would allow users to open channels in external PCNs while benefiting from the topology of our network. The implications of such interoperability warrant further investigation to understand its effects on network dynamics and incentives.

Acknowledgments

Sherman Chow (corresponding author) is supported in part by the General Research Fund (CUHK 14210621) from the Research Grant Council, Hong Kong; and by Direct Grant (4055238) and the Strategic Impact Enhancement Fund (3135517) from the Chinese University of Hong Kong, Hong Kong.

References

1. N. Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.
2. N. Alon and V. D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38:73–88, 1985.

3. L. Aumayr, M. Maffei, O. Ersoy, A. Erwig, S. Faust, S. Riahi, K. Hostáková, and P. Moreno-Sanchez. Bitcoin-compatible virtual channels. In *SP*, pages 901–918, 2021.
4. L. Aumayr, S. A. K. Thyagarajan, G. Malavolta, P. Moreno-Sanchez, and M. Maffei. Sleepy channels: Bi-directional payment channels without watchtowers. In *CCS*, pages 179–192, 2022.
5. G. Avarikioti, F. Laufenberg, J. Sliwinski, Y. Wang, and R. Wattenhofer. Towards secure and efficient payment channels. arXiv 1811.12740, 2018.
6. G. Avarikioti, R. Scheuner, and R. Wattenhofer. Payment networks as creation games. In *CBT Workshop (co-located with ESORICS)*, pages 195–210, 2019.
7. Z. Avarikioti, E. Kokoris-Kogias, R. Wattenhofer, and D. Zindros. Brick: Asynchronous incentive-compatible payment channels. In *FC-II*, pages 209–230, 2021.
8. Z. Avarikioti, T. Lizurej, T. Michalak, and M. Yeo. Lightning creation games. In *ICDCS*, pages 1–11, 2023.
9. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
10. S. S. M. Chow, C. Egger, R. W. F. Lai, V. Ronge, and I. K. Y. Woo. On sustainable ring-based anonymous systems. In *CSF*, pages 568–583, 2023.
11. F. R. K. Chung. Diameters and eigenvalues. *Journal of the American Mathematical Society*, 2:187–196, 1989.
12. F. R. K. Chung. *Spectral graph theory*. American Mathematical Society, 1997.
13. C. Decker, R. Russell, and O. Osuntokun. eltoo: A simple layer2 protocol for Bitcoin. `https://blockstream.com/eltoo.pdf`, 2018. Last accessed on September 12, 2024.
14. C. Decker and R. Wattenhofer. A fast and scalable payment network with Bitcoin duplex micropayment channels. In *SSS*, pages 3–18, 2015.
15. S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková. Multi-party virtual state channels. In *EUROCRYPT Part I*, pages 625–656, 2019.
16. S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *SP*, pages 106–123, 2019.
17. S. Dziembowski, S. Faust, and K. Hostáková. General state channel networks. In *CCS*, pages 949–966, 2018.
18. C. Egger, P. Moreno-Sanchez, and M. Maffei. Atomic multi-channel updates with constant collateral in Bitcoin-compatible payment-channel networks. In *CCS*, pages 801–815, 2019.
19. P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.
20. O. Ersoy, J. Decouchant, S. P. Kumble, and S. Roos. SyncPCN/PSyncPCN: Payment channel networks without blockchain synchrony. In *AFT*, pages 16–29, 2022.
21. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
22. L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. SoK: Layer-two blockchain protocols. In *FC*, pages 201–226, 2020.
23. M. Hearn. Micro-payment channels implementation now in bitcoinj. `https://bitcointalk.org/?topic=244656.0`, 2013. Last accessed on September 12, 2024.
24. A. Hertig. Bitcoin's lightning network is growing 'increasingly centralized,' researchers find. `https://www.coindesk.com/tech/2020/02/20/bitcoins-lightning-network-is-growing-increasingly-centralized-researchers-find`, 2020. Last accessed on September 12, 2024.
25. R. Khalil and A. Gervais. Revive: Rebalancing off-blockchain payment networks. In *CCS*, pages 439–453, 2017.

26. A. Kiayias and O. S. T. Litos. A composable security treatment of the Lightning network. In *CSF*, pages 334–349, 2020.
27. J. Lin, K. Primicerio, T. Squartini, C. Decker, and C. J. Tessone. Lightning network: a second path towards centralisation of the bitcoin economy. arXiv 2002.02819, 2020.
28. J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. R. Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. In *SOSP*, pages 63–79, 2019.
29. A. Lisi, D. D. F. Maesa, P. Mori, and L. Ricci. Lightnings over rose bouquets: an analysis of the topology of the Bitcoin Lightning Network. In *COMPSAC*, pages 324–331, 2021.
30. J. Liu and M. Manulis. Fast SNARK-based non-interactive distributed verifiable random function with Ethereum compatibility. In *AsiaCCS*, 2025. To appear, preliminary version at ia.cr/2024/968.
31. Z. Lu, R. Han, and J. Yu. General congestion attack on HTLC-based payment channel networks. In *Tokenomics*, pages 2:1–2:15, 2021.
32. A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.
33. G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei. SilentWhispers: Enforcing security and privacy in decentralized credit networks. In *NDSS*, 2017.
34. G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi. Concurrency and privacy with payment-channel networks. In *CCS*, pages 455–471, 2017.
35. G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS*, 2019.
36. P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller. Pisa: Arbitration outsourcing for state channels. In *AFT*, pages 16–30, 2019.
37. W. Meng, J. Wang, X. Wang, J. K. Liu, Z. Yu, J. Li, Y. Zhao, and S. S. M. Chow. Position paper on blockchain technology: Smart contract and applications. In *NSS*, pages 474–483, 2018.
38. A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry. Sprites and state channels: Payment networks that go faster than Lightning. In *FC*, pages 508–526, 2019.
39. A. Mizrahi and A. Zohar. Congestion attacks in payment channel networks. In *FC Part II*, pages 170–188, 2021.
40. M. Morgenstern. Existence and explicit constructions of $q+1$ regular Ramanujan graphs for every prime power q. *J. of Comb. Theory, Series B*, 62:44–62, 1994.
41. C. Ndolo and F. Tschorsch. Payment censorship in the Lightning network despite encrypted communication. In *AFT*, pages 12:1–12:24, 2024.
42. L. K. L. Ng, S. S. M. Chow, D. P. H. Wong, and A. P. Y. Woo. LDSP: Shopping with cryptocurrency privately and quickly under leadership. In *ICDCS*, pages 261–271, 2021.
43. R. Pass and a. shelat. Micropayments for decentralized currencies. In *CCS*, pages 207–218, 2015.
44. J. Poon and T. Dryja. The Bitcoin Lightning network: Scalable off-chain instant payments, version 0.5.9.2. `https://lightning.network/lightning-network-paper.pdf`, 2016. Last accessed on September 12, 2024.
45. r/CryptoCurrency. Research shows lightning network vulnerable to split attacks and congestion attacks. `https://www.reddit.com/r/CryptoCurrency/comments/f7csud/research_shows_lightning_network_vulnerable_to`, 2020. Last accessed on September 12, 2024.

46. S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In *NDSS*, 2018.
47. V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath. Routing cryptocurrency with the Spider network. In *HotNets*, pages 29–35, 2018.
48. S. Tang and S. S. M. Chow. Towards decentralized adaptive control of cryptocurrency liquidity via auction. In *ICDCS*, pages 910–919, 2023.
49. R. M. Tanner. Explicit concentrators from generalized N-gons. *SIAM Journal on Algebraic and Discrete Methods*, 5:287–293, 1984.
50. R. E. Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, 1974.
51. S. Tochner, A. Zohar, and S. Schmid. Route hijacking and DoS in off-chain networks. In *AFT*, pages 228–240, 2020.
52. Trustnodes. Lightning Network DDoS sends 20% of nodes down. `https://www.trustnodes.com/2018/03/21/lightning-network-ddos-sends-20-nodes`, 2018. Last accessed on September 12, 2024.
53. P. Wang, H. Xu, X. Jin, and T. Wang. Flash: Efficient dynamic routing for offchain networks. In *CoNEXT*, pages 370–381, 2019.
54. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.
55. H. W. H. Wong, J. P. K. Ma, and S. S. M. Chow. Secure multiparty computation of threshold signatures made more efficient. In *NDSS*, 2024.
56. P. Zabka, K. Förster, C. Decker, and S. Schmid. Short paper: A centrality analysis of the Lightning network. In *FC*, pages 374–385, 2022.

A Reviews of Payment Channels and Multi-hop Locks

A.1 An Exemplary (Unilateral) Payment Channel

We present a unilateral payment channel construction to illustrate how it works. Suppose A is a customer, and B is a service provider. A frequently makes small payments to B. To avoid prohibitive costs for making them all on-chain, they establish a payment channel using a smart contract and proceed as follows.

1. A deposits Q into the contract address as collateral. A and B locally record the channel state $\mathsf{rec} = (\mathsf{state}_0, \mathsf{state}_1, \ldots, \mathsf{state}_\ell)$ with an initial unitary state $\mathsf{state}_0 = \langle A, B, 0, Q\rangle$. Here, 0 is the initial version number, and Q is the remaining balance available for transfers through the channel.
2. For each payment of amount R', suppose the latest state is $\langle A, B, s, R\rangle$, A sends a message $\left\langle A, B, s+1, R-R', \mathsf{sig}_{s+1}\right\rangle$ to B. B accepts only if the version number increases by one and sig_{s+1} is a valid signature of A on the hash of $\langle A, B, s+1, R-R'\rangle$.
3. To close the channel and settle funds, B signs the latest state $\langle A, B, s, R\rangle$, and then sends both the state and its signature, sig_s, to the contract. Upon recipient and verification, the contract sends $Q-R$ to B and R to A.

A.2 Multi-hop Payment via Hashed Time-Lock Contracts (HTLC)

Suppose that an amount x is transferred from A to D via a sequence of channels, each having at least a balance of x, denoted by $A \xrightarrow[h]{x} B \xrightarrow[h]{x} C \xrightarrow[h]{x} D$. Using HTLC, the payment proceeds in two steps:

1. *Contract setup:* D picks random r and sends $h = H(r)$ to A, who forwards it along the chain to B and C. Each sender then signs a *time-lock contract*, freezing x in each channel. The contract stipulates that the funds are released upon providing r' such that $H(r') = h$. If the payment fails, the contract expires after a predetermined period[16]: Δ' for $C \to D$, $2\Delta'$ for $B \to C$, and $3\Delta'$ for $A \to B$.
2. *Fund release:* From D back to A, each receiver unlocks the frozen funds by providing r to the sender in the preceding channel. If a sender refuses to cooperate, the receiver can claim the funds on-chain using HTLC and r.[17]

A.3 Multi-hop Payment via Atomic Multi-hop Locks (AMHL)

AMHL is an alternative to HTLC with enhanced security, utilizing a homomorphic one-way function $g(\cdot)$. The key idea is that each node in the payment path ensures that subsequent payments are completed before unlocking its own funds, preventing wormhole attacks [35]. AMHL differs from HTLC in several ways: (1) The first sender samples random $y_0, y_1, \ldots, y_{n-1}$ and sends the tuple (Y_i, Y_{i+1}, y_i) to the i^{th} node, where $Y_i = g(k_i)$ and $k_i = \sum_{j=0}^{i-1} y_j$. Each node verifies $Y_{i+1} = Y_i + g(y_i)$. (2) Each adjacent node pair verifies that their received Y_{i+1} values match. (3) The release condition for each lock is Y_i with preimage k_i.

B Dispute Resolution

For A to withdraw, A presents the latest channel state and awaits possible disputes (from B). If B disagrees with the channel state presented by A, B invokes the dispute resolution process to settle the correct state.

The dispute resolution mechanism in Π_{sc} addresses two primary scenarios.

Conflicting States. B presents $\langle u, v, k', R', \mathsf{frz}', I', \mathsf{sig}'_A, \mathsf{sig}'_B \rangle$, a conflicting channel state that challenges the state submitted by A. Π_{sc} executes:

- Verify that $\mathsf{sig}'_A, \mathsf{sig}'_B$ are valid signatures corresponding to the provided state and are not associated with an unfreezing action.
- Parse the withdrawal state submitted by A: $\langle u, v, k, R, \mathsf{frz}, I, \mathsf{sig}_A, \mathsf{sig}_B \rangle$.
- Check if the balances or frozen amounts differ: $(R, \mathsf{frz}, I) \neq (R', \mathsf{frz}', I')$.
- If $k' = k$:

[16] The time-lock is often tied to block height and Δ' bounds network delays.
[17] In the LN, channel states are implemented as specialized UTXOs.

  - If $\mathsf{frz}' \neq 0$ while $\mathsf{frz} = 0$, it indicates an attempt to revert a revoked freezing. The contract aborts the withdrawal.
  - Otherwise, the contract updates the withdrawal state to the conflicting state provided by B.
- If $k' = k - 1$:
  - If $\mathsf{frz}' = 0$ and sig_B is empty in A's state (A is attempting an unfreezing action on a revoked freezing without B's consent), the withdrawal state is then switched to the state provided by B.

We have stipulated that only disputes over the latest two states (as submitted by A) are meaningful because these states include counterpart signatures on two consecutive versions. This ensures that B cannot revert to an earlier version by invoking unavoidable "conflicting states" that revoke freezing. An honest B should have a local version number of $k-1$ following an unfreezing, or k typically.

In the first case (version $k-1$), it prevents A from reverting a revoked state at $k-1$ and extending it to k through unfreezing. In the second case (version k), this prevents A (and B) from submitting a revoked freezing state and prohibits forking from historical states via state renewal.

State Renewal. If A is found to have submitted an obsolete state, the withdrawal state must be updated accordingly. Upon discovery, B provides a newer channel state r_n (along with its previous state r_p) with a higher version number k', which indicates a more recent agreement between the parties.

- Confirm that $k' > k$ and signatures of (r_p, r_n) are valid (if not blank).
- Verify that the transition from r_p to r_n follows the protocol rules.
- If any check fails, Π_{sc} aborts. Otherwise, Π_{sc} updates the withdrawal state to $\langle u, v, k', R', \mathsf{frz}', I', \mathsf{sig}'_A, \mathsf{sig}'_B \rangle$.

The above functionalities address the following disputes in particular. (1) If A tries to "fork" from an obsolete state (unfreezing or revoked freezing), B could first switch to the correct branch using the conflicting state rule and then extend to the latest state via state renewal. (2) If A withdraws with a freezing state while B has the preimage, B may unfreeze the funds via state renewal.

C Pseudocodes

Fig. 4 and Fig. 5 present the pseudocode for the smart contract Π_{sc} and users Π_{u}, respectively, using AMHL as multi-hop locks. For clarity, we assume aborted subroutines after any verification failure. Also, recall that $(\mathsf{message})_P$ denotes a signed message from P, implying that the tuple is received, parsed, and σ_P is verified against $\mathsf{message}$ (abort if it fails). We assume a predefined order of u (the withdrawal issuer) and v (the dispute proposer) if needed. The full code must account for determining this order. Initial signature verification is often omitted, as is fee-handling logic for clarity. Security analysis is provided in Appendix F.

```
$\Pi_{\mathsf{sc}}$ shares $\mathsf{T}$ with all users and the newest block height $t_{\mathsf{now}}$ with the global ledger.

– Initiation. On initiation with $(\mathcal{S}, s_0, \gamma, \mathsf{N}_0, \mathsf{nxt}, m, Q)$:
• $\mathsf{T} := \epsilon$, $\mathsf{it} := s_0$, $\mathsf{occupied}(u) := 0$ for all $u \in \mathcal{S}$, and $\mathcal{S}_q := \emptyset$

– Node Join. On $(\texttt{join}, P)_P$:      (Abort if not receiving transfer $mQ$)
• If $\mathsf{nxt}(\mathsf{it}) \neq \perp$:
      $P_{\mathsf{it}} := P$; $\mathsf{occupied}(\mathsf{it}) := 1$; $\mathsf{it}_p := \mathsf{it}$; $\mathsf{it} := \mathsf{nxt}(\mathsf{it})$; $\mathsf{N}(u) \leftarrow \mathsf{N}_0(u)$
• Else if $\mathcal{S}_q \neq \emptyset$:
      $P_{\mathsf{it}} := P$; $\mathsf{occupied}(\mathsf{it}) := 1$; $\mathsf{it}_p := \mathsf{it}$; $\mathsf{it} \overset{\$}{\leftarrow} \mathcal{S}_q$; $\mathsf{N}(u) \leftarrow \mathsf{N}_0(u)$; $\mathcal{S}_q \leftarrow \mathcal{S}_q \setminus \{\mathsf{it}\}$
• Send the tuple $(\texttt{join-confirm}, P, \mathsf{it}_p, \mathsf{N}(\mathsf{it}_p))$ to $P$

– Channel Establishment. On $(\texttt{addEdge}, P_u, P_v, u, v, \sigma_v)_{P_u}$:
• Abort if $|\mathsf{N}(u)| = m$, $|\mathsf{N}(v)| = m$, or $v \in \mathsf{N}(u)$
• Verify the signature $\sigma_v$, update $\mathsf{N}(u) \leftarrow \mathsf{N}(u) \cup \{v\}$ and $\mathsf{N}(v) \leftarrow \mathsf{N}(v) \cup \{u\}$

– Route Query. On $(\texttt{routeQ}, u)$:
• Return $(u, P_u)$ if $\mathsf{occupied}(u) = 1$; otherwise, return $\perp$

– Withdrawal Issuing. On $(\texttt{withdrawal}, P_u, u, v, \mathsf{state}_p, \mathsf{state})_{P_u}$:
• Abort if $v \notin \mathsf{N}(u)$ or $(u, v, t, r) \in \mathsf{T}$ for any $t, r$
• If $\mathsf{state}$ only contains one valid signature (unfreezing):
      verify that $\mathsf{state}_p$ contains two valid signatures;
      verify that the transition from $\mathsf{state}_p$ to $\mathsf{state}$ is legal
• If $\mathsf{state}$ only contains no signature (initial): verify that $\mathsf{state}$ is initial
• Add $(u, v, t_{\mathsf{now}} + \Delta, \mathsf{state})$ to $\mathsf{T}$

– Withdrawal Completion. On $(\texttt{withdrawal-complete}, P_u, u, v)$:
• Find $(u, v, t, r)$ from $\mathsf{T}$, abort if not found or $t_{\mathsf{now}} \leq t$
• Obtain the latest balance $e$ of $P_u$ from $r$; Send money $e$ to $P_u$ and $2Q - e$ to $P_v$
• Remove $v$ from $\mathsf{N}(u)$, $u$ from $\mathsf{N}(v)$, and $(u, v, t, r)$ from $\mathsf{T}$
• If $\mathsf{N}(u) = \emptyset$: update $\mathcal{S}_q \leftarrow \mathcal{S}_q \cup \{u\}$; if $\mathsf{N}(v) = \emptyset$: update $\mathcal{S}_q \leftarrow \mathcal{S}_q \cup \{v\}$

– Dispute Resolution. On $(\texttt{dispute}, P_v, u, v, \mathsf{type}, \mathsf{witness})$:
• Find $(u, v, t, r) \in \mathsf{T}$, abort if not found or $t_{\mathsf{now}} > t$
• Parse $r$ to $\langle u, v, k_r, R_r, \mathsf{frz}_r, I_r, \mathsf{sig}'_A, \mathsf{sig}'_B \rangle$
• if $\mathsf{type} = \texttt{conflict-record}$:
 - Parse $\mathsf{witness}$ to $\langle u, v, k, R, \mathsf{frz}, I, \mathsf{sig}_A, \mathsf{sig}_B \rangle$
 - Abort if either $\mathsf{sig}_A$ or $\mathsf{sig}_B$ is blank ($\_$)
 - If $k = k_r$, $\mathsf{sig}_A, \mathsf{sig}_B$ verified, $r \neq \mathsf{witness}$:
      if $\mathsf{frz} = 0$ or $\mathsf{frz}_r \neq 0$: modify $(u, v, t, r)$ to $(u, v, t, \mathsf{witness})$ in $\mathsf{T}$
 - If $k = k_r - 1$, $\mathsf{sig}_A, \mathsf{sig}_B$ verified:
      if $\mathsf{frz} = 0$ and $\mathsf{sig}'_B$ is empty: modify $(u, v, t, r)$ to $(u, v, t, \mathsf{witness})$ in $\mathsf{T}$
• if $\mathsf{type} = \texttt{state-renewal}$:
 - Parse $\mathsf{witness}$ to $(r_p, r_n)$ and parse $r_n$ to $\langle u, v, k, R, \mathsf{frz}, I, \mathsf{sig}_A, \mathsf{sig}_B \rangle$
 - Verify $k > k_r$, $(r_p, r_n)$ are valid, and that the transition from $r_p$ to $r_n$ is legal
 - Modify $(u, v, t, r)$ to $(u, v, t, \langle u, v, k, R, \mathsf{frz}, I, \mathsf{sig}_A, \mathsf{sig}_B \rangle)$ in $\mathsf{T}$
```

Fig. 4. Smart contract Π_{sc} (withdrawal issuing/completion are symmetric for P_v)

Suppose that the user P has access to the tuple $(\mathcal{S}, s_0, \gamma, \mathsf{N}_0, \mathsf{nxt}, m, Q)$.

– **Node Join.** To enroll a vertex:
• Assemble, sign, and send the tuple $(\mathtt{join}, P)_P$ to Π_{sc}, with the transfer of mQ
• After reciprocated with $(\mathtt{join\text{-}confirm}, P, u, \mathsf{N}(u))$, store u and $\mathsf{N}(u)$
• Initialize a record rec_v of channel states for each $v \in \mathsf{N}(u)$ with the initial state $\langle u, v, 0, Q, 0, \_, \_, \_\rangle$

– **Channel Establishment.** To establish an open-choice channel with P_v (v):
• Obtain σ_v from P_v, and send to Π_{sc} the signed tuple $(\mathtt{addEdge}, P, P_v, u, v, \sigma_v)_P$
• Initialize a record rec_v for v, containing the initial state $\langle u, v, 0, Q, 0, \_, \_, \_\rangle$

– **Intra-Channel Payment.** To update the balance from R to R' with P_v:
• Sign $\langle u, v, k, R', 0, \_\rangle$ as σ_u
• Retrieve and verify the signature σ_v from P_v (*i.e.*, the balance of P_v is now $2Q - R'$)
• Set $r := \langle u, v, k, R', 0, \_, \sigma_u, \sigma_v\rangle$, append r to rec_v

– **Multi-hop Payment.**
For a multi-hop payment to P_v (v) of amount a, repeat the following until $a = 0$:
• Probe a path $(u_0 = u, u_1, \ldots, u_\ell = v)$
• For each $i = 1, 2, \ldots, \ell - 1$, send $(\mathtt{routeQ}, u_i)$ to Π_{sc} and fetch P_{u_i}
• Determine the transfer amount a' after querying the nodes, let $a' := \mathsf{min}\{a', a\}$
• Sample $(y_0, y_1, \ldots, y_{\ell-1})$, let $k_i := \sum_{j=0}^{i-1} y_j$ and $Y_i := g(k_i)$ for each $i \in [\ell]$
• Send the triple $\langle Y_i, Y_{i+1}, y_i\rangle$ to P_{u_i} for each $i \in [\ell - 1]$, and send Y_ℓ to v
• Freeze the channel with P_{u_1}, *i.e.*, append $\langle u, v, k, R - a', a', Y_1, \sigma_u, \sigma_{u_1}\rangle$ to rec_{u_1}
• Send k_ℓ to P_v, pend until receiving k_1 from P_{u_1}
• Receive the unfrozen state $\langle u, v, k + 1, R - a', 0, k_1, \_, \sigma'_{u_1}\rangle$, add it to rec_{u_1}
• Update $a := a - a'$
(procedures for $u_1, u_2, \ldots, u_{\ell-1}$ and v are omitted)

– **Channel Withdrawal.** To withdraw the channel between P_u (u):
• Retrieve $\mathsf{state}_p, \mathsf{state}$ as the latest two items of rec_v
• Sign and send $(\mathtt{withdrawal}, P, u, v, \mathsf{state}_p, \mathsf{state})_P$ to Π_{sc}
• Pend for Δ and send to Π_{sc} the completion tuple $(\mathtt{withdrawal\text{-}complete}, P, u, v)$
• Update $\mathsf{N}(u) \leftarrow \mathsf{N}(u) \setminus \{v\}$

– **Dispute Issuing.** On withdrawal with state issued by P_v $(v \in \mathsf{N}(u))$:
• Let $k := |\mathsf{rec}_v|$ and $r := \mathsf{rec}_v[k-1]$ (indexed from 0)
• Revoke if $r = \mathsf{state}$
• Parse state to $\langle v, u, k', R', \mathsf{frz}', I, \sigma_v, \sigma_u\rangle$
• Parse r to $\langle v, u, k, R, \mathsf{frz}, I, \sigma_v, \sigma_u\rangle$ (equivalent to $\langle u, v, k, 2Q - R, -\mathsf{frz}, I, \sigma_u, \sigma_v\rangle$)
• If $\mathsf{rec}_v[k'] \neq \mathsf{state}$, send to Π_{sc} $(\mathtt{dispute}, P, v, u, \mathtt{conflict\text{-}record}, \mathsf{rec}_v[k'])$
• If $k' < k$, send to Π_{sc} $(\mathtt{dispute}, P, v, u, \mathtt{state\text{-}renewal}, (\mathsf{rec}_v[k-1], \mathsf{rec}_v[k]))$
• Update $\mathsf{N}(u) \leftarrow \mathsf{N}(u) \setminus \{v\}$

Fig. 5. User protocol Π_{u}

D A Straw-man Instance of the Basis Graph

We briefly discuss an exemplary basis graph based on full K-nary trees.

- *Network structure.* The set of vertices is denoted by $\mathcal{S} = [K]^*$, where each element is a string over the alphabet $[K]$, and the empty string ϵ represents the root. For each vertex $s \in \mathcal{S}$, $s||i$ denotes its i^{th} child. Each vertex s (except the root ϵ) has $m := K + 1$ neighbors $\mathsf{N}(s)$, consisting of its K children (for $\mathsf{N}(\epsilon)$) and its parent (for $\mathsf{N}(s)$).
- *Iterator.* The iterator it is initially set to ϵ. The function $\mathsf{nxt}(s)$ is defined for any $s \in [K]^*$ as follows:
  - If every element of s is K, $\mathsf{nxt}(s) := 1||1||\cdots||1$, a string of length $|s|+1$.
  - Otherwise, let k be the smallest index such that $s[k] < K$ and $s[j] = K$ for all $j > k$. Then,
  $$\mathsf{nxt}(s) := s[1]||s[2]||\cdots||(s[k]+1)||1||\cdots||1,$$
  where the trailing 1's extend the string to a length of $|s|$.
- *Routing function:* To find a shortest path from s_1 to s_2, we compute their lowest common ancestor $\hat{s}$. The path traverses from s_1 up to $\hat{s}$ and then down to s_2 along the tree.

Observably, this structure exhibits suboptimal robustness against separating attacks and is intended for demonstrative purposes only.

E Routing for Multi-hop Payments

E.1 Backgrounds on Network Flows

Network Flows. Network flow problems are fundamental in optimization and can be efficiently solved by converting optimization problems to graph-theoretic algorithms. These problems are modeled using a directed graph (V, E) with two special vertices: the *source* s and the *sink* t. Each edge (u, v) has a positive capacity $E(u, v)$, representing the maximum permissible flow from u to v.

A *maxflow* (MF) algorithm computes the greatest possible amount of flow $\mathsf{maxflow}_G(s, t)$ from the source s to the sink t. The MF problem can be formulated as a linear program, but it is often more intuitive to view capacities as the flow limits of pipes, aiming to maximize throughput from s to t. Importantly, each unit of flow contributes only once to the total flow, regardless of how many edges it traverses. For example, a flow of m' passing through k edges contributes m' (not km') to the total flow.

We denote the flow of edge (u, v) by $\mathsf{F}(u, v) \leq E(u, v)$ and refer to F as the *flow network.* The *residual network* is defined as $\widetilde{E} := E - \mathsf{F}$, capturing the remaining capacities after the current flow is accounted for.

In this paper, we consider a maxflow protocol that takes a directed graph (V, E) as input, where $E \colon V^2 \to \mathbb{N} \cup \{\perp\}$, $\perp$ indicating the absence of an edge. This protocol returns the maximum flow amount a and the residual network (V, R), where $R \colon V^2 \to \mathbb{N} \cup \{\perp\}$, after augmenting all possible flows. We denote this process succinctly as $\mathsf{MF}(V, E) = (a, R)$.

Minimum Cost Maximum Flow. The *minimum cost maximum flow* (MCMF) problem extends the MF problem by assigning a cost $C(u, v)$ to each edge (u, v). The objective is twofold: (1) to find a flow that achieves the maximum possible throughput from s to t, and (2) among all such flows, select one with the minimal total cost, calculated as $\sum_{(u,v)\in V^2}^{E(u,v)\neq\perp} \mathsf{F}(u, v) \cdot C(u, v)$. We denote the MCMF protocol by $\mathsf{MCMF}(V, E, C) = (a, c, R)$, where c is the minimal total cost.

Augmenting-based MCMF. Ford and Fulkerson [21] initiated the study of a well-known class of algorithms relying on *augmenting paths* for solving maximum flow and minimum cost maximum flow problems. Although not the most efficient in theory, their method incorporates a mechanism to reverse flow augmentations, introducing a *reverse edge* (initially set to 0 capacity) for every edge.[18]

In our context, an augmentation in the graph corresponds to a multi-hop payment, while sending funds in the reverse direction updates the graph via reverse edge augmentations. The algorithm iteratively probes paths from the source to the sink and incrementally augments the flow along them. Although some algorithms eventually yield the same maximum flow, they assume that all paths can be determined and probed simultaneously—a condition that does not hold in dynamic environments, especially for large payments requiring many paths where individual probes may fail due to changing channel conditions.

Although including reverse edges might suggest cycles or deadlocks, the algorithm guarantees that the total flow from s to t increases by at least one unit with each augmentation without exceeding the maximum flow capacity. Consequently, termination is ensured. The key difference between augmenting-based MF and MCMF algorithms lies in path selection: MF algorithms augment flow along arbitrary paths, whereas MCMF algorithms select the shortest paths in terms of cost. Both approaches ultimately achieve the maximum flow.

E.2 Routing

Two alternative approaches exist for probing multi-hop payments: *static probing* within the prescribed network and *MCMF-based probing*, like MF-based ones in related works. After either, the payment is finalized using the same procedure.

Static Probing. For $\gamma > 0$, the payer in a multi-hop transfer can statically determine paths in the basis graph, freeing light nodes from storing the entire network structure at the cost of path optimality. The probing method is not further detailed here.

MCMF-based Probing. Static probing misses open-choice channels, yielding sub-optimal paths. One can use MCMF-based routing, which runs an augmenting

[18] Reverse edges enable the algorithm to adjust or "undo" previous flow augmentations when a more favorable path is discovered. When flow on an edge is increased, the capacity of its reverse edge increases by the same amount, allowing the reversal.

maxflow algorithm during probing, emulating MCMF by repeatedly selecting the shortest path, probing along it, and augmenting the local network upon success.

In this approach, we abstract the network as $G\colon \mathcal{S}^2 \to \mathbb{N} \cup \{\infty, \perp\}$, where $\mathcal{S}$ is the set of nodes. For any $u, v \in \mathcal{S}$, $G(u,v)$ is the remaining capacity of the channel from u to v. If there is no channel between them, $G(u,v) = \perp$. In particular, for nodes u, v where $v \in \mathsf{hyperConn}(u)$, we set $G(u,v) = \infty$ (as a super-channel). Importantly, the values $G(u,v)$ are locally held by owners of u and v and are inaccessible before probing.

Let $F\colon \mathcal{S}^2 \to \mathbb{N} \cup \{\perp\}$ represent the flow function, which initially maps all vertex pairs to 0. Define the cost function $C\colon \mathcal{S}^2 \to \mathbb{N} \cup \{\perp\}$ as follows:

- If there is a channel from u to v, set $C(u,v) = 1$.
- For each reverse edge from v to u introduced during flow augmentation, set $C(v,u) = -1$.
- If no edge exists between u and v, set $C(u,v) = \perp$.

Suppose u attempts to transfer an amount W to v. We initialize the residual graph $\widetilde{G}$ as G. The protocol proceeds as follows:

1. *Path finding:* Find the shortest path P from u to v in the residual graph $\widetilde{G}$, where the path length is defined as the sum of the costs $C(x,y) \in \{1, -1\}$ for every edge (x,y) in P.
2. *Capacity query:* For each edge (x,y) in P, query the corresponding nodes to obtain the current residual capacity $\widetilde{G}(x,y)$. If any edge satisfies $\widetilde{G}(x,y) = 0$ or if the query fails, mark the connection as disconnected, remove that edge from $\widetilde{G}$, and return to Step 1.
3. *Flow augmentation:* Let the payment value
$$m := \min\left(\min_{(x,y)\in P} \widetilde{G}(x,y),\ W\right)$$
be the minimum residual capacities along the path P and the remaining amount W. Proceed with HTLC/AMHL for the path as in Appendix A.
4. *Update residual graph and flow function:*
   - For each edge (x,y) in P, update the residual capacities as follows:
$$\begin{aligned}\widetilde{G}(x,y) &:= \widetilde{G}(x,y) - m,\\ \widetilde{G}(y,x) &:= \widetilde{G}(y,x) + m.\end{aligned}$$
   - Also, update the flow function by setting:
$$\begin{aligned}F(x,y) &:= F(x,y) + m \quad \text{if } C(x,y) = 1,\\ F(y,x) &:= F(y,x) - m \quad \text{if } C(x,y) = -1.\end{aligned}$$
5. *Update remaining amount:* Set $W := W - m$. If $W > 0$, return to Step 1.

This algorithm, emulating the MCMF algorithm, halts and incurs the least fees if the network is static. In dynamic networks, the fee is not necessarily minimized but optimized under the given conditions.

F Security Analysis

We reason about safety and liveness from three observations.

(1) *An honest node has a unique local chain of channel states.* Both parties in a channel have to sign off on payments, freezing operations, and revocation of freezing. Consequently, neither party can perform these actions covertly, ensuring that all updates extend the local state consistently. For an honest sender, after a freezing operation, the channel is either unfrozen upon receipt of a valid update from the partner or revoked through mutual agreement. Similarly, an honest receiver unfreezes the channel upon receiving the required preimage (and forwards the updated state to the sender) or initiates a revocation. These guarantee a consistent local view of the channel states.

(2) *An honest node can withdraw with its newest local state, except in the case of a final unfreezing.* If the newest local state is the initial state, the node would always withdraw because extending it via disputes requires both signatures. If the latest state (say, of version i) is payment or unfreezing, the partner may only issue disputes by presenting a forked state of version i, as

(a) forking by an unfreezing (by releasing a preimage that it previously claimed not to have received) if the $(i-1)^{\text{th}}$ state is a freezing state;

$$\begin{tikzcd}
\mathsf{state}_{i-2} \arrow[r, "\text{freezing}"] & \mathsf{state}_{i-1} \arrow[r, "\text{(impossible)}"] \arrow[dr, squiggly, "\text{unfreezing}"'] & \mathsf{state}_{i} \\
 & & \mathsf{state}_{i}^{*}
\end{tikzcd}$$

This would not happen due to the following observation. In the honest sequence, freezing is always followed by unfreezing, since a freezing state is either revoked (replaced in the record) or unfrozen, unless it is the newest state that we have precluded.

(b) or forking by a revoked freezing like below.

$$\begin{tikzcd}
\mathsf{state}_{i-1} \arrow[r, "\text{revoke}", "\text{freezing}"'] \arrow[dr, squiggly, "\text{freezing}"'] & \mathsf{state}_{i} \\
 & \mathsf{state}_{i}^{*}
\end{tikzcd}$$

This case is bypassed by the conflict-state rule for "$k' = k$."

Finally, if the newest local state is freezing, there are two other cases besides the two above attacks we preclude.

(c) The partner may dispute by releasing the latest preimage as below.

$$\begin{tikzcd}
\mathsf{state}_{i-1} \arrow[r, "\text{freezing}"] & \mathsf{state}_{i} \arrow[r, squiggly, "\text{unfreezing}"] & \mathsf{state}_{i+1}^{*}
\end{tikzcd}$$

This would not cause harm. As the partner reveals the preimage $k_{\ell+1}$, the honest node would obtain $k_\ell := k_{\ell+1} - y_\ell$ (the preimage of Y_ℓ) and reclaim the lost funds from the previous channel. However, the partner may cheat that the multi-hop payment has failed, have the honest node revoke the channel with Y_ℓ, and then reveal $k_{\ell+1}$. This is precluded by the strategy of revoking a previous channel only after revoking the next.

(d) Also, the partner may issue a conflict-state dispute by revoked freezing, as shown below.

$$\begin{tikzcd}
\mathsf{state}_{i-1} \arrow[r, "\text{freezing}"] \arrow[dr, squiggly, "\text{revoke}", "\text{freezing}"'] & \mathsf{state}_i \\
& \mathsf{state}_i^*
\end{tikzcd}$$

Since revocation requires both signatures, this contradicts the honesty of the withdrawal issuer.

(3) *If the partner withdraws the channel, an honest node loses no funds.* Let i be the version number of the newest (honest) state. The possible attempts that the partner distorts from the honest sequence are as follows.

(a) It may withdraw using an earlier state, a prefix of the honest record. This case is addressed by state renewal.

(b) It may fork from a historical state (with version number $j < i$) and extend the sequence length to $j + 1$ by a revoked freezing (possibly followed by an unfreezing). This case is tackled by one conflicting state with one state renewal depicted below.

$$\begin{tikzcd}
\mathsf{state}_j \arrow[r, "\text{revoke}", "\text{freezing}"'] \arrow[dr, squiggly, "\text{freezing}"'] & \mathsf{state}_{j+1} \arrow[r, "\text{payment}", "\text{/freezing}"'] \arrow[rr, dashed, bend left, "\texttt{state}", "\texttt{renewal}"'] & \cdots \arrow[r] & \mathsf{state}_i \\
& \mathsf{state}^*_{j+1} \arrow[u, dashed, "\texttt{conflict state}"] \arrow[r, dotted, "\text{unfreeze}"] & \mathsf{state}^*_{j+2} \arrow[u, dashed, "\texttt{conflict state}"] &
\end{tikzcd}$$

(c) It may fork from a historical freezing state (of version number $j < i$) and extend the sequence length to $j + 1$ by unfreezing, as illustrated in the chart below.

$$\begin{tikzcd}
\mathsf{state}_{j-1} \arrow[r, "\text{freezing}"] & \mathsf{state}_j \arrow[r, "\text{(impossible)}"] \arrow[dr, squiggly, "\text{unfreezing}"'] & \cdots \arrow[r] & \mathsf{state}_i \\
& & \mathsf{state}^*_{j+1} &
\end{tikzcd}$$

As discussed in (2), freezing is always followed by unfreezing in the honest sequence. Thus, it is impossible as a fork.

(d) It may cheat by proposing a newer version. Since only unfreezing requires a unilateral signature, a newer version is only possible when extending a revoked freezing with one unfreezing. This possibility is precluded by the conflict-state rule, which mandates that "$k' = k - 1$," as below.

$$\begin{tikzcd}
\mathsf{state}_{i-1} \arrow[r, "\text{revoke}", "\text{freezing}"'] \arrow[dr, squiggly, "\text{freezing}"'] & \mathsf{state}_i & \\
& \mathsf{state}^*_i \arrow[r, dotted, "\text{unfreeze}"] & \mathsf{state}^*_{i+1} \arrow[ul, dashed, "\texttt{conflict state}"']
\end{tikzcd}$$

(4) *Safety.* As implied by (2) and (3), an honest node would not lose funds except in the case of transfers.

(5) *Liveness.* With AMHL and freezing revocation, honest nodes will not stall for a failed multi-hop payment. Even with an inactive or malicious partner, discussion (2) shows it will withdraw without losing funds. The underlying consensus and the P2P network ensure the liveness of interacting Π_{sc}. Finally, in the case of an inactive partner, one could issue a channel withdrawal, whose dispute-resolving terminates in a fixed number of block generations.

G Incentive for Participation

Experimental evidence reveals that average-sized users are incentivized to participate. We employ an LPS graph with parameters $(p, q) = (3, 5)$, comprising 120 nodes, average degree 4, and $\gamma = 0.25$. Each node connects to 1 LPS channel and 3 open-choice channels. For comparison, we simulate a fully open-choice random graph with the same total nodes and channels (using the technique outlined in §5.2). over 10^5 multi-hop payments, assuming a cost of 1 unit per channel per payment (earned by channel owners), the total fees amount to $300,739$ for the open-choice network and $332,050$ for ours.

The results show that the node with the most channels ($\beta = 0$) earns $47,659$ in the open-choice network and $38,625$ in ours. Fig. 6 shows more results for users with the fewest connections among top-β users, where β ranges from 0 to 0.5 in increments of 0.02. The orange curves show revenue in the open-choice network fitted via a Bézier curve. The blue ones show the revenue when a user occupies vertices randomly, with similar or slightly fewer channels, but adjusted in our PCN by multiplying by a factor of $\frac{300,739}{332,050}$ to align global costs and preclude the bias of higher revenues driven by higher total fees for fairness.

Notably, the top 5% of highly connected nodes earn more fees in the open-choice PCN. In contrast, users with moderately fewer connections ($0.16 \leq \beta \leq 0.4$) earn more in our PCN, showing that our design favors average-sized users.

H Constructing LPS graphs from PSL and PGL

For completeness, we give the formal definition of LPS graphs (which are *Cayley graphs*) for readers familiar with spectral and algebraic graph theory.

Definition 1. *The Cayley graph of a group $\mathcal{G}$ and a symmetric subgroup S of $\mathcal{G}$ (or generating set S) is one with vertices $V = \mathcal{G}$ and edges $E = \{\{u, v\} : v \in uS\}$.*

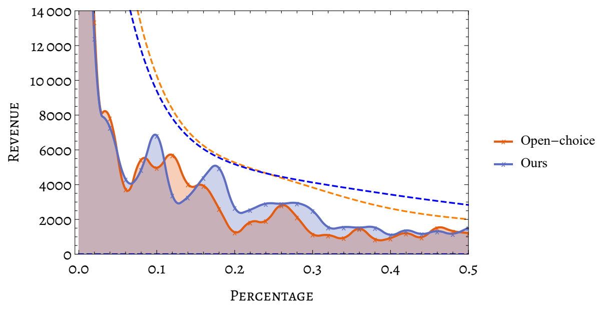


Fig. 6. Revenues of top 50% users in the open-choice PCN and ours ("hybrid")

Definition 2. *For any distinct odd primes* p, q*, an LPS graph* $\mathrm{LPS}(p, q)$ *is the Cayley graph of graph* $\mathcal{G}$ *with generating set* S*, where* $\mathcal{G} = \mathrm{PSL}(2, \mathbb{F}_q)$ *if* $(p|q) = 1$ *and* $\mathcal{G} = \mathrm{PGL}(2, \mathbb{F}_q)$ *otherwise. Here,* $\mathrm{PGL}(2, \mathbb{F}_q)$ *(*$\mathrm{PSL}(2, \mathbb{F}_q)$*) is a quotient of* $\mathrm{GL}(2, \mathbb{F}_q)$ *(*$\mathrm{SL}(2, \mathbb{F}_q)$*) by the group* Z_q *of its scales. Representing their elements by cosets of* 2×2 *matrices, the generating set* S *consists of all elements*

$$\begin{bmatrix} \alpha_0 + \alpha_1 x + \alpha_3 y & -\alpha_1 y + \alpha_2 + \alpha_3 x \\ -\alpha_1 y - \alpha_2 + \alpha_3 x & \alpha_0 - \alpha_1 x - \alpha_3 y \end{bmatrix}_{\sim_q}$$

for $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ *that (1)* $\alpha_0^2 + \alpha_1^2 + \alpha_2^2 + \alpha_3^2 = p$*; (2)* $\alpha_0 > 0$ *is odd, if* $p \equiv 1 \bmod 4$*; and (3)* $\alpha_0 > 0$ *is even, or* $\alpha_0 = 0$ *and* $\alpha_1 > 0$*, if* $p \equiv 3 \bmod 4$*. Here,* x *and* y *are solutions for* $x^2 + y^2 + 1 = 0 \bmod q$*, with* $[M]_{\sim_q}$ *representing the coset of matrices where each* $A \in [M]_{\sim_q}$ *satisfies* $A = xM$ *for a non-zero* $x \in \mathbb{F}_q$*.*