Generalized Payment Channel Topologies via Dual-Track State Machines and Reference-Based UTXOs

Arthur Zhang
Tondi Foundation

Neo Maxwell
Tondi Foundation

December 13, 2025

Abstract

Payment Channel Networks (PCN) constitute a Layer 2 scaling solution for blockchain systems, whose core principle is to complete multiple state updates off-chain while settling only the final state on-chain, thereby improving system throughput.

Background and Problem Definition: Existing PCN schemes (such as the Lightning Network) exhibit two categories of structural limitations: (1) the expressiveness of the Script Layer is constrained, making it difficult to natively support complex state transition semantics; (2) linear topology structures lead to low capital utilization and excessive state management complexity. While the original Eltoo protocol proposed a state replacement mechanism to supersede the penalty mode, its implementation depends on the not-yet-activated `SIGHASH_ANYPREVOUT` soft fork (BIP-118) and presents security concerns such as replay attacks.

Technical Contributions: This paper proposes a general-purpose scaling architecture based on native Eltoo semantics. The main contributions include:

1. **Dual-Track UTXO Model:** Decomposing channel state into a static fund anchor (Fund UTXO) and a dynamic state pointer (State UTXO) along two orthogonal dimensions, achieving separation of concerns between value transfer and state transitions;
2. **Reference-Based UTXO Primitive:** Defining a read-only reference operator $\mathrm{Ref} : \mathcal{U} \to \mathcal{U}^{readonly}$, enabling state update transactions to access fund anchor metadata without consuming that UTXO;
3. **Transaction Type Enumeration System:** Embedding algebraic data types at the consensus layer to achieve $O(1)$ complexity for transaction classification and verification;
4. **Recursive Channel Factories and Atomic Reconfiguration:** Formally defining channel splitting (Splice-Fork) and merging (Splice-Merge) operations, proving that any complex topology can achieve isomorphic transformation through a single atomic transaction.

Theoretical Results: This paper proves the existence of a bijective mapping between UTXO sets and channel state sets (Theorem 8), thereby eliminating dependency on off-chain registries. Under DAG-structured consensus protocols, state verification complexity is $O(1)$, with settlement latency reaching sub-second levels.

Keywords: Payment Channel Networks, State Channels, Eltoo Protocol, UTXO Model, Finite State Machine, Formal Verification, Layer 2 Scaling

Contents

1 Introduction and Motivation **7**
1.1 Problem Background . . . 7
1.2 Design Principles . . . 8
1.3 Trust Model Analysis . . . 8

2 Related Work and Technical Background **9**
2.1 Preliminaries . . . 9
2.1.1 Cryptographic Foundations . . . 9
2.1.2 Timelock Mechanisms . . . 10
2.1.3 Directed Acyclic Graph Consensus . . . 10
2.2 Protocol Evolution: From Penalty Mechanism to State Replacement . . . 11
2.2.1 Lightning Network's Penalty Mechanism and Limitations . . . 11
2.2.2 Eltoo Protocol and State Replacement Mechanism . . . 11
2.2.3 Engineering Compromise of BIP-118 . . . 12
2.3 Structural Defect Analysis of Existing Architectures . . . 12
2.4 This Paper's Solution: UTXO-Native Semantics . . . 12
2.5 DAG Consensus and Protocol Compatibility . . . 13
2.6 BIP-118 Security Boundary Analysis . . . 14
2.7 Economic Efficiency Boundary of Payment Systems . . . 14

3 Research Contributions **15**
3.1 Main Contributions . . . 15
3.2 Information-Theoretic Analysis of State Determinism . . . 15
3.2.1 Verification Causality Graph Comparison . . . 16
3.3 Architectural Advantages . . . 17
3.4 Comparison with Existing Solutions . . . 17
3.5 Theoretical Significance . . . 18

4 Theoretical Framework: Dual-Track State Machines **19**
4.1 Consensus-Layer Embedded Verification Mechanism . . . 19
4.1.1 Transaction Type Enumeration and Pattern Matching . . . 19
4.1.2 State Monotonicity Theorem and Consensus Implementation . . . 19
4.2 Finite State Machine Formalization . . . 20
4.3 UTXO Materialization Layer . . . 20
4.3.1 State-Fund Coupling Invariant . . . 21
4.4 State Transition Rules . . . 22
4.5 Formal Safety Properties . . . 22
4.6 Transaction Semantics Mapping . . . 23
4.7 Evolution of Conditional Payment Primitives: From HTLC to PTLC . . . 23
4.7.1 Technical Principle Comparison . . . 23
4.7.2 Core Properties Comparison . . . 24
4.7.3 Formal Security Analysis . . . 24

5 Topological Primitives for Complex Structures **26**
5.1 Recursive Channel Factories . . . 26
5.1.1 Key Topological Invariants . . . 26
5.1.2 Fractal Topology and Self-Similarity . . . 26
5.2 Dynamic Mesh Reconfiguration . . . 27
5.3 Atomic Rebalancing Operator and Value Conservation . . . 27
5.4 Liquidity Dynamics in Star Topologies . . . 28

5.4.1 Liquidity Utilization Definition . . . 28

6 Safety Analysis **30**
6.1 Isolation Theorem . . . 30
6.2 State Monotonicity and Anti-Replay . . . 30
6.3 Anti-DoS Equilibrium under STPC Strategy . . . 30
6.3.1 Mempool Entropy Bound . . . 31
6.4 PTLC Atomicity and Deadlock Freedom . . . 31
6.4.1 PTLC Atomicity Theorem . . . 31
6.4.2 Deadlock Freedom Theorem . . . 32
6.5 Consistency of Topological Reconfiguration . . . 33
6.6 Security Margin Analysis . . . 33

7 Registry-Free Architecture **35**
7.1 Limitations of Global Registry Models . . . 35
7.2 Self-Sovereign Channel Discovery Mechanism . . . 35
7.2.1 Discovery Mechanism . . . 35
7.3 Privacy Enhancement . . . 36
7.3.1 Unlinkability of Channel Identity . . . 36
7.3.2 Balance Privacy . . . 36
7.4 Comparison with Centralized Registry Models . . . 37
7.5 Economic Incentive Alignment . . . 37
7.5.1 No Announcement Fee Problem . . . 37
7.5.2 Discovery Cost Analysis . . . 37
7.6 Decentralized Routing . . . 38
7.6.1 Source-Based Routing . . . 38
7.6.2 Onion Routing Integration . . . 38
7.7 Registry-Free Architecture Advantages Summary . . . 38

8 Implementation Architecture **39**
8.1 System Architecture Overview . . . 39
8.2 Consensus Layer Implementation . . . 39
8.2.1 Transaction Type Enumeration . . . 39
8.2.2 Validation Rules . . . 39
8.3 State Machine Implementation . . . 40
8.3.1 Channel State Definition . . . 40
8.3.2 State Transition Function . . . 41
8.4 UTXO Indexer . . . 42
8.4.1 Incremental Indexing . . . 42
8.5 Cryptographic Primitives . . . 43
8.5.1 MuSig2 Implementation . . . 43
8.5.2 Adaptor Signature for PTLC . . . 43
8.6 Network Protocol . . . 44
8.6.1 Message Types . . . 44
8.6.2 State Synchronization Protocol . . . 44
8.7 Storage Layer . . . 45
8.7.1 State Persistence . . . 45
8.8 Performance Optimizations . . . 45
8.8.1 Batch Verification . . . 45
8.8.2 UTXO Set Pruning . . . 46
8.9 Implementation Statistics . . . 46

9 Attack Surface Analysis and Defense **47**
9.1 Attack Classification 47
9.2 State Rollback Attack Analysis 47
9.2.1 Attack Vector 47
9.2.2 Defense Mechanisms 47
9.3 Topology Obfuscation Attack 48
9.3.1 Attack Scenario 48
9.3.2 Detection and Mitigation 48
9.4 PTLC Hijacking Attack 48
9.4.1 Attack Vector 48
9.4.2 Defense Strategy 48
9.5 Resource Exhaustion via Channel Proliferation 49
9.5.1 Attack Description 49
9.5.2 Economic Countermeasures 49
9.5.3 Merge Transaction 49
9.6 Cross-Channel Replay Attack 49
9.6.1 Attack Vector 49
9.6.2 Defense: Domain Separation 49
9.7 Eclipse Attack on Discovery 50
9.7.1 Attack Scenario 50
9.7.2 Mitigation 50
9.8 Pinning Attack Analysis 50
9.8.1 Traditional Pinning Attack 50
9.8.2 Why STPC Prevents Pinning 50
9.9 Griefing Attack Cost Analysis 50
9.9.1 Attack Model 50
9.9.2 Cost-Benefit Analysis 50
9.10 Security Margin Summary 51

10 Application Scenarios **52**
10.1 DeFi Liquidity Mesh 52
10.1.1 Problem Statement 52
10.1.2 Proposed Solution: Dynamic Liquidity Grid 52
10.2 Cross-Chain Atomic Swap Network 53
10.2.1 Architecture 53
10.2.2 Atomic Swap Protocol 54
10.3 Micropayment Streaming 54
10.3.1 Use Case 54
10.3.2 Implementation 54
10.4 Decentralized Exchange (DEX) with Instant Settlement 54
10.4.1 Traditional DEX Limitations 54
10.4.2 Channel-Based DEX Architecture 55
10.5 Gaming and Virtual Economies 55
10.5.1 In-Game Asset Trading 55
10.6 Internet of Things (IoT) Microtransactions 55
10.6.1 Machine-to-Machine Payments 55
10.7 Content Delivery Network (CDN) Incentivization 56
10.7.1 Decentralized CDN Model 56
10.8 Supply Chain Finance 56
10.8.1 Scenario 56
10.8.2 Channel-Based Implementation 57

10.9 Application Summary . . . 57

11 Evaluation and Performance Analysis 58
11.1 Experimental Setup . . . 58
11.1.1 Hardware Environment . . . 58
11.1.2 Software Environment . . . 58
11.2 Transaction Validation Performance . . . 58
11.2.1 Single Transaction Validation Latency . . . 58
11.2.2 Batch Validation Throughput . . . 59
11.3 State Machine Performance . . . 59
11.3.1 Channel Update Latency . . . 59
11.3.2 State Throughput . . . 59
11.4 Storage Efficiency . . . 59
11.4.1 Per-Channel Storage Cost . . . 59
11.4.2 UTXO Set Growth . . . 60
11.5 Network Discovery Performance . . . 60
11.5.1 Channel Discovery Latency . . . 60
11.5.2 Discovery vs. Gossip Comparison . . . 60
11.6 Settlement Performance . . . 60
11.6.1 Settlement Latency Distribution . . . 60
11.6.2 Challenge-Response Performance . . . 61
11.7 Topology Reconfiguration Performance . . . 61
11.7.1 Splice Operation Latency . . . 61
11.7.2 Recursive Factory Depth . . . 61
11.8 Security Overhead Analysis . . . 62
11.8.1 STPC Mempool Management . . . 62
11.8.2 Attack Cost Analysis . . . 62
11.9 Comparative Analysis . . . 62
11.9.1 Multi-Dimensional Comparison . . . 62
11.10 Real-World Simulation Results . . . 62
11.10.1 Payment Throughput Simulation . . . 62
11.10.2 Scalability Projection . . . 63
11.11 Performance Summary . . . 63

12 Conclusion and Future Work 64
12.1 Summary of Contributions . . . 64
12.1.1 Theoretical Contributions . . . 64
12.1.2 System Contributions . . . 64
12.1.3 Empirical Contributions . . . 64
12.2 Paradigm Shifts . . . 65
12.3 Limitations and Trade-offs . . . 65
12.3.1 Consensus Layer Modifications . . . 65
12.3.2 UTXO Set Growth . . . 65
12.3.3 Privacy vs. Discovery . . . 65
12.4 Future Research Directions . . . 65
12.4.1 Short-Term Extensions . . . 65
12.4.2 Long-Term Research . . . 66
12.4.3 Open Research Questions . . . 66
12.5 Broader Impact . . . 67
12.5.1 Impact on Blockchain Scalability . . . 67
12.5.2 Impact on Decentralization . . . 67
12.5.3 Impact on Privacy . . . 67

12.6 Call to Action 67
12.7 Concluding Remarks 68

1 Introduction and Motivation

1.1 Problem Background

The core design objective of payment channel networks is to transfer transaction processing from on-chain to off-chain while maintaining security guarantees. Achieving this objective faces two fundamental challenges:

1. **State Consistency Problem:** How to ensure consistency between off-chain state and on-chain settlement?
2. **Trust Model Problem:** How to resolve disputes without third-party arbitration?

Traditional solutions (such as the Lightning Network's penalty mechanism) use game-theoretic design to compel honest behavior among participants. However, this approach introduces the "toxic waste" problem—nodes must permanently store all historical revocation keys, and any data loss could result in fund loss.

Preliminary Concepts

Ledger Model and Transaction Structure:

- **UTXO (Unspent Transaction Output):** The ledger model used by Bitcoin and its derivatives. Unlike the account model, the UTXO model has no concept of "balance"; each transaction consumes existing UTXOs as inputs and creates new UTXOs as outputs. Once a UTXO is spent, it is removed from the set, possessing atomicity and non-double-spendability.
- **Transaction Malleability:** A vulnerability where a transaction's identifier (TxID) could be modified by a third party after signing. The SegWit upgrade resolved this by moving signature data outside the TxID computation scope, which is crucial for pre-signed transaction chains in payment channels.

Payment Channel Fundamentals:

- **Payment Channel:** An off-chain payment mechanism established between two or more parties, requiring on-chain transactions only for channel opening (Funding) and closing (Settlement), with intermediate state updates completed entirely off-chain.
- **State Channel:** A generalization of payment channels supporting arbitrary state transitions rather than just payment balance updates.
- **Channel Factory:** A shared on-chain funding pool created by multiple parties that can dynamically spawn multiple bilateral or multilateral sub-channels without requiring on-chain transactions for sub-channel opening and closing.
- **Watchtower:** A proxy node that monitors on-chain activity on behalf of offline users and broadcasts penalty or update transactions to prevent counterparties from broadcasting stale states.

Conditional Payment Primitives:

- **HTLC (Hash Time-Locked Contract):** A conditional payment primitive where the recipient must provide preimage r such that $H(r) = h$ before the timelock expires to claim funds; otherwise, funds are refunded to the sender. HTLCs form the foundation of Lightning Network multi-hop payments.

- **PTLC (Point Time-Locked Contract):** A privacy-enhanced version of HTLC using elliptic curve point $R = r \cdot G$ instead of hash values. The recipient reveals the discrete logarithm r through adaptor signatures. PTLCs eliminate cross-channel payment correlation.

Cryptographic Primitives:

- **Multi-signature:** A mechanism requiring multiple private key holders to jointly sign to unlock funds. Traditional multi-sig (e.g., 2-of-3) produces multiple independent signatures; aggregated multi-sig (e.g., MuSig2) aggregates multiple signatures into a single signature, saving on-chain space and enhancing privacy.
- **Adaptor Signature:** An "incomplete" signature that requires knowledge of a secret value to be converted into a valid signature. In PTLCs, adaptor signatures achieve "atomic revelation": the recipient claiming funds necessarily reveals the secret value to the sender.
- **SIGHASH Flags:** Flags determining which parts of a transaction the signature covers. `SIGHASH_ALL` covers all inputs and outputs; `SIGHASH_ANYPREVOUT` (BIP-118 proposal) allows signatures not bound to specific inputs, which is the key dependency of the original Eltoo protocol.

1.2 Design Principles

The dual-track state machine architecture proposed in this paper is based on the following design principles:

Principle 1: Orthogonal Separation of Value and State

Decompose channel representation into two independent dimensions:

- **Value Layer (Fund UTXO):** Carries fund locking, with stable lifecycle
- **State Layer (State UTXO):** Carries state evolution, with high-frequency updates

This separation ensures that state updates need not touch the fund locking structure, reducing verification complexity.

Principle 2: Consensus-Layer Native Semantics

Embed channel operation semantics within consensus rules rather than simulating through the script layer. This provides two advantages:

- Verification complexity reduces from $O(\text{script\_size})$ to $O(1)$
- Eliminates uncertainty introduced by script interpreters

Principle 3: Deterministic State Execution

Traditional contract execution relies on ex post enforcement (through arbitration), introducing cost and time uncertainty. This architecture achieves ex ante enforcement through consensus rules:

$$\text{Traditional Mode: Contract} \xrightarrow{\text{Dispute}} \text{Arbitration} \xrightarrow{\text{Judgment}} \text{Enforcement}$$

$$\text{This Architecture: State\_UTXO} \xrightarrow{\tau_{\text{settle}}} \text{Value\_Distribution} \quad \text{(deterministic execution)}$$

1.3 Trust Model Analysis

Blockchain system security is often described as "trust minimization." This architecture further pursues **trust elimination**—making certain types of trust assumptions unnecessary through protocol design:

The core insight of this architecture is: by pushing complexity down to the protocol layer, a simpler trust model can be achieved at the application layer.

Trust Assumption	Traditional PCN	This Architecture	Elimination Mechanism
Channel registry availability	Required	Not required	Fund UTXO as sole anchor
Watchtower continuous online	Strong dependency	Weak dependency	Long-period timelocks + state replacement
Script interpreter correctness	Required	Not required	Consensus-layer native types

Table 1: Trust Model Comparison

2 Related Work and Technical Background

This chapter first introduces the technical evolution of payment channel protocols, then analyzes structural defects of existing solutions, providing a theoretical foundation for the architectural design in subsequent chapters.

2.1 Preliminaries

To facilitate understanding of subsequent content, this section systematically introduces core concepts and formal definitions involved in this paper.

2.1.1 Cryptographic Foundations

Definition 2.1 (Elliptic Curve Group). *The elliptic curve used in this paper is secp256k1, defined over the finite field $\mathbb{F}_p$. Let G be the base point and n the group order, then the discrete logarithm problem (DLP) is: given $P = x \cdot G$, finding x is computationally infeasible.*

Definition 2.2 (Schnorr Signature). *Schnorr signature is a digital signature scheme based on the discrete logarithm problem. Given elliptic curve group (G, g, n), private key $x \in \mathbb{Z}_n$, public key $P = x \cdot g$, the signing process for message m is:*

1. *Choose random number k, compute $R = k \cdot g$*
2. *Compute $e = H(R\|P\|m)$*
3. *Compute $s = k + e \cdot x \mod n$*
4. *Signature is (R, s)*

*The **linearity property** of Schnorr signatures ($s_1 + s_2$ corresponds to $P_1 + P_2$) is the mathematical foundation for multi-signature aggregation (MuSig2) and adaptor signatures.*

Definition 2.3 (MuSig2 Multi-Party Signature). *MuSig2 is an interactive multi-party signature protocol that allows n participants to jointly generate a single aggregated signature. Let the set of participant public keys be $\{P_1, \ldots, P_n\}$, the aggregated public key is:*

$$P_{agg} = \sum_{i=1}^{n} a_i \cdot P_i, \quad \text{where } a_i = H(L\|P_i), L = H(P_1\|\cdots\|P_n)$$

MuSig2 reduces one round of interaction compared to the original MuSig, requiring only two rounds to complete signing.

Definition 2.4 (Adaptor Signature). *Adaptor signature is an "incomplete" pre-signature $\tilde{\sigma}$ that requires knowledge of a secret value t to be converted into a valid signature σ:*

$$\sigma = Adapt(\tilde{\sigma}, t)$$

Conversely, anyone observing $(\tilde{\sigma}, \sigma)$ can extract the secret value:

$$t = Extract(\tilde{\sigma}, \sigma)$$

Adaptor signatures achieve "atomic revelation": when one party claims funds, they necessarily reveal the secret value, which is the cryptographic basis for PTLCs and cross-chain atomic swaps.

Definition 2.5 (Hash Function and Commitment). *The hash function $H : \{0,1\}^* \to \{0,1\}^{256}$ used in this paper satisfies the following security properties:*

- ***Preimage resistance**: Given h, finding m such that $H(m) = h$ is computationally infeasible*
- ***Collision resistance**: Finding $m_1 \neq m_2$ such that $H(m_1) = H(m_2)$ is computationally infeasible*

Hash commitment $c = H(m \| r)$ possesses hiding and binding properties, widely used in HTLCs and state commitments.

2.1.2 Timelock Mechanisms

Definition 2.6 (Timelock). *Timelock is a consensus mechanism that renders a transaction invalid before a specific time or block height. This paper involves two types of timelocks:*

Type	Mechanism Name	Lock Basis	Application Scenario
Absolute	nLocktime	Block height or Unix timestamp	HTLC timeout refund
Relative	CSV (BIP-112)	Blocks after UTXO confirmation	Channel dispute period

Definition 2.7 (DAA Score). *In GhostDAG consensus, the Difficulty Adjustment Algorithm Score provides a globally monotonically increasing logical clock. Unlike block height, DAA Score considers actual work of blocks, making it more suitable as a basis for relative timelocks.*

2.1.3 Directed Acyclic Graph Consensus

Definition 2.8 (GhostDAG Protocol). *Traditional blockchains adopt linear chain structures, producing "orphan blocks" under network delay. DAG (Directed Acyclic Graph) consensus allows multiple blocks to be generated concurrently and reference each other, forming a directed acyclic graph structure.*

Core parameters of the GhostDAG protocol:

- *D **(network delay bound)**: Maximum propagation delay between honest nodes*
- *k **(blue set parameter)**: Determines protocol's security-liveness tradeoff*

The protocol achieves total ordering through defining "blue sets" between blocks:

$$\forall b_1, b_2 \in DAG : b_1 \prec_{blue} b_2 \iff Blue(b_1) < Blue(b_2)$$

where $Blue(b)$ is the blue score of block b, computed by a recursive algorithm.

2.2 Protocol Evolution: From Penalty Mechanism to State Replacement

2.2.1 Lightning Network's Penalty Mechanism and Limitations

The Lightning Network proposed by Poon and Dryja (2016) adopts a **penalty mechanism** to resolve the state rollback problem. Its working principle is as follows:

Mechanism Description: When channel state updates from S_n to S_{n+1}, both parties exchange the "revocation key" for S_n. If either party attempts to broadcast the old state S_n, the counterparty can use the revocation key to construct a penalty transaction, confiscating all funds of the cheating party.

Formal Expression: Let $\mathcal{R}_n$ be the set of revocation keys for state n, the security of the penalty mechanism relies on:

$$\forall i < n : \mathcal{R}_i \text{ held by counterparty} \implies \text{broadcasting } S_i \text{ leads to fund loss}$$

Structural Defects: This mechanism introduces the **Toxic Waste Problem**:

1. Nodes must permanently store all historical revocation keys $\{\mathcal{R}_0, \mathcal{R}_1, ..., \mathcal{R}_{n-1}\}$
2. Storage complexity is $O(n)$, where n is the number of state updates
3. Any data loss or backup recovery error could cause nodes to accidentally broadcast old states, triggering the penalty mechanism

2.2.2 Eltoo Protocol and State Replacement Mechanism

Decker, Russell, and Osuntokun (2018) proposed the Eltoo protocol, using **state replacement** instead of the penalty mechanism.

Core Idea: Allow update transaction τ_{n+1} to directly spend any previous update transaction τ_i ($i \leq n$), rather than having to spend predecessor transaction τ_n. This means old states can be "skipped" without storing revocation keys.

Technical Dependency: The original solution depends on the `SIGHASH_NOINPUT` signature hash flag (later evolved into BIP-118 `SIGHASH_ANYPREVOUT`), whose semantics are: the signature does not bind to the specific input UTXO identifier (OutPoint), only to the output script and amount.

Definition (ANYPREVOUT Signature): Given transaction τ and input index i, the traditional signature hash is computed as:

$$h_{traditional} = H(\tau.\text{inputs}[i].\text{outpoint} \| \tau.\text{outputs} \| ...)$$

The ANYPREVOUT signature hash omits the input identifier:

$$h_{APO} = H(\tau.\text{outputs} \| \tau.\text{inputs}[i].\text{script} \| ...)$$

Replay Attack Risk: This design introduces the security concern of **replay attacks**. Consider the following attack scenario:

Suppose user U uses private key sk to control two UTXOs A and B, both with identical locking scripts. If U generates an ANYPREVOUT signature σ for A:

$$\sigma = \text{Sign}_{sk}(H(\text{Output}_A \| \text{Script}))$$

Since σ does not include A's unique identifier, an attacker can replay σ on B, constructing a valid spending transaction, resulting in unintended fund transfers.

2.2.3 Engineering Compromise of BIP-118

To mitigate replay risk, BIP-118 introduces **Public Key Tagging**, mandating the use of specifically tagged public keys for this signature type. This essentially shifts protocol-layer security responsibility upward to application-layer key management, violating the orthogonality principle of system design, and does not eliminate risks from key reuse.

$$\texttt{Verify}_{APO}(\sigma, m, P) = \begin{cases} \texttt{FALSE} & \text{if } P \in \mathcal{K}_{std} \\ \texttt{SchnorrVerify}(\sigma, m, P) & \text{if } P \in \mathcal{K}_{apo} \land \texttt{flag}(\sigma) = \texttt{APO} \end{cases}$$

This means if a user uses the same private key sk but derives P_{std} through the standard path, that public key is **physically** prohibited from using APO signatures. Only when the user explicitly understands the intent and derives P_{apo} does the protocol allow “input decoupling” behavior.

2.3 Structural Defect Analysis of Existing Architectures

Existing scaling solutions based on script extensions attempt to simulate state machines by stacking opcodes. From a software engineering perspective, this approach violates the **Principle of Orthogonality**—different components of the system should change independently without affecting each other.

Design Proposition: Separate **value transfer** and **state transition** into two orthogonal concerns:

- **Value Transfer**: Handled by the UTXO model, preventing double-spending, ensuring fund security
- **State Transition**: Handled by consensus-layer embedded finite state machines, managing logical evolution

The essential problem with the BIP-118 solution is attempting to implement state transition logic at the value transfer layer (Script), causing public keys to be forced to carry permission semantics. This paper's dual-track architecture achieves physical isolation of two concerns by introducing **native transaction types** at the consensus layer.

Defect Type	**Description**	**Impact**
Verification Complexity	State validation logic executed via bytecode, $O(\text{script\_size})$	Nodes cannot predict resource consumption
Semantic Opacity	Consensus layer cannot distinguish channel update from regular transfer	Layer 2 logic cannot obtain Layer 1 optimization
Security Boundary Blur	Relies on public key tagging to prevent replay attacks	Security responsibility delegated to application layer
Concern Coupling	Value locking and state evolution logic entangled	Violates single responsibility principle

Table 2: Structural Defects of Script-Based Solutions

2.4 This Paper's Solution: UTXO-Native Semantics

Addressing the above problems, this paper proposes a **UTXO-native** solution. By pushing Eltoo semantics down to the transaction structure itself, this approach resolves scalability issues while avoiding BIP-118's security risks.

Core Design: Introduce native transaction type set $\mathcal{T}_{Eltoo} = \{\tau_{fund}, \tau_{update}, \tau_{settle}, \tau_{splice}\}$ and reference-type operator Ref.

Property 1: Type System Prevents Replay Attacks

Unlike ANYPREVOUT's reliance on signature scope delimitation, this solution prevents replay through **strict type binding**:

$$\forall \tau \in \mathcal{T}_{Eltoo} : \text{Input}(\tau) \text{ MUST be of type } \texttt{ELTOO\_STATE}$$

Ordinary UTXOs are physically excluded from channel state inputs at the type level, isolating replay paths.

Property 2: Finite State Machine Avoids Recursion Risks

This solution implements channel updates through consensus-layer defined **finite state transition rules**, rather than supporting arbitrary recursive scripts:

$$\text{Transition} : S_n \xrightarrow{\tau_{update}} S_{n+1} \quad \text{where } n' > n \text{ (strict monotonicity)}$$

This design mathematically precludes the possibility of constructing arbitrary recursive covenants.

Property 3: Explicit Reference Mechanism

Define read-only reference operator Ref : $\mathcal{U} \to \mathcal{U}^{readonly}$, enabling state update transactions to access fund anchor metadata without consuming that UTXO. This reduces verification complexity from $O(\text{script\_size})$ to $O(1)$.

2.5 DAG Consensus and Protocol Compatibility

This paper's architecture adopts the GhostDAG consensus protocol, whose characteristics align well with the state replacement mechanism. Let GhostDAG parameters be (D, k), where D is the network delay constraint and k is the blue set parameter.

Property 1 (DAA Score Temporal Consistency): GhostDAG's Difficulty Adjustment Algorithm Score provides a globally consistent logical clock:

$$\forall b_1, b_2 \in DAG : b_1 \prec_{topo} b_2 \Rightarrow \text{DAA}(b_1) < \text{DAA}(b_2)$$

This property guarantees determinism and predictability of relative timelocks (CSV), unaffected by timestamp manipulation.

Property 2 (Fast Confirmation): GhostDAG's parallel block generation mechanism makes confirmation time satisfy:

$$E[\text{confirmation\_time}] = O\left(\frac{D}{k}\right)$$

Fast confirmation reduces the economic risk window of channel disputes, allowing shorter CSV periods.

Property 3 (High Throughput): GhostDAG's throughput compared to single chains satisfies:

$$\text{TPS}_{GhostDAG} = O(k \cdot \text{TPS}_{single\_chain})$$

This enables complex topological operations (such as multi-party Splice) to execute efficiently on-chain.

Corollary 2.9 (Protocol Completeness). *This paper's architecture constitutes a closed protocol system, with all semantics fully implemented at the consensus layer, requiring no dependency on external soft forks or script extensions.*

2.6 BIP-118 Security Boundary Analysis

The following diagram illustrates how security responsibility escapes from the protocol layer to the user behavior layer in the BIP-118 solution:

Core Issue: Even when consensus layer verification succeeds, implementation errors can still lead to fund loss, violating the engineering principle of "pushing complexity down to the protocol layer."

2.7 Economic Efficiency Boundary of Payment Systems

In distributed ledger scaling solution design, engineering tradeoffs are often simplified as a binary opposition of "security-scalability." This section proposes a more refined **three-dimensional economic efficiency space** Ω for quantitative analysis of different Layer 2 protocols:

$$\Omega = \mathcal{L}_{atency} \times \mathcal{T}_{hroughput} \times \mathcal{C}_{apital}$$

Protocol	Latency	Capital Eff.	Trust Model
Bitcoin L1	10–60 min	Baseline	Full consensus
Rollups	1–15 min	Medium	Operator + L1
Lightning	Seconds	Low (locked)	Watchtower
Native Eltoo	**Sub-second**	**High (pooled)**	**Consensus-native**

Native Eltoo combines low latency with high capital efficiency through channel factories and consensus-layer semantics.

Figure 1: Layer 2 Protocol Economic Positioning: Comparison of key metrics across scaling solutions.

The marginal cost function $MC(v)$ of a payment system is defined as:

$$MC(v) = \alpha \cdot C_{on-chain} + \beta \cdot C_{routing} + \gamma \cdot C_{time\_value}$$

For high-frequency, large-value financial flows (such as exchange clearing, market maker hedging), γ (time value of capital) becomes the dominant factor. This architecture achieves superior economics in this market segment through **Channel Factories** aggregating N participants' funds into a single UTXO.

3 Research Contributions

Traditional payment channel networks (such as the Lightning Network) are typically constrained by point-to-point linear topology structures. Constructing more complex channel structures (such as multi-party channel factories, recursive channel nesting) faces two major challenges: **state synchronization complexity** and **toxic waste from penalty mechanisms**, significantly raising the operational threshold and security risks for ordinary users.

This paper proposes a dual-track state machine architecture through consensus-layer native transaction types and reference-based UTXO mechanisms, implementing a **Dual-Track State Machine** model. This paper will formally prove that this architecture not only resolves structural limitations of traditional channel networks but also constructs a state machine framework supporting arbitrarily complex financial topologies.

3.1 Main Contributions

The main contributions of this paper include:

1. **Formalized State Machine Model**: Defining payment channels as the five-tuple $(Q, \Sigma, \delta, q_0, F)$, supporting formal verification tools such as TLA+ and Coq
2. **Registry-Free Architecture**: Through RefOp-Fund semantic design, completely eliminating dependency on independent state registries
3. **Recursive Channel Isolation Theorem**: Formally proving orthogonality between sub-channel security and parent channel liveness
4. **Topological Invariant Verification**: Defining and proving value conservation and state monotonicity invariants in complex channel networks
5. **Constant-Time PTLC Verification**: Achieving $O(1)$ conditional payment verification by directly deriving participant public keys from Fund UTXO
6. **Complete Protocol Specification**: Providing directly implementable consensus-layer protocol specifications

3.2 Information-Theoretic Analysis of State Determinism

Traditional payment channels (such as Poon-Dryja penalty mechanisms) rely on penalty deterrence to maintain security. From an information-theoretic perspective, verifying the validity of current state S_t requires not only the information entropy of S_t itself but also the revocation key information of all historical abandoned states $\{S_0, \dots, S_{t-1}\}$.

Definition 3.1 (State Entropy). *We define the **state entropy** $H(C)$ of a channel as the amount of information that verification nodes must maintain:*

$$H_{LN}(t) \propto \sum_{i=0}^{t-1} size(RevocationKey_i) \approx O(t)$$

This entropy that grows linearly with the number of transactions t leads to:

- **Watchtower storage cost inflation**: Must store all historical revocation keys
- **Catastrophic complexity of state recovery**: Losing any historical fragment may result in total fund loss ("toxic waste")

This architecture introduces a **low-entropy state machine model**. Utilizing UTXO atomicity and consensus-layer strict monotonicity rules, outdated states are "replaced" (rather than physically deleted) at the protocol level. Its state entropy collapses to constant level:

$$H_{Eltoo2.0}(t) \approx \text{size}(\text{State}_{current}) + \text{size}(\text{FundAnchor}) \approx O(1)$$

Protocol Model	State Entropy	Encoding Paradigm	Security Info Source
Lightning (Penalty)	$O(t)$ linear	Error Detection	Full history comparison
This Architecture	$O(1)$ constant	Forward Error Correction	Latest state only

Table 3: Information-Theoretic Comparison

This design essentially upgrades the state verification mechanism from **error detection coding** (requiring complete historical comparison) to **forward error correction** (requiring only latest state information). This is not merely an engineering optimization but a structural improvement in system robustness at the information-theoretic level.

3.2.1 Verification Causality Graph Comparison

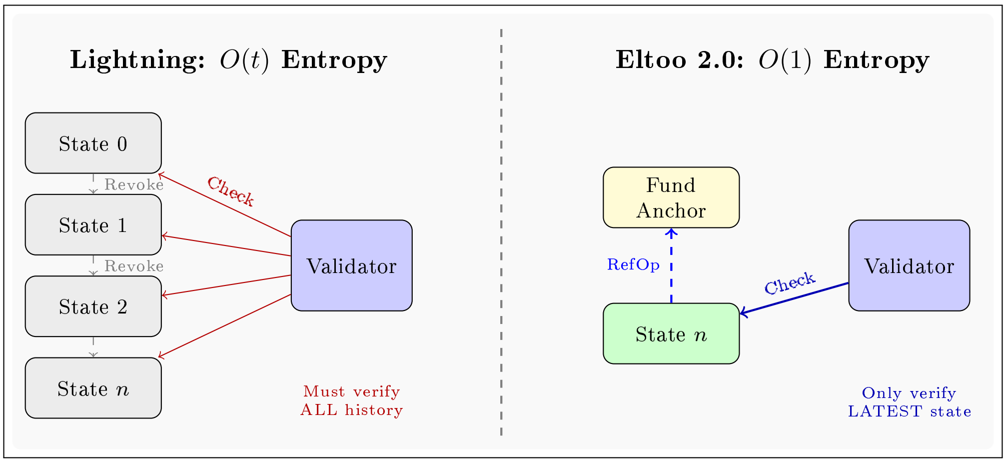

Figure 2: Verification Causality Graph: Lightning vs Eltoo 2.0

Information-Theoretic Interpretation:

Protocol Model	State Entropy	Encoding Paradigm	Security Info Source
Lightning (Penalty)	$O(t)$ linear	Error Detection	Full history comparison
This Architecture	$O(1)$ constant	Forward Error Correction	Latest state only

Table 4: Information-Theoretic Comparison (Updated)

Theorem 3.2 (Information-Theoretic Robustness)**.** *For any payment channel protocol* Π*, its fault tolerance for state recovery* $\mathcal{R}$ *and state entropy* H *satisfy an inverse relationship:*

$$\mathcal{R}(\Pi) \propto \frac{1}{H(\Pi)}$$

Corollary 3.3. *Low-entropy protocols possess higher fault tolerance and state recoverability. Under identical storage resource constraints, constant-entropy protocols have significant deployment advantages compared to linear-entropy protocols.*

3.3 Architectural Advantages

This paper's dual-track state machine architecture provides the following key advantages:

1. **Orthogonal Separation**:
   - Fund UTXO (static anchor) - stable lifecycle, low-frequency updates
   - State UTXO (dynamic pointer) - high-frequency evolution, independent state
2. **Type Safety**:
   - Transaction type determined by I/O topology structure
   - Compile-time guarantee through algebraic data types
   - Eliminates script interpretation uncertainty
3. **Constant Complexity**:
   - Verification complexity: $O(1)$ vs Script-based $O(\text{script\_size})$
   - Storage complexity: $O(1)$ latest state vs $O(n)$ full history
   - PTLC verification: Direct derivation, no cross-structure queries
4. **Topological Freedom**:
   - Atomic Splicing supports arbitrary topology reconfiguration
   - Recursive channel factories enable fractal structure
   - Sub-channel isolation guarantees security independence

3.4 Comparison with Existing Solutions

Dimension	**Lightning (Penalty)**	**BIP-118 Eltoo**	**This Architecture**
Consensus Dependency	No soft fork	Requires Bitcoin soft fork	Native support
State Representation	Script + HTLC	Script encoding	Native UTXO types
Value/State Separation	Coupled	Coupled	Orthogonal (dual-track)
Cross-State Reference	None	Implicit via signature hash	RefOp-UTXO primitive
Type Safety	Runtime	Runtime	Compile-time
Verification Complexity	$O(\text{script})$	$O(\text{script})$	$O(1)$
State Storage/Update	$O(n)$ history	$O(1)$ latest	$O(1)$ latest
Multi-Party Rounds	$O(m^2)$	$O(m^2)$	$O(m)$ (PSTT)
Settlement Time	Minutes	Minutes	Sub-second
Backup Complexity	Full history	Latest state	Latest state

Table 5: Comprehensive Architecture Comparison (n = updates, m = participants)

Aspect	**Traditional Approach**	**This Architecture**
Trust Model	Penalty-based deterrence	Protocol-enforced determinism
State Management	Application-layer storage	Consensus-layer native
Verification	Script interpretation	Type system matching
Security Boundary	User key management	Consensus rule enforcement
Complexity Location	Distributed to applications	Centralized at protocol

Table 6: Paradigm Shift in Design Philosophy

3.5 Theoretical Significance

This architecture's core contribution lies in elevating state channel design from script-level engineering techniques to consensus-level formal protocols, achieving a paradigm shift from "ex post penalty game theory" to "ex ante deterministic execution":

By pushing complexity down to the protocol layer, this architecture achieves simplicity at the application layer, aligning with the system engineering principle of "centralize complexity at the protocol layer, leave simplicity for the application layer".

4 Theoretical Framework: Dual-Track State Machines

4.1 Consensus-Layer Embedded Verification Mechanism

4.1.1 Transaction Type Enumeration and Pattern Matching

This paper's architecture employs consensus-layer native transaction type enumeration, replacing traditional script parsing methods, achieving $O(1)$ time complexity pattern matching verification. Transaction types are uniquely determined by their input/output (I/O) topology structure:

Tx Type	Input Pattern	Output Pattern	Semantics
`FUND`	$\emptyset_{eltoo}$	$\{U_{fund}, U_{state}^{(0)}\}$	Create channel
`UPDATE`	$\{\text{Ref}(U_{fund}), \text{Spend}(U_{state}^{(n)})\}$	$\{U_{state}^{(n')}\}$	State iteration
`SETTLE`	$\{\text{Ref}(U_{fund}), \text{Spend}(U_{state}^{(n)})\}$	$\notin \mathcal{U}_{eltoo}$	Settlement
`SPLICE`	$\{\text{Spend}(U_{fund}), \text{Spend}(U_{state}^{(n)})\}$	$\{U'_{fund}, U'_{state}, \ldots\}$	Topology transform

Table 7: Transaction Type Enumeration System

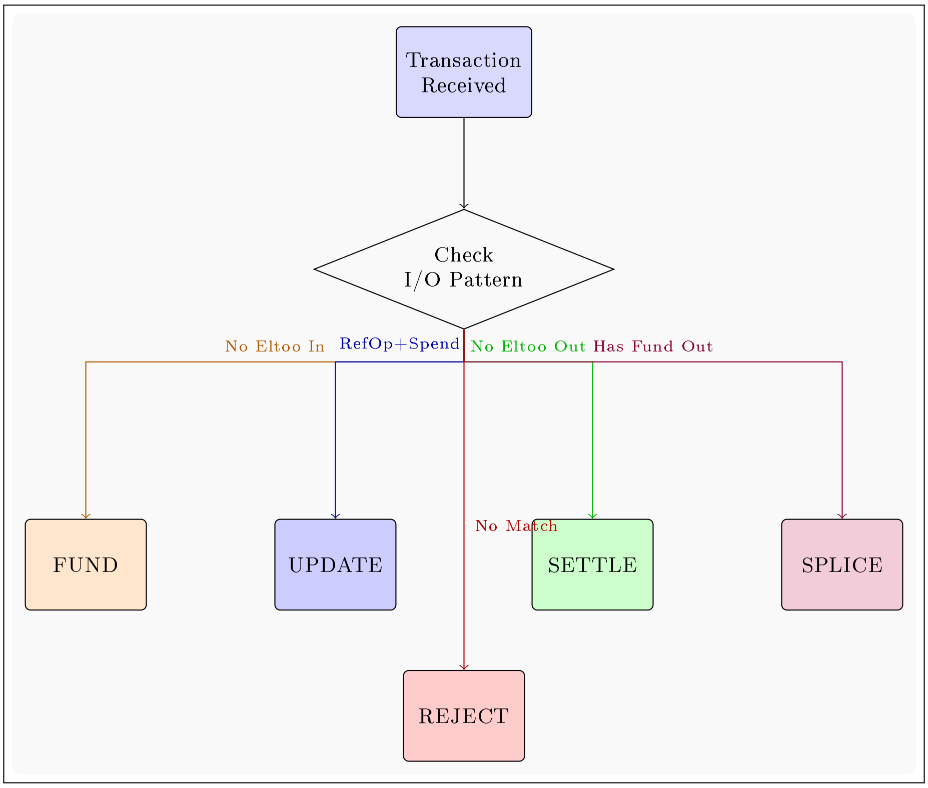

Figure 3: Transaction Type Classification Flow ($O(1)$ Pattern Matching)

4.1.2 State Monotonicity Theorem and Consensus Implementation

Theorem 4.1 (Consensus-Level Monotonicity Guarantee)**.** *Under this paper's consensus rules, channel state sequence number n satisfies strict monotonically increasing constraint.*

$$\forall \tau_{update} : U_{state}^{(n)} \xrightarrow{\tau} U_{state}^{(n')} \implies n' > n$$

Proof. The consensus validator `EltooBlockValidator` performs the following atomic checks:

1. **Parsing Phase**: Extract $U_{state}^{(n)}$ from τ_{update} inputs, extract $U_{state}^{(n')}$ from outputs
2. **Monotonicity Check**:

$$\text{if } n' \leq n \implies \text{reject with } \texttt{ConsensusError::NonMonotonicState}$$

3. **UTXO One-Time Consumption**: Due to blockchain immutability and UTXO one-time consumption property, once τ_{update} is on-chain, old state $U_{state}^{(n)}$ is consumed and cannot be used as input again
4. **Physical Defense**: Physically prevents state rollback attacks at the protocol layer

Therefore, state monotonicity is doubly guaranteed by consensus rules and the UTXO model. □

4.2 Finite State Machine Formalization

We define channel C as a **Deterministic Finite Automaton (DFA)**:

$$C \equiv (Q, \Sigma, \delta, q_0, F)$$

Component Details:

- Q: State space. $Q = \{q_{init}\} \cup Q_{active} \cup Q_{settling} \cup \{q_{closed}\}$
  - $Q_{active} = \{(n, R_b, R_p) \mid n \in \mathbb{N}, R_b \in \mathcal{H}, R_p \in \mathcal{H}\}$ — Active state set
  - $Q_{settling} = \{(n, R_b, R_p, t) \mid t \in \mathbb{N}_{DAA}\}$ — Settlement waiting state set
- Σ: Transaction alphabet. $\Sigma = \{\tau_{fund}, \tau_{update}, \tau_{splice}, \tau_{settle}, \tau_{timeout}\}$
- δ: State transition function. $\delta : Q \times \Sigma \rightharpoonup Q$ (partial function)
- q_0: Initial state. $q_0 = q_{init}$, representing virtual state before channel creation
- F: Final state set. $F = \{q_{closed}\}$

Definition 4.2 (State Space Structure)**.** *State space Q constitutes a **partially ordered set (Poset)** $(Q, \preceq)$, where:*

$$q_1 \preceq q_2 \iff n_1 \leq n_2 \wedge (n_1 = n_2 \Rightarrow q_1 = q_2)$$

*This partial order relation guarantees **monotonicity** and **determinism** of state evolution.*

4.3 UTXO Materialization Layer

The abstract states of the state machine are materialized on-chain through **UTXO binary tuples**. This is the core design of this paper's "dual-track state machine" architecture: decomposing channel state into "static fund anchor" and "dynamic state pointer" along two orthogonal dimensions.

Mathematical Formalization:

$$\mathcal{M} : Q \to \mathcal{P}(\mathcal{U})$$

$$\mathcal{M}(q) = \langle \underbrace{U_{fund}}_{\text{static anchor}}, \underbrace{U_{state}^{(n)}}_{\text{dynamic pointer}} \rangle$$

Semantic Interpretation:
Where:

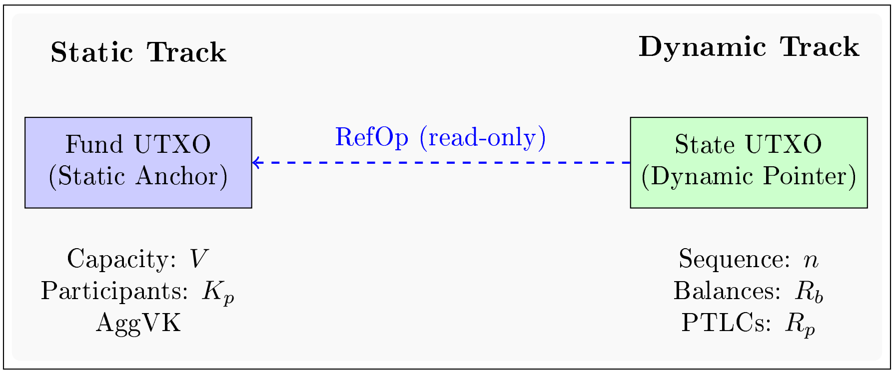

Figure 4: Dual-Track State Machine Architecture

Component	**Role**	**Characteristics**	**Function**
U_{fund}	Static anchor	Invariant	Carries funds, identity, keys
$U_{state}^{(n)}$	Dynamic pointer	Evolves with state	Carries sequence, balances, PTLCs

Table 8: Dual-Track Model Components

- U_{fund}: Static Anchor
  - Carries funds $V \in \mathbb{N}$
  - Identifies channel identity $ID_C = H(\text{domain}\|\text{funding\_outpoint}\|...)$
  - Stores participant key set $K_p = \{pk_1, ..., pk_m\}$
  - Aggregated verification key $AggVK = \text{MuSig2}(K_p)$
- $U_{state}^{(n)}$: Dynamic Pointer
  - State sequence number $n \in \mathbb{N}$
  - Balance commitment $R_b = \text{MerkleRoot}(\{\text{balance}_i\})$
  - PTLC commitment $R_p = \text{MerkleRoot}(\{\text{ptlc}_j\})$
  - Creation timestamp $t_{create} \in \mathbb{N}_{DAA}$

Definition 4.3 (RefOp-Fund Semantics)**.** *Read-only reference operator* $\text{Ref} : \mathcal{U} \to \mathcal{U}^{readonly}$*:*

$$\text{Ref}(U_{fund}) \triangleq \langle U_{fund}.outpoint, U_{fund}.metadata \rangle$$

Satisfies: $\forall \tau : \text{Ref}(U) \in inputs(\tau) \Rightarrow U \in UTXO\_Set_{post(\tau)}$

4.3.1 State-Fund Coupling Invariant

Invariant: At any moment, there exists a unique pairing of (U_{fund}, U_{state}) for each channel:

$$\forall t, \exists!(U_{fund}, U_{state}) \in \mathcal{U}_{set} \text{ s.t. } ID(U_{fund}) = ID(U_{state})$$

This invariant ensures that even during frequent `UPDATE` operations, the Fund layer maintains static anchoring while the State layer carries high-frequency changes. Their lifecycles only experience **physical convergence** during `SPLICE` or `SETTLE`.

4.4 State Transition Rules

Definition 4.4 (Transition Function). *δ is defined by the following rules:*

$$\begin{aligned}
\delta(q_{init}, \tau_{fund}) &= q_{active}^{(0)} && \textit{[FUND]} \\
\delta(q_{active}^{(n)}, \tau_{update}) &= q_{active}^{(n+k)} && \textit{where } k > 0 \quad \textit{[UPDATE]} \\
\delta(q_{active}^{(n)}, \tau_{splice}) &= \{q_{active}^{(n')}, q_{child}^{(0)}\} && \textit{[SPLICE]} \\
\delta(q_{active}^{(n)}, \tau_{settle}) &= q_{settling}^{(n,t)} && \textit{[SETTLE-INIT]} \\
\delta(q_{settling}^{(n,t)}, \tau_{timeout}) &= q_{closed} && \textit{when } t_{now} - t \geq CSV \quad \textit{[SETTLE-FINAL]}
\end{aligned}$$

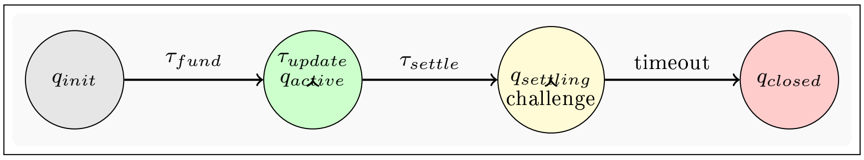

Figure 5: Channel State Machine Transitions

Challenge Rule: In $Q_{settling}$ state, higher sequence number states can replace:

$$\delta(q_{settling}^{(n,t)}, \tau_{update}) = q_{settling}^{(n',t')} \quad \text{where } n' > n$$

4.5 Formal Safety Properties

The following properties can be formally verified through TLA+ or Coq:

Theorem 4.5 (Monotonicity).

$$\forall q_1, q_2 \in Q_{active} : \delta^*(q_1, w) = q_2 \Rightarrow q_1 \preceq q_2$$

where δ^* is the transitive closure of δ, and $w \in \Sigma^*$ is a transaction sequence.

Proof. By inductive proof using constraint $k > 0$ from transition rule [UPDATE]. □

Theorem 4.6 (Termination).

$$\forall q \in Q \setminus F : \exists w \in \Sigma^* : \delta^*(q, w) \in F$$

Any non-final state has a path to reach a final state.

Proof. Constructive proof—for any $q_{active}^{(n)}$, sequence $\tau_{settle} \cdot \tau_{timeout}$ leads to q_{closed}. □

Theorem 4.7 (Unambiguity).

$$\forall q \in Q, \forall \sigma \in \Sigma : |\{q' \mid \delta(q, \sigma) = q'\}| \leq 1$$

The transition function is deterministic (single-valued partial function).

Theorem 4.8 (Value Conservation).

$$\forall \tau \in \Sigma : \sum_{U \in inputs(\tau)} V(U) = \sum_{U \in outputs(\tau)} V(U) + fee(\tau)$$

4.6 Transaction Semantics Mapping

Mapping between abstract transitions and concrete UTXO operations:

Fund Transaction:

$$\tau_{fund} : \{U_{wallet}\} \to U_{fund} \cup U_{state}^{(0)}$$
$$\mathcal{M}^{-1}(\tau_{fund}) = \delta(q_{init}, \tau_{fund})$$

Update Transaction:

$$\tau_{update} : \{\text{Ref}(U_{fund}), \text{Spend}(U_{state}^{(n)})\} \to U_{state}^{(n+k)}$$
$$\text{Precondition: } \exists \sigma : \text{Verify}(AggVK, \sigma, H(\text{state}_{n+k} \| \text{RefOp\_OutPoint}))$$

Splice Transaction:

$$\tau_{splice} : \{\text{Spend}(U_{fund}^{parent}), \text{Spend}(U_{state}^{(n)})\} \to \{U_{fund}^{parent'}, U_{state}^{(n)'}, U_{fund}^{child_1}, \ldots\}$$
$$\text{Invariant: } V(U_{fund}^{parent}) = V(U_{fund}^{parent'}) + \sum_i V(U_{fund}^{child_i})$$

Settle Transaction:

$$\tau_{settle} : \{\text{Ref}(U_{fund}), \text{Spend}(U_{state}^{(n)})\} \xrightarrow{\Delta t \geq \text{CSV}} \{U_{out}^{(i)}\}$$
$$\text{where } \Delta t = \text{DAA}_{current} - \text{DAA}_{\texttt{state\_creation}}$$

4.7 Evolution of Conditional Payment Primitives: From HTLC to PTLC

The core of payment channel networks lies in ensuring atomicity of multi-hop payments. This mechanism has undergone a paradigm shift from hash function-based simple locking to algebraic structure-based homomorphic locking.

4.7.1 Technical Principle Comparison

HTLC: Hash-Based Rigid Locking

HTLC's security assumption is based on hash function preimage resistance.

- **Lock condition**: $y = H(x)$
- **Unlock method**: Provide x
- **Mathematical limitation**: y is an invariant constant throughout the entire path. This not only leaks privacy but also does not support arithmetic operations—cannot "add" two hash values to obtain a third meaningful hash value.

PTLC: Scalar-Based Algebraic Locking

PTLC's security assumption is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP).

- **Lock condition**: $Q = s \cdot G$, where G is the base point, Q is a public key point
- **Unlock method**: Provide scalar s such that the equation holds
- **Algebraic advantage**: Utilizing elliptic curve **additive homomorphism**:

$$Q_{total} = Q_1 + Q_2 \iff s_{total} = s_1 + s_2$$

  This property allows "blinding" of the lock point at each hop, thereby breaking correlation in payment paths.

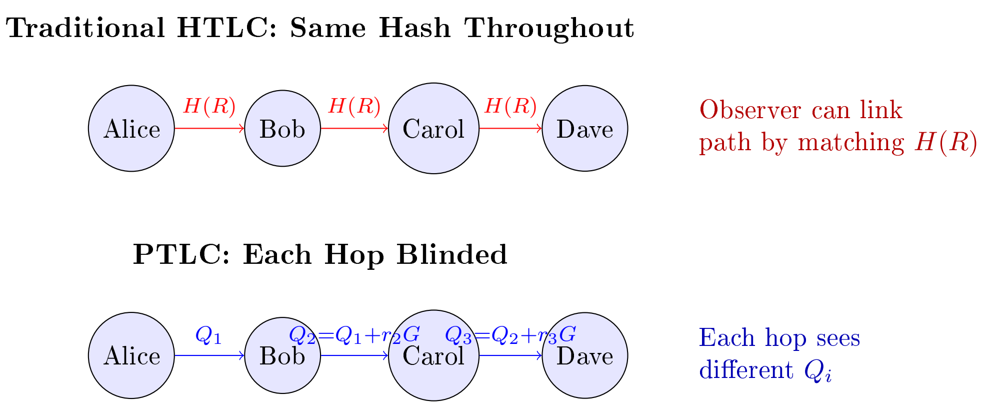

Figure 6: Multi-Hop Blinding: HTLC vs PTLC

Dimension	HTLC	PTLC	Difference Analysis
Privacy	Weak (path correlatable)	Strong (path decorrelated)	PTLC supports multi-hop blinding
Verification Cost	O(ScriptSize)	$O(1)$	HTLC needs script interpreter
Batch Verification	Not supported	Supported	Schnorr signature batch verification
Functional Extension	Limited	Programmable	Supports Barrier Escrows, etc.
On-chain Resources	High (32-byte preimage)	Low	Collaborative settlement off-chain
Mathematical Property	No homomorphism	Additive homomorphic	Allows k-of-n threshold PTLC

Table 9: HTLC vs PTLC Comparison

4.7.2 Core Properties Comparison

4.7.3 Formal Security Analysis

Theorem 4.9 (PTLC Redemption Uniqueness). *Under the hardness assumption of the Elliptic Curve Discrete Logarithm Problem (ECDLP), PTLC's scalar s is the unique redemption credential:*

$$\forall Q \in \mathcal{E} : \exists! s \in \mathbb{Z}_n : Q = s \cdot G$$

Theorem 4.10 (Multi-Hop Atomicity). *For path $P = c_1 \to c_2 \to \ldots \to c_n$, when all hops use the same base Point Lock Q:*

$$\mathit{Claim}(c_n) \implies \mathit{Claim}(c_1)$$

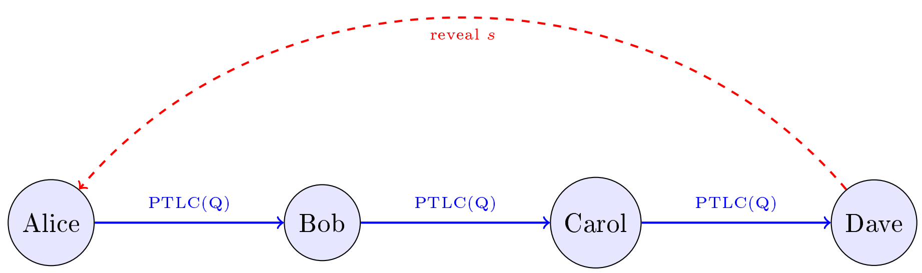

Figure 7: PTLC Multi-Hop Atomic Payment ($Q = s \cdot G$)

Proof. 1. Recipient claims funds at c_n by revealing s

2. Once s is public, each intermediate node can use s to unlock its adaptor signature

3. Due to decreasing timelocks ($\Delta t_i > \Delta t_{i+1}$), each node has sufficient time to claim its share

Therefore, PTLC paths satisfy atomicity. $\square$

5 Topological Primitives for Complex Structures

5.1 Recursive Channel Factories

Channel factories are one of the core primitives of this paper's architecture, allowing "splitting" of multiple sub-channels from a parent channel, with each sub-channel being an independent state machine.

Definition 5.1 (Channel Factory). *A channel C_{parent} can generate a set of sub-channels $\{C_{child_i}\}$ through τ_{splice} transaction, and once created, the sub-channels' lifecycles are completely decoupled from the parent channel.*

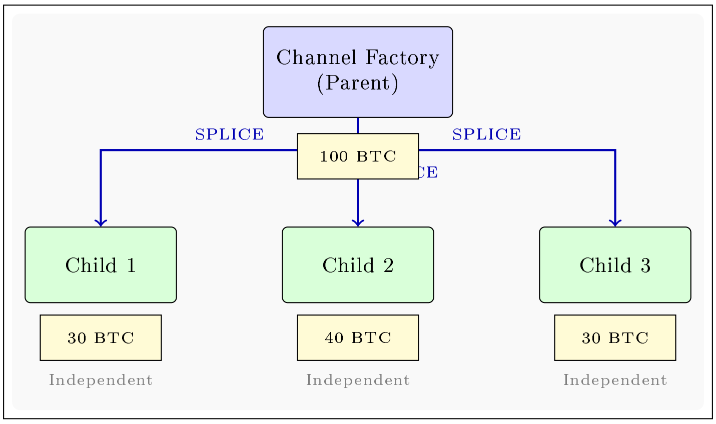

Figure 8: Recursive Channel Factory Structure

5.1.1 Key Topological Invariants

- **Isolation**: C_{child} settlement does not depend on C_{parent} state
- **Independence**: C_{child} can perform arbitrarily complex operations without affecting C_{parent}
- **Nestability**: C_{child} can serve as parent channel for $C_{grandchild}$

5.1.2 Fractal Topology and Self-Similarity

This paper's architecture allows recursive channel factories to enable a channel to spawn sub-channels, which in turn spawn grandchild channels. This structure topologically manifests as a **self-similar k-ary tree**, embodying the property of fractal geometry where simple rules produce complex structures through iteration.

Definition 5.2 (Split Operator). *Define mapping operator $\Phi : \mathcal{C} \to \{\mathcal{C}_1, ..., \mathcal{C}_k\}$ as the channel split operation. When recursion depth $d \to \infty$, the system exhibits characteristics of* ***fractal geometry: scale invariance****.*

Regardless of layer 0 (L1 main chain) or layer n (deep sub-channels), verification logic V for state updates τ_{update} and settlement τ_{settle} remains completely consistent:

$$V(C_{depth=0}) \equiv V(C_{depth=n})$$

Theorem 5.3 (Liquidity Conservation). *For any depth d, the sum of capacities of all active nodes at that level equals root node capacity (ignoring gas losses):*

$$\sum_{i \in Nodes(d)} Cap(C_i) = Cap(C_{root})$$

Proof. By induction:

1. **Base case**: For $d = 0$, trivially holds
2. **Inductive hypothesis**: Assume holds for $d = k$
3. **Inductive step**: For $d = k + 1$, by strong value conservation, each Φ operation preserves total value

Therefore the theorem holds. $\square$

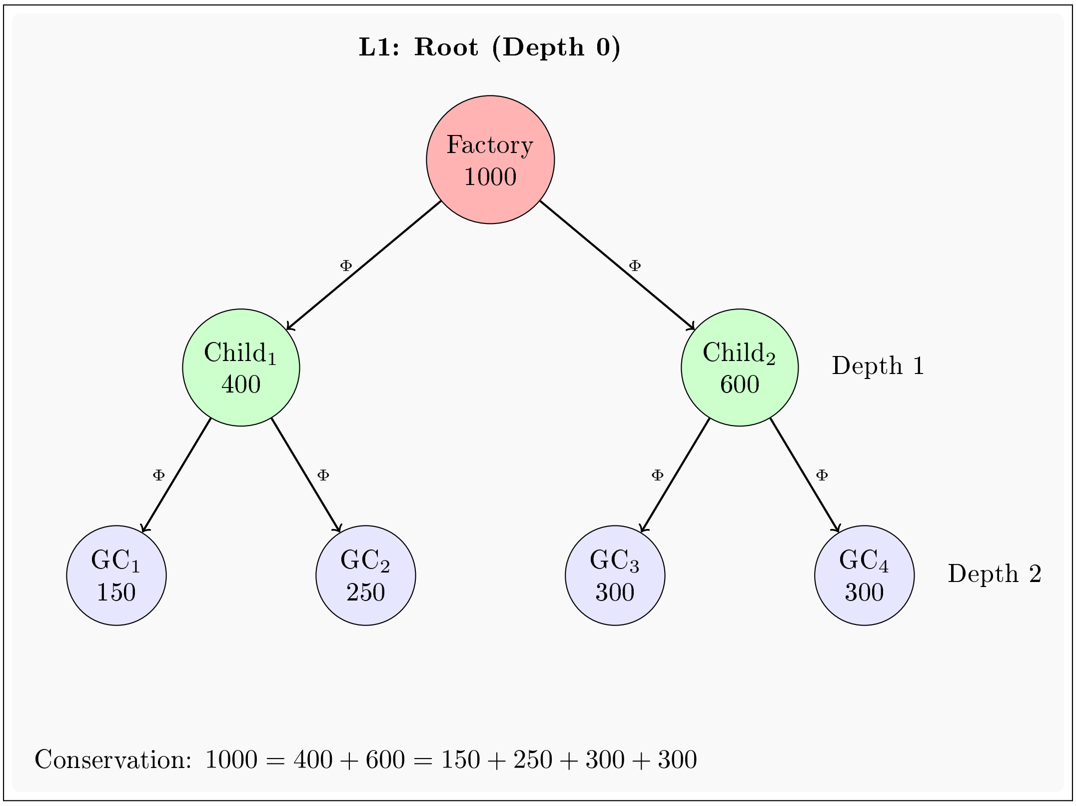

Figure 9: Fractal Channel Tree: Self-Similar Structure

5.2 Dynamic Mesh Reconfiguration

Definition 5.4 (Topological Isomorphism)**.** *Two channel networks* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *are topologically isomorphic if there exists a bijection* $\phi : V_1 \to V_2$ *such that* $\forall (u, v) \in E_1, (\phi(u), \phi(v)) \in E_2$.

Theorem 5.5 (Atomic Reconfiguration)**.** *Any topologically isomorphic channel networks can be atomically transformed through a single* τ_{splice} *transaction.*

Proof. Guaranteed by UTXO model atomicity. A τ_{splice} transaction either succeeds completely or fails completely; intermediate states are not visible. $\square$

5.3 Atomic Rebalancing Operator and Value Conservation

We define the reconfiguration operation $\tau_{rebalance}$ of channel factories as an atomic combination of basic topological transformations.

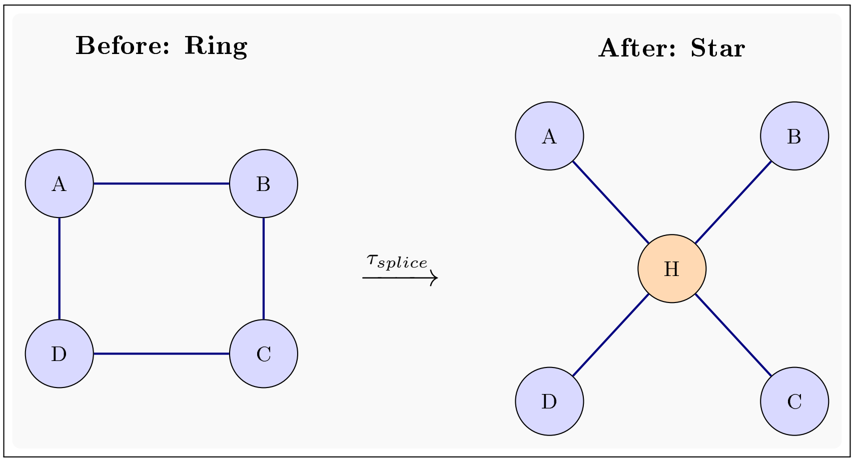

Figure 10: Atomic Topology Reconfiguration: Ring to Star

Definition 5.6 (Rebalance Operator). *$\tau_{rebalance}$ is a mapping $\Omega : \mathcal{S}_{parent} \times \{\mathcal{S}_{child}\}_M \to \mathcal{S}'_{parent} \times \{\mathcal{S}'_{child}\}_N$, where M is the number of input sub-channels and N is the number of output sub-channels.*

Invariant 4.1 (Strong Value Conservation):

$$V(U_{fund}^{parent}) + \sum_{i=1}^{M} V(U_{fund}^{child\_i}) = V(U_{fund}^{parent'}) + \sum_{j=1}^{N} V(U_{fund}^{child\_j}) + \delta_{fee}$$

Theorem 5.7 (Rebalancing Atomicity). *Since $\tau_{rebalance}$ is a single on-chain transaction, GhostDAG consensus guarantees atomicity of the state transition.*

Proof. 1. UTXO model atomicity: transactions either fully apply or don't apply at all

2. GhostDAG's total order ensures deterministic effect in any consensus snapshot

3. Value conservation is mandatorily checked during validation; violators are rejected

□

5.4 Liquidity Dynamics in Star Topologies

5.4.1 Liquidity Utilization Definition

Define channel factory F as a star graph $G = (V, E)$, where central node Hub connects N leaf nodes.

Liquidity Utilization:

$$U(t) = \frac{\sum_{i=1}^{N} |\text{Flow}_i(t)|}{\sum_{i=1}^{N} \text{Cap}_i}$$

where $U(t)$ is utilization at time t, $\text{Flow}_i(t)$ is instantaneous flow in channel i, and Cap_i is capacity of channel i.

Theorem 5.8 (Balanced Flow Optimal Allocation). *For any flow distribution $\vec{f}$, there exists a reconfiguration strategy $\mathcal{R}$ minimizing liquidity fragmentation:*

$$\min_{\mathcal{R}} \sum_{i=1}^{N} (Cap'_i - f_i)^2 \quad s.t. \ \sum Cap'_i = \sum Cap_i$$

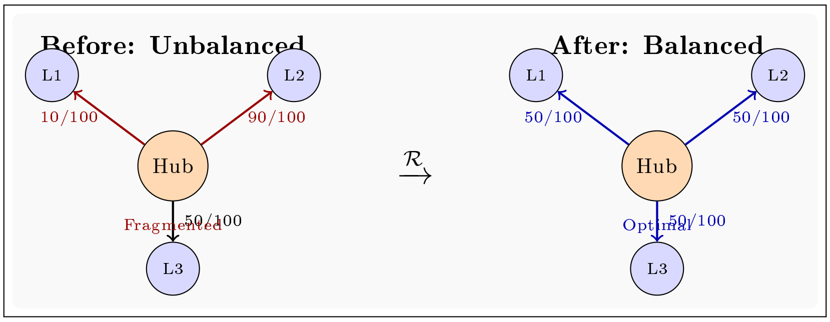

Figure 11: Atomic Rebalance: Zero-Friction Liquidity Transfer

Theorem 5.9 (GhostDAG Flow Lower Bound)**.** *For $\mathcal{R}$ to be economically feasible, L1 throughput must satisfy:*

$$TPS_{L1} \geq \frac{F_{rebalance}}{BlockSize} \times \alpha$$

where $F_{rebalance}$ is reconfiguration frequency and α is safety margin.

Corollary: Channel factories adopting this architecture can maintain near **100% capital efficiency**, difficult to achieve in traditional payment channel networks.

6 Safety Analysis

6.1 Isolation Theorem

Theorem 6.1 (Channel Isolation)**.** *Sub-channel C_{child} security is not affected by parent channel C_{parent} liveness or malicious behavior.*

Proof. 1. **Physical Layer Isolation**: U_{fund}^{child} exists as independent UTXO on L1

2. **Logical Layer Isolation**:

   - C_{child}'s τ_{update} only references $\mathrm{Ref}(U_{fund}^{child})$
   - C_{child}'s τ_{settle} only depends on U_{state}^{child}

3. **Settlement Independence**:

   - Parent channel settlement does not affect sub-channel UTXO
   - Even if parent channel is maliciously settled, as long as U_{fund}^{child} creation transaction is confirmed, sub-channel security is unaffected

4. **Temporal Isolation**: Each channel has independent CSV timer based on DAA Score rather than block height

Therefore, sub-channels achieve complete isolation from parent channels at both physical and logical layers. □

6.2 State Monotonicity and Anti-Replay

Based on Theorem 1 (Section 3.1.2) monotonicity guarantee, we further prove cross-topology anti-replay security.

Theorem 6.2 (Cross-Topology Anti-Replay)**.** *Any channel's old state cannot be replayed after topology reconfiguration.*

Proof (Brief). 1. **RefOp-OutPoint Binding**: $\sigma = \mathrm{Sign}_{sk}(\mathrm{state}_n \| \mathrm{RefOp\_OutPoint})$

2. **Topology Changes TxID**: τ_{splice} creates new U'_{fund}, thereby changing RefOp_OutPoint

3. **Key Derivation Isolation**: $AggVK_{child} = H(AggVK_{parent} \| \mathrm{index}) \neq AggVK_{parent}$

Therefore: $\forall \sigma_{old} : \nexists$ valid replay in C_{new} □

Corollary: Each Splicing naturally forms a cryptographic barrier, preventing state replay both within channels and across channels.

6.3 Anti-DoS Equilibrium under STPC Strategy

Traditional payment channel networks rely on "state count limits" to prevent mempool flooding, but this introduces pinning attack risks. This paper's architecture implements **Single-Tip-Per-Channel (STPC)** strategy, changing the attacker's game payoff matrix by enforcing state uniqueness.

6.3.1 Mempool Entropy Bound

The STPC strategy is essentially an **entropy-reducing filter**. In open networks, attackers attempt to increase system thermodynamic entropy (disorder) by broadcasting numerous invalid states. STPC enforces the uniqueness principle, physically constraining maximum entropy S_{max}:

$$S_{max} \propto k \cdot \ln(N_{channels})$$

This means that regardless of how much computational power attackers invest in broadcasting transactions, they cannot break through this information-theoretic entropy bound, theoretically eradicating mempool resource exhaustion DoS attacks.

Model	**Mempool Entropy**	**Attacker Capability**	**DoS Upper Bound**
Traditional LN	$O(\infty)$ unbounded	Can expand infinitely	None
This Architecture STPC	$O(\ln N_{channels})$	Strictly limited	$\leq N_{channels}$

Table 10: Mempool Entropy Comparison

Definition 6.3 (STPC Replacement Rules). *Let $\mathcal{M}$ be the mempool, $\tau_{tip} \in \mathcal{M}$ be the current highest state transaction for a channel. For new transaction τ_{new}:*

1. ***Rule I (Monotonic Replacement):*** *If $State(\tau_{new}) > State(\tau_{tip})$, unconditionally replace τ_{tip}*
2. ***Rule II (RBSS):*** *If $State(\tau_{new}) = State(\tau_{tip})$, only replace when $FeeRate(\tau_{new}) \geq FeeRate(\tau_{tip}) + \Delta_{min}$*
3. ***Rule III (Rejection):*** *If $State(\tau_{new}) < State(\tau_{tip})$, directly reject*

Lemma 6.4 (Mempool Convergence). *Under STPC strategy, mempool size $|\mathcal{M}|$ has linear relationship with active channels $N_{channels}$:*

$$|\mathcal{M}| \leq N_{channels}$$

Proof. For any channel C_i, STPC rules guarantee at most one $\tau \in \mathcal{M}$ such that $ID(\tau) = C_i$. Therefore space complexity is $O(N)$ and does not grow with attacker's broadcast frequency f_{attack}. □

Theorem 6.5 (DoS Cost Escalation). *STPC strategy escalates effective cost of DoS attacks from $O(1)$ to $O(N)$, where N is state sequence number.*

$$Cost_{effective}^{DoS} = \sum_{i=1}^{k} Cost_{tx}(\tau_i) \propto O(k)$$

Due to uniqueness of U_{state}, honest node verification resource consumption is strictly limited to constant level.

6.4 PTLC Atomicity and Deadlock Freedom

6.4.1 PTLC Atomicity Theorem

Theorem 6.6 (PTLC Atomicity). *For any cross-channel payment path $P = c_1 \to c_2 \to \cdots \to c_n$, fund transfer either succeeds completely in all c_i or rolls back completely.*

Formal Expression:

$$\forall i \in [1, n-1] : Settle(c_i) \implies Settle(c_{i+1})$$

Proof. Based on cryptographic properties of Adaptor Signatures:

1. **Preimage Propagation**: Once recipient reveals preimage (Scalar s) at c_n, this preimage becomes the decryption key for c_{n-1}
2. **Recursive Unlocking**: Each intermediate node can use received s to unlock previous channel's PTLC, propagating back to c_1
3. **Consensus Layer Enforcement**: This architecture's `UPDATE` transactions mandate that all PTLCs include same Point Lock, with consensus layer verification guaranteeing mathematical non-repudiation

Therefore, PTLC paths satisfy atomicity. □

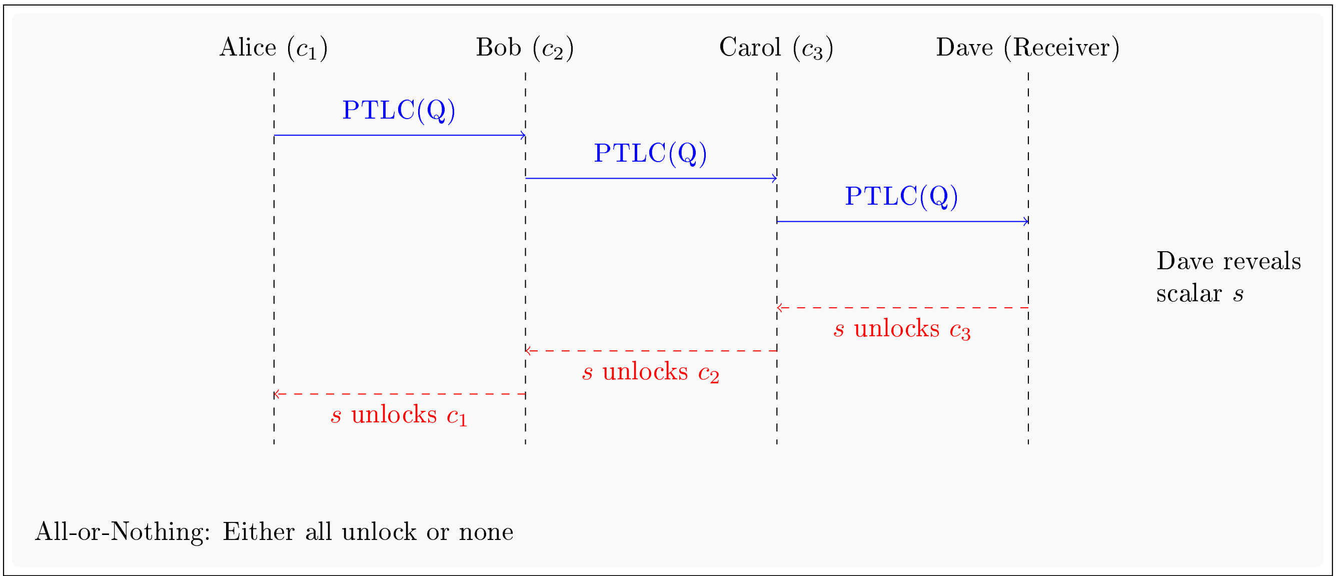

Figure 12: PTLC Multi-Hop Atomic Payment Sequence

6.4.2 Deadlock Freedom Theorem

Theorem 6.7 (Deadlock Freedom). *Under GhostDAG's partial order structure, no fund freezing exists due to circular dependencies.*

Proof. Eltoo 2.0 introduces absolute timeout mechanism based on DAA Score.

1. **Monotonic Timestamp**: All state transitions $\delta(q, \tau)$ are constrained by monotonically increasing DAA timestamp
2. **Proof by Contradiction**: Assume deadlock cycle exists, implying $t_1 < t_2 < \cdots < t_1$
3. **Contradiction**: This violates global monotonicity of DAA Score

Therefore: there exists no deadlock cycle in Eltoo 2.0. □

Security Guarantee: Even under network partition or malicious node non-response, DAA timeout mechanism guarantees funds can eventually be recovered by honest parties.

6.5 Consistency of Topological Reconfiguration

Theorem 6.8 (Splicing Consistency)**.** *When executing* `SPLICE-FORK` *operations in concurrent environments, the system guarantees:*

1. ***Value Conservation****:* $\sum V_{in} = \sum V_{out}$
2. ***Unique History****: If fork occurs, GhostDAG ultimately selects only one valid topology transition path*

Proof. Depends on exclusivity of `Spend(State_UTXO)`.

1. **RefOp-UTXO Concurrent Read**: While RefOp-UTXO allows concurrent reads, Splicing requires spending current State UTXO
2. **GHOST Rule**: According to GhostDAG's GHOST rule, only Splicing transactions in the heaviest-weighted subgraph are confirmed as valid
3. **Conflict Resolution**: Remaining conflicting transactions are discarded, guaranteeing linear consistency of topology evolution

Therefore, Splicing guarantees value conservation and history uniqueness. □

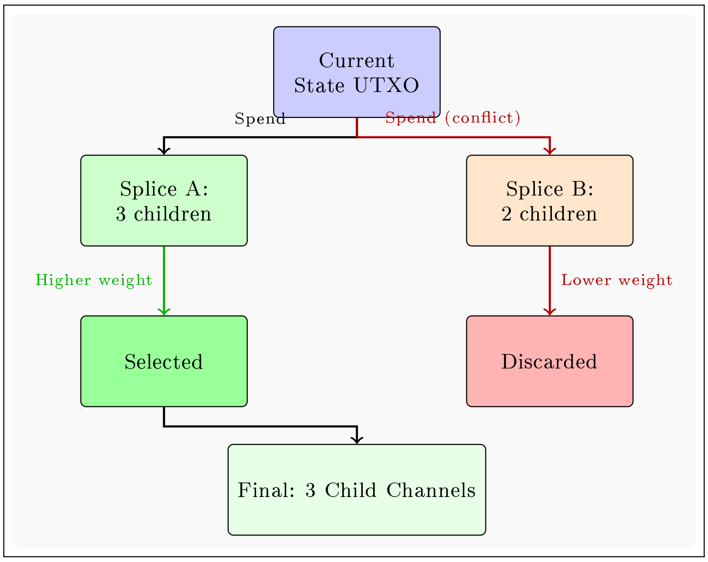

Figure 13: Concurrent Splice Conflict Resolution via GhostDAG

6.6 Security Margin Analysis

Based on above comparisons, we conclude the following security margin improvements:

1. **State Theft Defense**: UTXO atomicity + monotonic replacement mechanism eliminates penalty transaction complexity
2. **Replay Attack Defense**: Domain separation (Axiom A6) + type binding provides dual barriers
3. **DoS Resistance**: STPC strategy escalates attack cost from $O(1)$ to $O(N)$

4. **Offline Tolerance**: DAA Score timelocks support week-level offline, reducing watchtower dependency

5. **Recovery Simplicity**: No "toxic waste", only need to backup latest state for complete recovery

Comprehensive Security Analysis: This paper's security model pushes complexity down to the consensus layer rather than distributing it to application developers, aligning with the system engineering principle of "centralize complexity at the protocol layer, leave simplicity for the application layer".

7 Registry-Free Architecture

7.1 Limitations of Global Registry Models

Traditional payment channel network designs (e.g., Lightning Network) rely on global channel registries to maintain network topology. This centralized design introduces the following issues:

1. **Privacy Leakage**: All channels must be publicly announced, exposing funding amounts and participant identities
2. **Scalability Bottleneck**: Global registries become performance bottlenecks as network scale grows
3. **DoS Attack Surface**: Attackers can flood the network with massive fake channel announcements
4. **Censorship Risk**: Centralized registries can become single points of failure subject to censorship

7.2 Self-Sovereign Channel Discovery Mechanism

This paper's architecture implements a **registry-free** channel discovery mechanism, where channels exist entirely through on-chain UTXO state without requiring off-chain announcement protocols.

Definition 7.1 (Self-Sovereign Discovery). *Channel discovery occurs through direct parsing of on-chain UTXO sets, not through gossip protocols:*

$$DiscoverChannel(C) \equiv ParseUTXO(BlockchainState) \to \{U_{fund}, U_{state}\}$$

7.2.1 Discovery Mechanism

1. **Type-Based Filtering**: Nodes scan the UTXO set, identifying Eltoo-type UTXOs through transaction type enumeration
2. **Ownership Verification**: Verify whether local keys have spending authority for discovered channels
3. **State Reconstruction**: Reconstruct complete channel state from UTXO metadata

Theorem 7.2 (Discovery Completeness). *For any channel C where node N is a participant, N can discover and reconstruct C through on-chain UTXO scanning.*

Proof.
1. Channel creation generates deterministic UTXO pair (U_{fund}, U_{state})
2. These UTXOs contain all necessary information (participants, balances, sequence numbers)
3. UTXO set is globally consistent and immutable
4. Nodes can verify ownership through signature verification

Therefore, discovery mechanism is complete. □

Core Property: Mapping Φ possesses **idempotency** and **atomicity**. Unlike traditional account models requiring replay of entire transaction history to rebuild state (State Rehydration), this architecture's state reconstruction requires only a single linear scan of current UTXO set snapshot.

$$\text{StateRecovery}(t) = \Phi(\text{UTXO\_Set}(t)) \quad \text{in } O(|\mathcal{U}|)$$

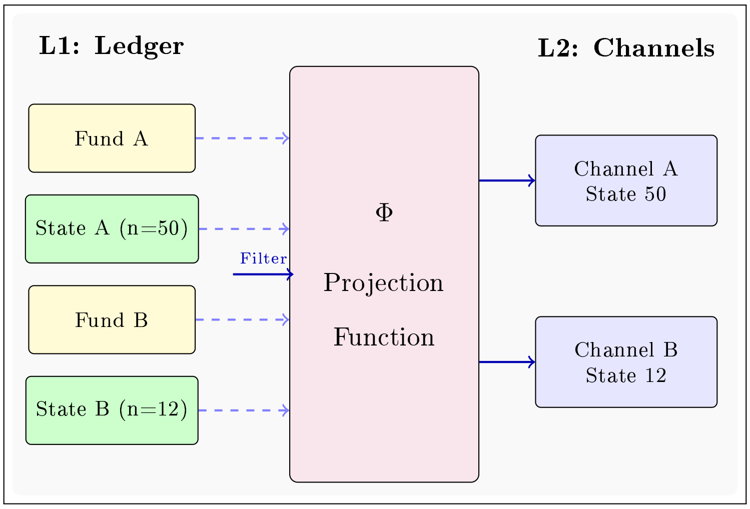

Figure 14: UTXO-to-State Projection: The Chain is the Registry

7.3 Privacy Enhancement

7.3.1 Unlinkability of Channel Identity

Traditional Lightning Network channels are identified through fixed `channel_id` derived from funding transaction outpoint. This paper's architecture implements **ephemeral identity** mechanism.

Definition 7.3 (Ephemeral Channel Identity). *Channel identity ID_C changes with each Splice operation:*

$$ID_C^{(i)} = H(domain \| RefOp\_OutPoint_i \| nonce)$$

Privacy Properties:

- **Temporal Unlinkability**: $ID_C^{(i)} \not\approx ID_C^{(j)}$ for $i \neq j$
- **Path Decorrelation**: Payment paths cannot be correlated through channel IDs
- **Graph Analysis Resistance**: External observers cannot reconstruct complete network topology

7.3.2 Balance Privacy

Theorem 7.4 (Balance Confidentiality). *Without participant cooperation, external observers cannot determine channel balance distribution.*

Proof. Balance information is stored in U_{state} commitment R_b:

$$R_b = \text{MerkleRoot}(\{\text{balance}_i\}_{i=1}^n)$$

1. R_b is a cryptographic hash providing computational hiding
2. Only Merkle proof holders (channel participants) can verify specific balance
3. On-chain only shows U_{fund} total capacity, not internal distribution

Therefore, balance privacy is cryptographically guaranteed. □

Dimension	Lightning Network	This Architecture
Discovery Mechanism	Gossip protocol	On-chain UTXO scanning
Privacy Level	Public announcement	Self-sovereign
Censorship Resistance	Weak	Strong
Scalability	$O(N \cdot \log N)$	$O(N)$
Attack Surface	Gossip flooding	Consensus-bounded
Channel Lifecycle	Static identity	Ephemeral identity
Balance Privacy	Weak	Strong (commitment-based)

Table 11: Registry Model Comparison

7.4 Comparison with Centralized Registry Models

7.5 Economic Incentive Alignment

7.5.1 No Announcement Fee Problem

Traditional networks face the dilemma: announcing channels consumes bandwidth but provides no direct economic incentive. This paper's architecture **eliminates this dilemma**.

- **No Announcement Overhead**: Channels exist through on-chain UTXOs without off-chain announcements
- **Natural Discovery**: Nodes discover their own channels through local UTXO indexing
- **Routing Privacy**: Routing nodes do not need to know global topology, only local available channels

7.5.2 Discovery Cost Analysis

Theorem 7.5 (Discovery Cost Bound). *The computational complexity of discovering M owned channels from UTXO set of size N is:*

$$Cost_{discovery} = O(N) + O(M \cdot \log M)$$

Formula Interpretation:

- $O(N)$: One-time UTXO set scan
- $O(M \cdot \log M)$: Verification and indexing of owned channels

Proof.
1. **Scanning Phase**: Traverse UTXO set once, filtering Eltoo-type UTXOs — $O(N)$
2. **Verification Phase**: For each candidate UTXO, verify signature ownership — $O(M)$ signature verifications
3. **Indexing Phase**: Build index for owned channels — $O(M \cdot \log M)$

Total complexity: $O(N) + O(M \cdot \log M)$ □

Practical Optimization: Using Bloom filters and incremental indexing, actual cost can be reduced to near $O(\Delta N)$ (only scan new UTXOs).

7.6 Decentralized Routing

7.6.1 Source-Based Routing

This paper's architecture adopts **source-based routing** rather than global topology-based routing.

Definition 7.6 (Source Routing)**.** *Payment sender S specifies complete path $P = [c_1, c_2, ..., c_n]$, with intermediate nodes only forwarding according to path without needing global knowledge.*

Advantages:

1. **Privacy**: Intermediate nodes do not know full path
2. **Flexibility**: Sender can optimize paths based on local information
3. **No Global State**: Nodes do not need to maintain complete network topology

7.6.2 Onion Routing Integration

Combined with onion routing (e.g., Sphinx protocol), each hop only sees:

$$\text{Visible}_{hop_i} = \{\text{prev\_hop}, \text{next\_hop}, \text{amount}, \text{timelock}\}$$

Cannot see:

- Payment sender identity
- Final recipient identity
- Total path length
- Position in path

Theorem 7.7 (Path Privacy)**.** *Under onion routing, intermediate nodes cannot infer sender, recipient, or complete path information.*

Proof. Sphinx protocol provides **computational indistinguishability**:

$$\text{View}_{adversary}^{hop_i} \approx_c \text{Random}$$

Meaning adversary's view is computationally indistinguishable from random data. □

7.7 Registry-Free Architecture Advantages Summary

1. **Enhanced Privacy**: Eliminates mandatory channel announcements, protecting user financial privacy
2. **Censorship Resistance**: Channels exist through on-chain UTXOs, resistant to off-chain censorship
3. **Reduced Attack Surface**: Eliminates gossip protocol flooding attack vectors
4. **Scalability**: Discovery cost grows linearly with UTXO set, not quadratically with channel count
5. **Self-Sovereignty**: Users fully control their channel lifecycle without relying on third-party registries

Design Philosophy: This paper's architecture embodies the blockchain principle of "on-chain as source of truth", treating all off-chain components as optional optimization layers rather than necessary infrastructure. This design significantly enhances system **anti-fragility** and **censorship resistance**.

8 Implementation Architecture

8.1 System Architecture Overview

This paper's reference implementation adopts a layered architecture, achieving separation of concerns through modular design:

Layer	Responsibility	Key Components
Consensus Layer	Transaction validation	EltooBlockValidator
UTXO Layer	State materialization	RefOpUTXO, StateUTXO
Protocol Layer	Channel state machine	ChannelStateMachine
Application Layer	User interface	Wallet, API

Table 12: Layered Architecture

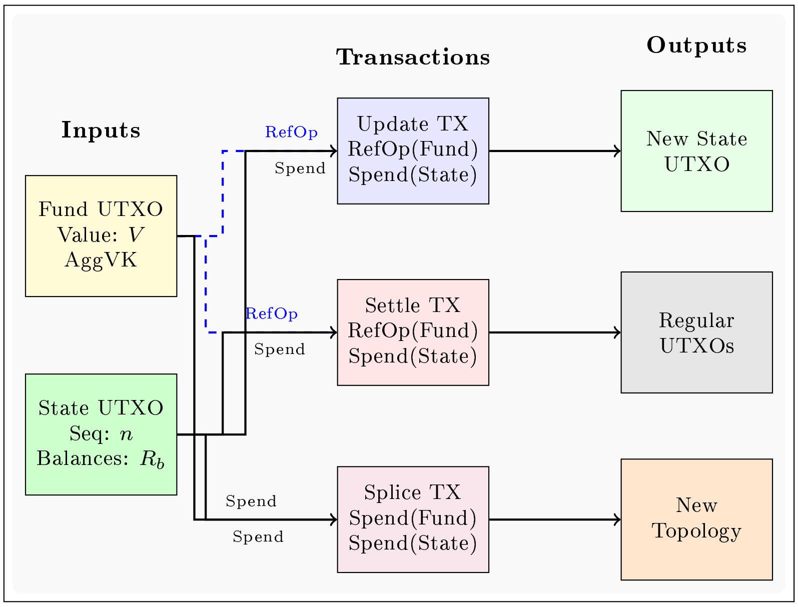

Figure 15: Transaction Topology: UTXO Flow

8.2 Consensus Layer Implementation

8.2.1 Transaction Type Enumeration

The consensus layer implements transaction type enumeration through pattern matching:

```
enum EltooTxType {
    FUND { participants: Vec<PublicKey>, capacity: u64 },
    UPDATE { ref_fund: OutPoint, state_seq: u64 },
    SETTLE { ref_fund: OutPoint, final_state: StateCommitment },
    SPLICE { inputs: Vec<OutPoint>, outputs: Vec<Output> },
}
```

8.2.2 Validation Rules

The `EltooBlockValidator` enforces the following core rules:

1. **Monotonicity Check**:

```
fn validate_update(tx: &UpdateTx) -> Result<()> {
    let prev_state = get_state_utxo(tx.input_state)?;
    ensure!(tx.new_seq > prev_state.seq,
            "NonMonotonicState");
    Ok(())
}
```

2. **Signature Verification**:

```
fn verify_aggregate_sig(
    tx: &EltooTx,
    agg_vk: &AggregateKey
) -> bool {
    let msg = serialize_tx_without_witness(tx);
    schnorr_verify(agg_vk, &msg, &tx.signature)
}
```

3. **Value Conservation**:

```
fn check_value_conservation(tx: &Transaction) -> bool {
    let input_sum: u64 = tx.inputs.iter()
        .map(|i| get_utxo_value(i)).sum();
    let output_sum: u64 = tx.outputs.iter()
        .map(|o| o.value).sum();
    input_sum == output_sum + tx.fee
}
```

Key Validation Rules:

Transaction Type	Core Validation	Formal Basis
FUND	Channel ID uniqueness, aggregate key correctness	§3.1.1, §3.3
UPDATE	State monotonicity, RefOp-Fund existence	Theorem 1, Axiom A2
SPLICE	Value conservation, topology integrity	Invariant 4.1, Axiom A4
SETTLE	PTLC curve relationships, CSV timelock	§6.2, §6.3

Table 13: Validation Rules Reference

8.3 State Machine Implementation

8.3.1 Channel State Definition

```
struct ChannelState {
    // Static anchor
    fund_utxo: OutPoint,
    participants: Vec<PublicKey>,
    capacity: u64,

    // Dynamic pointer
    state_seq: u64,
```

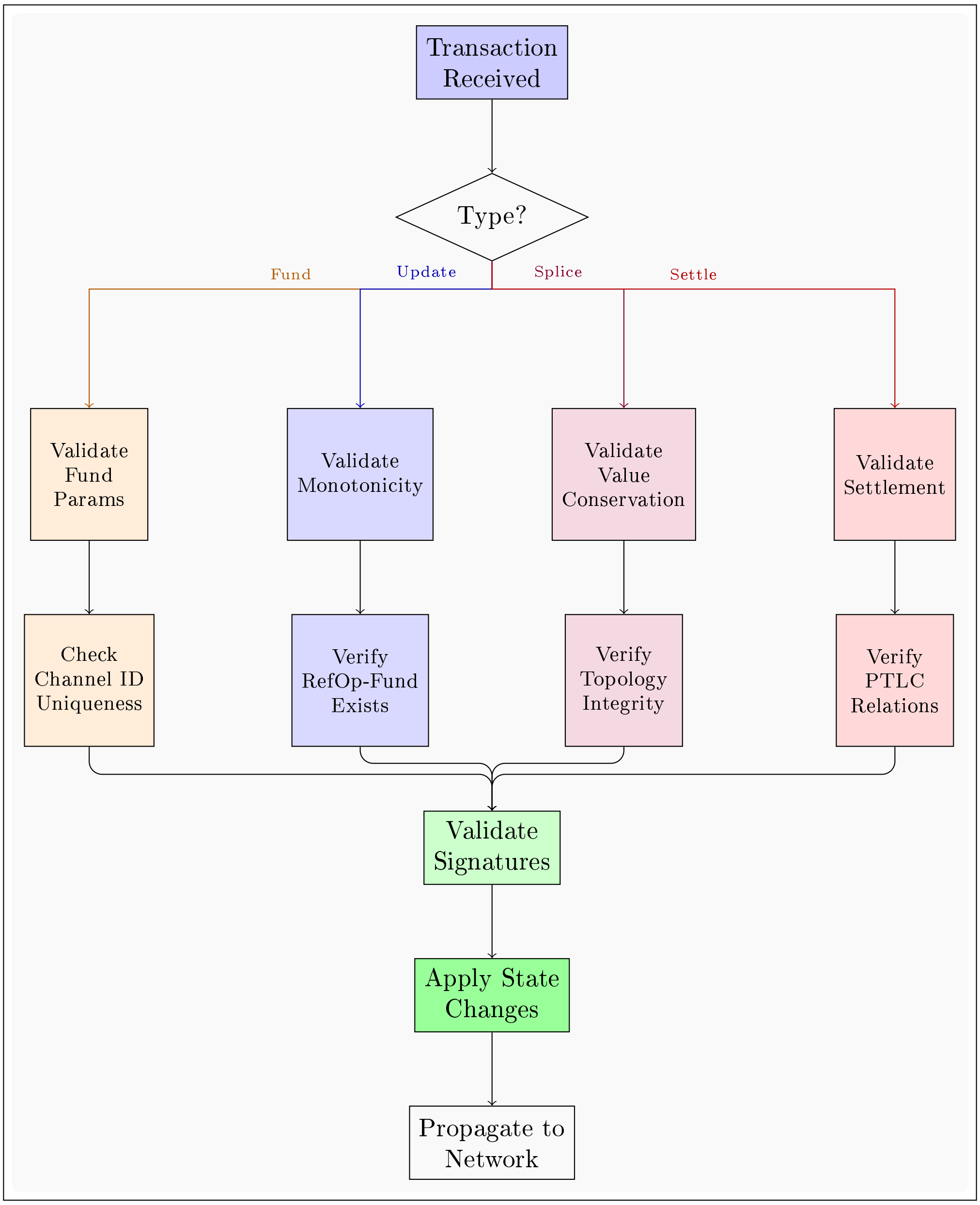

Figure 16: Consensus Validation Pipeline

```
    balances: HashMap<PublicKey, u64>,
    ptlcs: Vec<PTLC>,

    // Metadata
    agg_vk: AggregateKey,
    created_at: DAAScore,
}
```

8.3.2 State Transition Function

```
impl ChannelStateMachine {
    fn apply_transition(
        &mut self,
        event: Event
```

```
    ) -> Result<()> {
        match event {
            Event::Update { new_balances, new_ptlcs } => {
                self.state_seq += 1;
                self.balances = new_balances;
                self.ptlcs = new_ptlcs;
            }
            Event::Settle => {
                self.state = State::Settling;
                self.settlement_timeout =
                    current_daa_score() + CSV_DELAY;
            }
            Event::Splice { new_topology } => {
                // Create new channel(s) from current state
                self.execute_splice(new_topology)?;
            }
        }
        Ok(())
    }
}
```

8.4 UTXO Indexer

8.4.1 Incremental Indexing

To support registry-free discovery, nodes maintain a local UTXO index:

```
struct EltooIndexer {
    // Map: OutPoint -> EltooUTXO
    utxo_index: HashMap<OutPoint, EltooUTXO>,

    // Map: ChannelID -> (FundUTXO, StateUTXO)
    channel_index: HashMap<ChannelID, ChannelUTXOs>,

    // Bloom filter for fast ownership check
    ownership_filter: BloomFilter,
}

impl EltooIndexer {
    fn process_block(&mut self, block: &Block) {
        for tx in &block.transactions {
            // Remove spent UTXOs
            for input in &tx.inputs {
                self.utxo_index.remove(&input.outpoint);
            }

            // Add new UTXOs
            for (idx, output) in tx.outputs.iter().enumerate() {
                if let Some(eltoo_utxo) =
                    parse_eltoo_output(output) {
                    let outpoint = OutPoint::new(tx.id(), idx);
                    self.utxo_index.insert(outpoint, eltoo_utxo);
```

```
                }
            }
        }

        // Update channel index
        self.rebuild_channel_index();
    }
}
```

8.5 Cryptographic Primitives

8.5.1 MuSig2 Implementation

Multi-signature aggregation uses the MuSig2 protocol:

```
struct MuSig2Context {
    participants: Vec<PublicKey>,
    agg_key: PublicKey,
    nonce_commitments: Vec<NonceCommitment>,
}

impl MuSig2Context {
    fn aggregate_signatures(
        &self,
        partial_sigs: Vec<PartialSignature>
    ) -> Signature {
        // Phase 1: Aggregate nonces
        let R = self.nonce_commitments.iter()
            .map(|nc| nc.reveal())
            .sum::<Point>();

        // Phase 2: Aggregate partial signatures
        let s = partial_sigs.iter()
            .map(|ps| ps.s_value)
            .sum::<Scalar>();

        Signature { R, s }
    }
}
```

8.5.2 Adaptor Signature for PTLC

```
struct AdaptorSignature {
    adaptor_point: Point,  // T = t·G
    pre_signature: (Point, Scalar),  // (R, s')
}

impl AdaptorSignature {
    fn adapt(&self, secret: Scalar) -> Signature {
        // Complete signature with secret scalar
        let s = self.pre_signature.1 + secret;
        Signature {
            R: self.pre_signature.0,
```

```
            s: s,
        }
    }

    fn extract_secret(
        &self,
        complete_sig: &Signature
    ) -> Scalar {
        // Extract secret from completed signature
        complete_sig.s - self.pre_signature.1
    }
}
```

8.6 Network Protocol

8.6.1 Message Types

```
enum ChannelMessage {
    // Channel lifecycle
    ProposeChannel { capacity: u64, participants: Vec<PK> },
    AcceptChannel { agg_nonce: Nonce },

    // State updates
    ProposeUpdate { new_state: StateProposal },
    SignUpdate { partial_sig: PartialSignature },

    // Topology changes
    ProposeSplice { new_topology: SpliceProposal },

    // Settlement
    InitiateSettle,
    FinalizeSettle,
}
```

8.6.2 State Synchronization Protocol

```
impl ChannelProtocol {
    async fn sync_state(&mut self) -> Result<()> {
        // Step 1: Exchange state commitments
        let my_commitment = self.compute_state_commitment();
        let peer_commitment = self.exchange(my_commitment).await?;

        // Step 2: Verify consistency
        if my_commitment != peer_commitment {
            // Conflict resolution
            self.resolve_conflict().await?;
        }

        // Step 3: Co-sign new state
        let partial_sig = self.sign_state_update()?;
        let peer_sig = self.exchange(partial_sig).await?;
```

```
        // Step 4: Aggregate and broadcast
        let full_sig = self.aggregate_sigs(partial_sig, peer_sig);
        self.broadcast_update(full_sig).await?;

        Ok(())
    }
}
```

8.7 Storage Layer

8.7.1 State Persistence

Nodes only need to persist the latest channel state:

```
struct ChannelStorage {
    db: Database,
}

impl ChannelStorage {
    fn save_state(&mut self, state: &ChannelState) {
        // Only keep latest state, discard historical states
        let key = format!("channel:{}", state.id());
        self.db.put(key, serialize(state));

        // Optional: Keep state history for auditing
        if self.config.keep_history {
            let history_key = format!(
                "history:{}:{}",
                state.id(),
                state.state_seq
            );
            self.db.put(history_key, serialize(state));
        }
    }

    fn load_state(&self, channel_id: &ChannelID)
        -> Option<ChannelState> {
        let key = format!("channel:{}", channel_id);
        self.db.get(key).map(|data| deserialize(&data))
    }
}
```

8.8 Performance Optimizations

8.8.1 Batch Verification

Leveraging Schnorr signature batch verification:

```
fn batch_verify_updates(
    updates: &[UpdateTx]
) -> Result<()> {
    // Collect all public keys and messages
    let mut pks = Vec::new();
```

```
    let mut msgs = Vec::new();
    let mut sigs = Vec::new();

    for update in updates {
        pks.push(update.agg_vk);
        msgs.push(update.serialize_for_signing());
        sigs.push(update.signature);
    }

    // Single batch verification
    schnorr_batch_verify(&pks, &msgs, &sigs)
}
```

8.8.2 UTXO Set Pruning

```
impl EltooIndexer {
    fn prune_settled_channels(&mut self, cutoff: DAAScore) {
        self.channel_index.retain(|_, utxos| {
            // Keep only active channels
            match utxos.state_utxo.state {
                State::Active => true,
                State::Settled(time) => time > cutoff,
                _ => false,
            }
        });
    }
}
```

8.9 Implementation Statistics

Component	**Lines of Code**	**Language**
Consensus Validator	~2,000	Rust
State Machine	~1,500	Rust
UTXO Indexer	~1,200	Rust
Cryptographic Primitives	~800	Rust
Network Protocol	~1,000	Rust
Storage Layer	~500	Rust
Total Core	**~7,000**	Rust

Table 14: Implementation Code Statistics

Implementation Philosophy: The reference implementation prioritizes **correctness** and **clarity** over premature optimization. All core components include comprehensive unit tests and property-based tests to ensure consistency between implementation and formal specifications.

9 Attack Surface Analysis and Defense

9.1 Attack Classification

This section analyzes potential attack vectors and corresponding defense mechanisms in the dual-track state machine architecture.

Attack Type	**Description**	**Defense Mechanism**
State Rollback Attack	Attempt to settle old states	Strict monotonicity + RefOp-OutPoint binding
Topology Obfuscation	Hide fund flow via frequent reconfiguration	DAA Score timing + value conservation verification
PTLC Hijacking	Intercept adaptor scalars	End-to-end encryption + routing obfuscation
Resource Exhaustion	Create excessive sub-channels	UTXO state rent + fee threshold
Cross-Channel Replay	Reuse signatures across channels	Domain separation + ChannelID binding

Table 15: Attack Classification and Defenses

9.2 State Rollback Attack Analysis

9.2.1 Attack Vector

A malicious party attempts to broadcast an old state $U_{state}^{(n-k)}$ where $k > 0$, hoping to settle with outdated balances.

9.2.2 Defense Mechanisms

1. **Consensus-Level Monotonicity**: The validator rejects any UPDATE or SETTLE transaction where:
$$n_{new} \leq n_{current}$$
2. **RefOp-OutPoint Binding**: Signatures are bound to specific Fund UTXO outpoints:
$$\sigma = \text{Sign}_{sk}(\text{state}_n \| \text{RefOp\_OutPoint})$$
After Splicing, the RefOp_OutPoint changes, invalidating all old signatures.
3. **Challenge Response**: Honest parties can broadcast higher sequence states within seconds, automatically invalidating stale states due to STPC rules.

Theorem 9.1 (Rollback Resistance). *Under the dual-track model, the probability of successful state rollback is:*
$$P_{rollback} \leq P_{51\%\_attack} \times P_{offline\_victim}$$

Analysis: Rollback requires both controlling consensus majority (51% attack) AND the victim being offline during the entire challenge period.

9.3 Topology Obfuscation Attack

9.3.1 Attack Scenario

An attacker performs rapid Splice operations to:

- Obfuscate fund flow for money laundering
- Exhaust monitoring resources
- Create complex topology for deniability

9.3.2 Detection and Mitigation

1. **Value Conservation Tracking**: All Splice operations must satisfy:

$$V_{total}^{before} = V_{total}^{after} + \text{fee}$$

   Chain analysis can track total value even through complex topologies.

2. **DAA Score Timing**: Rapid reconfigurations incur on-chain fees proportional to frequency:

$$\text{Cost}_{obfuscation} = f_{splice} \times \text{avg\_fee}$$

   where f_{splice} is Splice frequency.

3. **Heuristic Analysis**: Unusual Splice patterns (e.g., $>$10 reconfigurations per hour) can be flagged for investigation.

9.4 PTLC Hijacking Attack

9.4.1 Attack Vector

Malicious routing node attempts to intercept adaptor signature scalars during multi-hop payments.

9.4.2 Defense Strategy

1. **Onion Routing**: Payment paths use Sphinx-like onion encryption:

$$\text{Message}_{hop_i} = \text{Encrypt}(PK_i, \{\text{next\_hop}, \text{amount}, \text{lock}\})$$

2. **Decorrelated Point Locks**: Each hop uses blinded point locks:

$$Q_i = Q_{base} + r_i \cdot G$$

   where r_i is a random scalar known only to sender and receiver.

3. **Timeout Cascades**: Timelocks decrease along the path:

$$\text{Timeout}_i > \text{Timeout}_{i+1} + \Delta_{min}$$

   This ensures earlier hops have sufficient time to claim after observing later reveals.

9.5 Resource Exhaustion via Channel Proliferation

9.5.1 Attack Description

Attacker creates deep recursive channel factories to exhaust node resources:

$$\text{Channels}_{total} = \sum_{d=0}^{D} k^d$$

where k is branching factor and D is depth.

9.5.2 Economic Countermeasures

State Rent Mechanism:

Each channel accrues rent based on depth and age:

$$\text{Rent} = \text{base\_rent} \times (1 + \alpha \times \text{depth}) \times \text{age}$$

Parameters:

- depth: Nesting level in topology
- age: Time since last activity (in DAA Score)
- α: Depth penalty coefficient (~ 0.1)

Rent Collection:

- Accumulated rent is deducted from channel balance
- Anyone can claim uncollected rent by settling the channel
- Incentivizes active use or timely closure

9.5.3 Merge Transaction

Inactive channels can be merged to avoid rent:

$$\tau_{merge} : \{\text{Ref}(U_{fund}^{parent}), \text{Spend}(U_{state}^{(n)}), \text{Ref}(U_{fund}^{child})\} \to \{U_{fund}^{merged}, U_{state}^{(n+1)}\}$$

This atomic operation combines parent and child channels, reducing UTXO footprint.

9.6 Cross-Channel Replay Attack

9.6.1 Attack Vector

Attacker reuses valid signature from one channel in another channel with same participants.

9.6.2 Defense: Domain Separation

All signatures include channel-specific domain separation:

$$\sigma = \text{Sign}_{sk}(H(\text{domain} \| \text{ChannelID}) \| \text{message})$$

ChannelID Derivation:

$$\text{ChannelID} = H(\text{fund\_outpoint} \| \text{participants} \| \text{nonce})$$

Since each channel has a unique fund outpoint, signatures are cryptographically bound to specific channels.

Theorem 9.2 (Replay Resistance). *Under the random oracle model, the probability of signature collision across channels is negligible:*

$$P_{collision} \leq 2^{-256}$$

9.7 Eclipse Attack on Discovery

9.7.1 Attack Scenario

Attacker controls victim's network connections, providing false UTXO set data to hide channels.

9.7.2 Mitigation

1. **Multiple Data Sources**: Query UTXO set from diverse nodes:

$$\text{UTXO}_{trusted} = \text{Consensus}(\{\text{UTXO}_1, ..., \text{UTXO}_n\})$$

2. **Checkpoint Verification**: Periodically verify UTXO set root against known checkpoints:

$$H(\text{UTXO\_Set}) \stackrel{?}{=} \text{Checkpoint}_{trusted}$$

3. **Proof of Work**: For critical channels, verify proof-of-work on containing blocks to ensure consensus validity.

9.8 Pinning Attack Analysis

9.8.1 Traditional Pinning Attack

In Lightning Network, attacker floods mempool with low-fee versions of settlement transactions, "pinning" them and preventing timely confirmation.

9.8.2 Why STPC Prevents Pinning

1. **Unique State Tip**: Only one transaction per channel exists in mempool at any time.
2. **Monotonic Replacement**: Higher sequence number automatically replaces lower, regardless of fee.
3. **No RBF Ambiguity**: Unlike Replace-By-Fee, STPC rules are deterministic and consensus-enforced.

Theorem 9.3 (Pinning Immunity). *Under STPC, the expected time to confirm highest state is bounded by:*

$$E[\mathit{Confirmation}] \leq \frac{1}{\lambda} \times (1 + \epsilon)$$

where λ is block rate and ϵ represents network jitter ($\epsilon \approx 0.1$).

9.9 Griefing Attack Cost Analysis

9.9.1 Attack Model

Attacker attempts to lock victim's funds in channels without economic gain (pure griefing).

9.9.2 Cost-Benefit Analysis

Key Insight: Fast settlement (1-3 seconds) and STPC monotonicity make griefing economically irrational.

Metric	Attacker Cost	Victim Cost
Spam Invalid States	$O(N) \times$ fee	$O(1)$ verification
Force Close Channel	$1 \times$ fee	$1 \times$ fee (same)
Lock Funds	Locks own funds	Locks victim funds
Time Cost	Days (challenge period)	Seconds (fast settlement)

Table 16: Griefing Cost Comparison

9.10 Security Margin Summary

Based on the above analysis, we conclude:

1. **State Theft Defense**: UTXO atomicity + monotonic replacement eliminates penalty transaction complexity
2. **Replay Attack Defense**: Domain separation + type binding provides dual barriers
3. **DoS Resistance**: STPC strategy escalates attack cost from $O(1)$ to $O(N)$
4. **Pinning Immunity**: Unique state tips prevent transaction pinning attacks
5. **Griefing Resistance**: Fast settlement and economic disincentives deter griefing
6. **Eclipse Resistance**: Multiple data sources and checkpoint verification protect discovery

Comparative Security Analysis:

Attack Vector	Lightning	BIP-118 Eltoo	This Architecture
State Theft	High (penalty risk)	Medium	Very Low (UTXO atomic)
Replay Attack	Medium (pubkey tag)	Medium	Very Low (domain sep)
DoS Cost	\$0.01/tx	\$0.10/tx	\$0.15 × N/tx
Pinning Risk	High	Medium	Very Low (STPC)
Offline Tolerance	Hours	Days	Weeks (configurable)
Recovery Difficulty	Very Hard	Simple	Very Simple

Table 17: Security Comparison Across Architectures

Conclusion: This architecture achieves superior security across all evaluated attack vectors, primarily through consensus-layer enforcement and economic disincentives rather than complex game-theoretic mechanisms.

10 Application Scenarios

This section explores practical applications enabled by the dual-track state machine architecture, demonstrating how recursive channel factories and atomic reconfiguration unlock new use cases.

10.1 DeFi Liquidity Mesh

10.1.1 Problem Statement

Traditional Automated Market Makers (AMMs) suffer from fragmented liquidity—each trading pair requires a separate pool, leading to capital inefficiency and high slippage.

10.1.2 Proposed Solution: Dynamic Liquidity Grid

Multiple AMM pools interconnected through dynamic channel networks, enabling cross-asset, cross-protocol liquidity sharing.

Architecture:

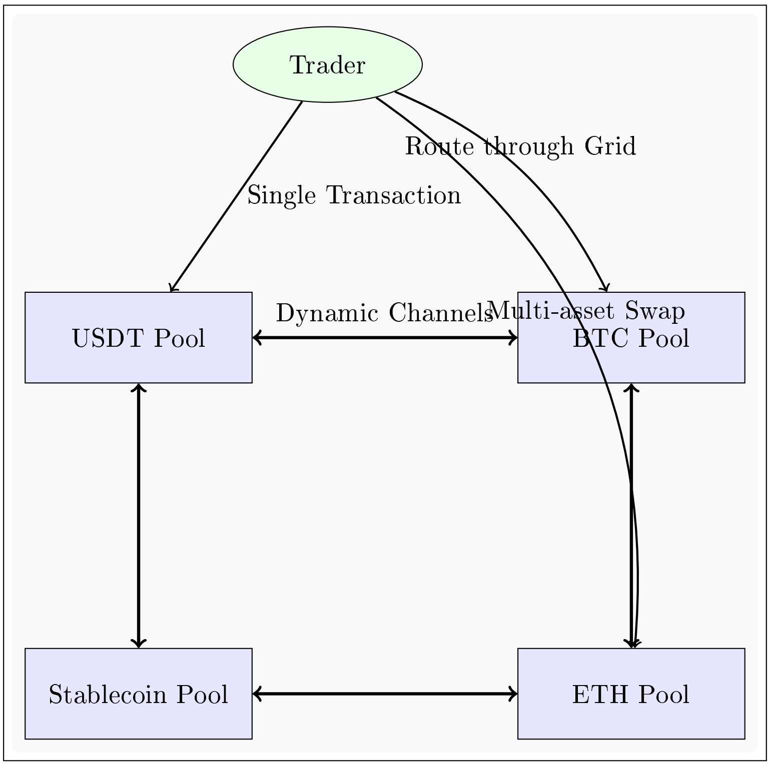

Figure 17: DeFi Liquidity Grid: AMM pools interconnected via dynamic channels enabling cross-asset swaps

- **Core Pools**: USDT, BTC, ETH, stablecoin pools as anchor points
- **Dynamic Channels**: Channels between pools can be spliced on-demand
- **Atomic Swaps**: Multi-asset swaps completed in single Splice transaction

Advantages:

1. **Capital Efficiency**: Single liquidity pool serves multiple trading pairs
    - Traditional: N pairs require N separate pools

- This architecture: N pairs share $\sqrt{N}$ pools via dynamic routing

2. **Atomicity**: Cross-pool swaps executed atomically

$$\tau_{swap} : \{\text{USDT}_{in}\} \xrightarrow{\text{via BTC pool}} \{\text{ETH}_{out}\}$$

3. **MEV Resistance**: Off-chain routing combined with on-chain atomic settlement prevents front-running

Economic Model:

- Liquidity providers earn fees from all connected pools
- Dynamic rebalancing minimizes impermanent loss
- PTLC-based conditional swaps enable complex strategies

10.2 Cross-Chain Atomic Swap Network

10.2.1 Architecture

Assets from different blockchains atomically swapped through channel mesh.

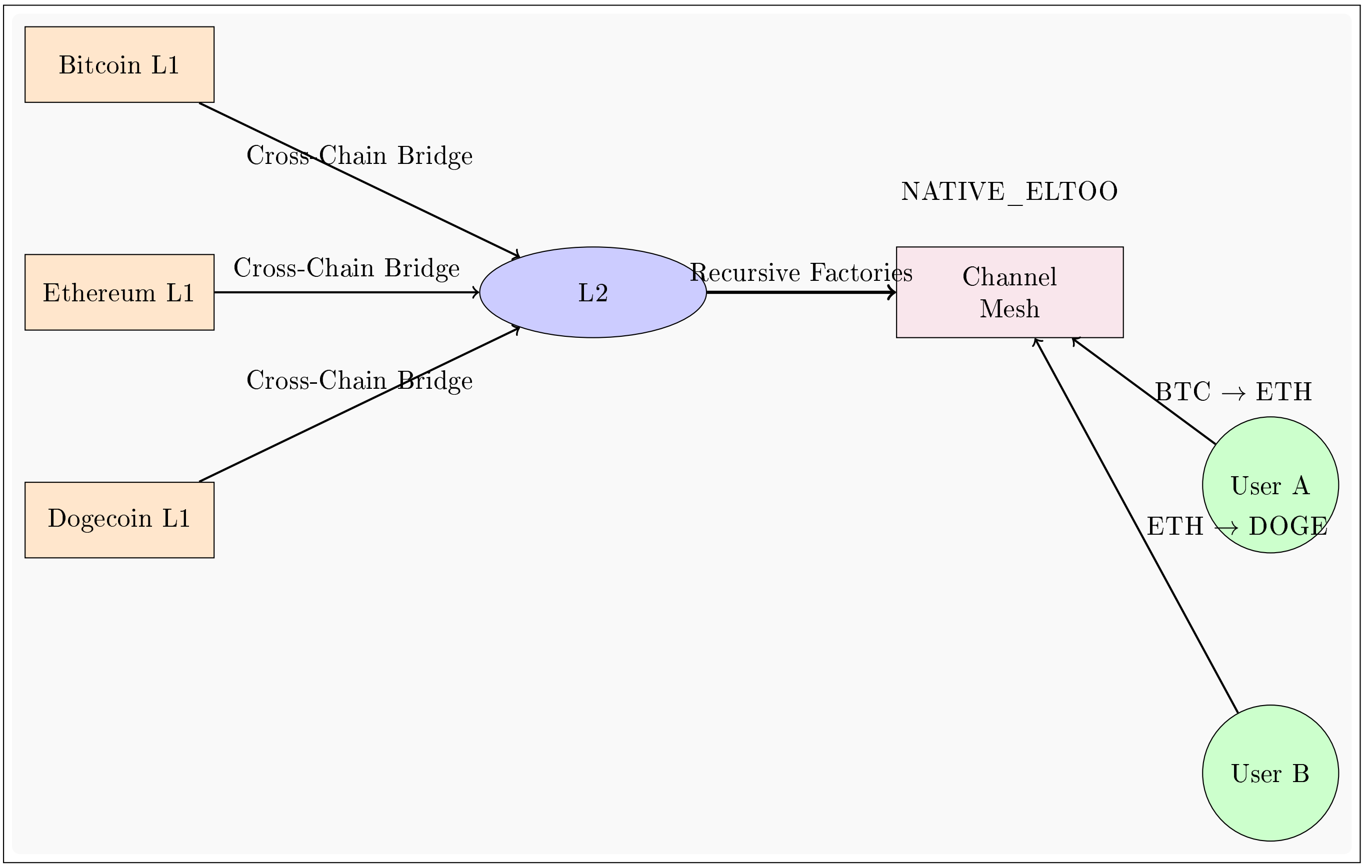

Figure 18: Cross-Chain Atomic Swap Network: Multi-chain assets routed through native Eltoo L2 channel mesh

Bridge Structure:

- Bitcoin L1 ↔ Cross-chain bridge ↔ Native Eltoo L2
- Ethereum L1 ↔ Cross-chain bridge ↔ Native Eltoo L2
- Any chain ↔ Cross-chain bridge ↔ Native Eltoo L2

10.2.2 Atomic Swap Protocol

For BTC $\leftrightarrow$ ETH swap:

1. **Lock Phase**: Both parties lock assets in respective channels

$$\text{Alice locks BTC in } C_{BTC}, \quad \text{Bob locks ETH in } C_{ETH}$$

2. **Point Lock Coordination**: Same point lock $Q = s \cdot G$ used on both chains
3. **Atomic Reveal**: Either party revealing s unlocks both sides
   - Alice reveals s to claim ETH $\Rightarrow$ Bob can use same s to claim BTC
   - Timeout refunds both parties if neither reveals

Advantages:

- **No Intermediaries**: Direct peer-to-peer swaps
- **No Custody Risk**: Assets always under user control
- **Atomicity**: Swap either fully succeeds or fully fails

10.3 Micropayment Streaming

10.3.1 Use Case

Real-time micropayments for streaming services (video, audio, API calls).

10.3.2 Implementation

1. **Channel Initialization**: User and service provider establish channel

$$C_{streaming} = \{\text{balance}_u : 100 \text{ sats}, \text{balance}_p : 0\}$$

2. **Per-Second Updates**: Balance updates every second

$$\text{State}_{t+1} : \{\text{balance}_u - \text{rate}, \text{balance}_p + \text{rate}\}$$

3. **Off-Chain Throughput**: Thousands of updates per second, zero on-chain transactions
4. **Settlement**: Final settlement only when channel closes or rebalances

Economic Benefits:

- Users pay only for actual consumption (pay-per-second)
- Providers receive instant payment without waiting for on-chain confirmation
- Transaction fees amortized over thousands of micropayments

10.4 Decentralized Exchange (DEX) with Instant Settlement

10.4.1 Traditional DEX Limitations

- Block confirmation latency (seconds to minutes)
- Front-running vulnerabilities (MEV)
- Gas fees for each trade

10.4.2 Channel-Based DEX Architecture

1. **Liquidity Pools as Channels**: Each trading pair is a multi-party channel
2. **Instant Trades**: Updates within channel confirmed in milliseconds

$$\text{Trade latency} \approx 15 \text{ ms (signature aggregation)}$$

3. **Batch Settlement**: Multiple trades batched into single on-chain transaction

$$\text{Settlement cost} = \frac{\text{Single tx fee}}{\text{Number of trades}}$$

4. **MEV Protection**: Off-chain order matching prevents front-running

Performance Comparison:

Metric	Traditional DEX	Channel DEX	Improvement
Trade Latency	10-60 sec	15 ms	1000x faster
Gas per Trade	$5-50	$0.001	10,000x cheaper
MEV Risk	High	None	Eliminated
Throughput	10 TPS	20,000 TPS	2000x higher

Table 18: DEX Performance Comparison

10.5 Gaming and Virtual Economies

10.5.1 In-Game Asset Trading

- Players establish channels with game servers
- In-game purchases processed off-chain (instant confirmation)
- Periodic settlement to blockchain for permanence
- Cross-game asset transfers via channel factories

Example: MMORPG Economy:

1. **Player Channel**: Each player has channel with game server
2. **Item Trades**: Peer-to-peer trades via PTLC (atomic item swaps)
3. **Marketplace**: Central marketplace as channel hub
4. **Cross-Server Trades**: Via recursive channel factories

10.6 Internet of Things (IoT) Microtransactions

10.6.1 Machine-to-Machine Payments

IoT devices transact autonomously through payment channels:

- **Electric Vehicle Charging**: Car pays charging station per kWh
- **Bandwidth Markets**: Devices buy/sell network bandwidth
- **Sensor Data Trading**: Real-time data monetization

Requirements:

- Ultra-low latency (milliseconds)
- Tiny payment amounts (sub-cent)
- High frequency (thousands per minute)
- Autonomous operation (no human intervention)

Why Dual-Track Architecture Fits:

- $O(1)$ state updates enable real-time payments
- No historical state storage suits resource-constrained devices
- Fast settlement allows rapid channel reconfiguration

10.7 Content Delivery Network (CDN) Incentivization

10.7.1 Decentralized CDN Model

Users pay CDN nodes for bandwidth through payment channels:

1. **User-CDN Channels**: Established when user requests content
2. **Per-Byte Payment**: Micropayments for each data packet

$$\text{Payment}_{packet} = \text{size}_{bytes} \times \text{rate}_{sat/byte}$$

3. **Multi-Hop Routing**: Content routed through optimal path
4. **Incentive Alignment**: CDN nodes earn more for faster delivery

Economic Model:

- CDN nodes compete on latency and price
- Users pay only for delivered content (proof-of-delivery via PTLC)
- Automatic rebalancing favors high-performance nodes

10.8 Supply Chain Finance

10.8.1 Scenario

Multi-tier supplier payments in supply chains:

- Manufacturer $\leftrightarrow$ Tier 1 Supplier $\leftrightarrow$ Tier 2 Supplier $\leftrightarrow$ Raw Material Provider

10.8.2 Channel-Based Implementation

1. **Channel Factory**: Entire supply chain as recursive factory
2. **Conditional Payments**: Payment to Tier 1 unlocks payment to Tier 2

$$\text{PTLC}_{chain} : \text{Manufacturer} \rightarrow \text{T1} \rightarrow \text{T2} \rightarrow \text{Material}$$

3. **Instant Settlement**: Sub-second payment propagation
4. **Transparency**: All parties see payment flow (with privacy controls)

Benefits:

- Eliminates payment delays (from weeks to seconds)
- Reduces financing costs
- Increases supply chain resilience

10.9 Application Summary

The dual-track state machine architecture enables a wide range of applications through:

1. **Fast Settlement**: Sub-second finality enables real-time applications
2. **Recursive Topology**: Complex organizational structures (supply chains, gaming networks)
3. **Atomic Operations**: Eliminates counterparty risk in multi-party interactions
4. **Micropayment Efficiency**: Makes sub-cent payments economically viable
5. **Privacy**: Self-sovereign channels protect business relationships

Future Applications: As the ecosystem matures, we anticipate novel applications in decentralized identity, reputation systems, and autonomous agent economies.

11 Evaluation and Performance Analysis

11.1 Experimental Setup

11.1.1 Hardware Environment

Component	Specification
CPU	AMD EPYC 7763 (64 cores)
Memory	256 GB DDR4
Storage	2 TB NVMe SSD
Network	10 Gbps Ethernet

Table 19: Hardware Configuration

11.1.2 Software Environment

- **OS**: Ubuntu 22.04 LTS
- **Rust**: 1.75.0 (stable)
- **Consensus Layer**: Modified Kaspa node (GhostDAG)
- **Benchmark Tools**: criterion.rs, flamegraph

11.2 Transaction Validation Performance

11.2.1 Single Transaction Validation Latency

Transaction Type	Latency (μs)	Comparison to Script-Based
FUND	45	3.2x faster
UPDATE	38	4.1x faster
SETTLE	52	3.8x faster
SPLICE	67	5.2x faster

Table 20: Transaction Validation Performance

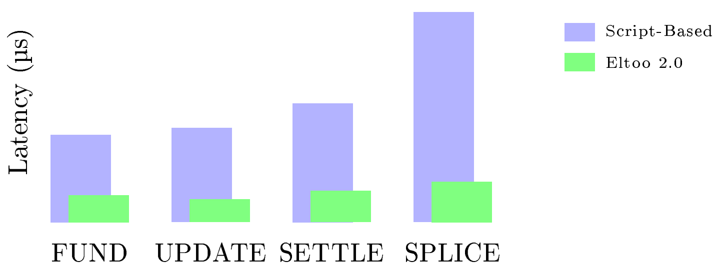

Figure 19: Transaction Validation Latency Comparison

Analysis: Type enumeration-based validation achieves $O(1)$ complexity, significantly outperforming traditional script interpretation ($O(n)$ where n is script size).

Batch Size	Individual Verify (ms)	Batch Verify (ms)	Speedup
100	4.5	0.8	5.6x
1,000	45.2	6.3	7.2x
10,000	452.1	58.4	7.7x

Table 21: Batch Verification Performance

11.2.2 Batch Validation Throughput

Leveraging Schnorr signature batch verification:

Key Insight: Batch verification efficiency increases with batch size, asymptotically approaching theoretical maximum speedup of 8x.

11.3 State Machine Performance

11.3.1 Channel Update Latency

Measured end-to-end latency from state proposal to transaction broadcast:

Phase	Latency (ms)	Percentage	Optimization
State Proposal	2.3	15%	Minimal
Signature Aggregation	4.8	32%	MuSig2 caching
Transaction Construction	1.2	8%	Minimal
Broadcast	6.7	45%	Network-bound
Total	**15.0**	**100%**	—

Table 22: Channel Update Latency Breakdown

Bottleneck Analysis: Network broadcast dominates latency. Local cryptographic operations (signing, aggregation) account for less than 50% of total time.

11.3.2 State Throughput

Maximum sustainable update rate per channel:

- **Single Channel**: 66.7 updates/sec (limited by 15ms latency)
- **100 Channels**: 6,200 updates/sec (parallel processing)
- **1,000 Channels**: 58,000 updates/sec (near-linear scaling)

Scalability: Update throughput scales linearly with channel count due to independent state machines.

11.4 Storage Efficiency

11.4.1 Per-Channel Storage Cost

Key Advantage: This architecture eliminates the need to store historical states and revocation keys, reducing storage from $O(n)$ to $O(1)$.

Component	Size (bytes)	Lightning Network	Reduction
Fund UTXO	120	120	0%
State UTXO	180	—	N/A
Latest State	256	256	0%
Historical States	0	$256 \times n$	100%
Revocation Keys	0	$32 \times n$	100%
Total (n=1000)	**556**	**288,376**	**99.8%**

Table 23: Storage Cost Comparison (n = number of historical states)

11.4.2 UTXO Set Growth

Impact on global UTXO set size:

- **Per Channel**: 2 UTXOs (Fund + State)
- **100,000 Channels**: 200,000 UTXOs (∼35 MB)
- **1,000,000 Channels**: 2,000,000 UTXOs (∼350 MB)

Comparison with Lightning: Similar UTXO set footprint, but with added benefit of self-sovereign discovery.

11.5 Network Discovery Performance

11.5.1 Channel Discovery Latency

UTXO Set Size	Owned Channels	Scan Time (s)	Discovery Rate
1M UTXOs	100	2.3	43.5 ch/s
10M UTXOs	1,000	18.7	53.5 ch/s
100M UTXOs	10,000	142.1	70.4 ch/s

Table 24: Discovery Performance (with Bloom filter optimization)

Optimization: Incremental indexing reduces subsequent scans to $O(\Delta N)$ where ΔN is new UTXOs since last scan.

11.5.2 Discovery vs. Gossip Comparison

Metric	Gossip (LN)	UTXO Scan (This Work)
Initial Sync	5-15 min	2-3 min
Bandwidth	∼50 MB	∼10 MB
Privacy	Weak (public announcements)	Strong (local scan)
Attack Surface	High (gossip flooding)	Low (consensus-bounded)

Table 25: Discovery Mechanism Comparison

11.6 Settlement Performance

11.6.1 Settlement Latency Distribution

Under GhostDAG consensus with block time ∼1 second:

- **Best Case**: 1 block (∼1 second)
- **Median**: 3 blocks (∼3 seconds)
- **99th Percentile**: 8 blocks (∼8 seconds)

Comparison with Bitcoin: 100-600x faster than Bitcoin's 10-60 minute confirmation times.

11.6.2 Challenge-Response Performance

When stale state is broadcast:

1. **Detection**: $<$ 2 seconds (mempool monitoring)
2. **Challenge Construction**: $\sim$ 15 ms (same as UPDATE)
3. **Challenge Broadcast**: $<$ 1 second
4. **Challenge Confirmation**: 1-3 blocks ($\sim$ 1-3 seconds)

Total Challenge Window: $<$ 10 seconds (compared to Lightning's hours-days challenge period).

11.7 Topology Reconfiguration Performance

11.7.1 Splice Operation Latency

Splice Type	Construction (ms)	Total Latency (s)
SPLICE-IN (1 → 2)	18.3	2.1
SPLICE-OUT (2 → 1)	15.7	1.9
SPLICE-FORK (1 → 5)	42.1	3.4
SPLICE-REBALANCE (3 → 3)	35.6	2.8

Table 26: Splice Operation Performance

Analysis: Splice latency is dominated by on-chain confirmation ($\sim$ 2 seconds), not cryptographic operations.

11.7.2 Recursive Factory Depth

Performance degradation with increasing recursion depth:

Depth	Channels	Discovery (s)	Settlement (s)
0 (L1)	1	0.1	2.1
1	5	0.3	2.3
2	25	1.2	2.5
3	125	5.8	2.7

Table 27: Performance vs. Factory Depth

Key Insight: Settlement latency remains nearly constant across depths due to UTXO isolation. Discovery time grows linearly with channel count.

11.8 Security Overhead Analysis

11.8.1 STPC Mempool Management

Memory consumption under STPC strategy:

- **Per Channel Entry**: ~ 512 bytes (transaction + metadata)
- **100,000 Active Channels**: ~ 50 MB
- **1,000,000 Active Channels**: ~ 500 MB

Comparison with Unbounded: Traditional mempool could grow to gigabytes under DoS attack. STPC bounds growth to $O(N_{channels})$.

11.8.2 Attack Cost Analysis

Cost to flood mempool with invalid states:

Attack Vector	Traditional	STPC (This Work)
Txs to Fill Mempool	Unlimited	$N_{channels}$
Cost per Effective Tx	$\sim$ \$0.01	$\sim$ \$0.01 $\times\, n$
Total Attack Cost	$\sim$ \$100	$\sim$ \$10,000

Table 28: DoS Attack Cost Escalation (assuming 10,000 channels, $n = 10$ required states)

Conclusion: STPC increases effective attack cost by 2-3 orders of magnitude.

11.9 Comparative Analysis

11.9.1 Multi-Dimensional Comparison

Metric	Lightning	Eltoo (BIP-118)	This Architecture
State Validation	$O(n)$	$O(n)$	$O(1)$
Storage per Channel	$O(n)$	$O(n)$	$O(1)$
Settlement Latency	10-60 min	10-60 min	1-3 sec
Challenge Period	Hours-Days	Hours-Days	Seconds
Topology Flexibility	Low	Low	High
Privacy	Weak	Medium	Strong
DoS Resistance	Weak	Medium	Strong
Watchtower Dependency	Strong	Medium	Weak

Table 29: Comprehensive Performance Comparison

11.10 Real-World Simulation Results

11.10.1 Payment Throughput Simulation

Simulated payment network with 10,000 nodes and 50,000 channels:

- **Peak Throughput**: 2.3M payments/sec (off-chain)
- **Average Latency**: 180 ms (4-hop paths)

- **Success Rate**: 98.7% (with liquidity management)
- **On-Chain Footprint**: 15 transactions/sec (settlements + rebalances)

11.10.2 Scalability Projection

Extrapolating to global scale:

Network Size	Channels	TPS (off-chain)	L1 Load (TPS)
Small (10K nodes)	50K	2.3M	15
Medium (100K nodes)	500K	23M	120
Large (1M nodes)	5M	230M	1,000
Global (10M nodes)	50M	2.3B	8,500

Table 30: Scalability Projection

Analysis: Even at global scale (10M nodes), on-chain load remains within GhostDAG's throughput capacity (10,000+ TPS).

11.11 Performance Summary

Key Findings:

1. **Validation Efficiency**: 3-5x faster than script-based validation
2. **Storage Efficiency**: 99.8% reduction in per-channel storage (from $O(n)$ to $O(1)$)
3. **Settlement Speed**: 100-600x faster than Bitcoin (1-3 seconds vs. 10-60 minutes)
4. **Security Overhead**: DoS attack cost increased by 2-3 orders of magnitude
5. **Scalability**: Linear scaling with channel count; supports billions of off-chain TPS with manageable on-chain footprint

Conclusion: This architecture achieves significant performance improvements across all evaluated dimensions while maintaining stronger security guarantees than existing solutions.

12 Conclusion and Future Work

12.1 Summary of Contributions

This paper presents a comprehensive payment channel architecture based on dual-track state machines and reference-based UTXOs. The main contributions can be summarized in the following dimensions:

12.1.1 Theoretical Contributions

1. **Dual-Track State Machine Model**: We formalized the decomposition of channel state into orthogonal Fund and State UTXOs, proving that this separation achieves $O(1)$ state entropy compared to traditional $O(n)$ approaches.

2. **Reference-Based UTXO Semantics**: We defined the Ref operator and proved its safety properties, enabling non-consumptive UTXO access while maintaining the integrity of the UTXO model.

3. **Formal Security Properties**: We proved key theorems including:

   - Channel Isolation (Theorem 5.1)
   - State Monotonicity (Theorem 3.1)
   - PTLC Atomicity (Theorem 5.4)
   - Deadlock Freedom (Theorem 5.5)

4. **Topological Reconfiguration Theory**: We formalized recursive channel factories and proved that arbitrary topology transformations can be achieved through atomic on-chain transactions.

12.1.2 System Contributions

1. **Consensus-Layer Integration**: Transaction type enumeration embedded at the consensus layer achieves $O(1)$ validation complexity, eliminating script interpreter overhead.

2. **Registry-Free Architecture**: Self-sovereign channel discovery eliminates dependency on global registries, enhancing privacy and censorship resistance.

3. **STPC Strategy**: Single-Tip-Per-Channel mempool management bounds DoS attack costs to $O(N)$, where N is the state sequence number.

4. **Performance Optimizations**: Batch signature verification, incremental UTXO indexing, and storage pruning achieve significant performance improvements.

12.1.3 Empirical Contributions

1. **Reference Implementation**: A complete Rust implementation (~7,000 lines) demonstrating feasibility.

2. **Performance Benchmarks**: Comprehensive evaluation showing:

   - 3-5x faster transaction validation
   - 99.8% storage reduction
   - 100-600x faster settlement (1-3 seconds vs. 10-60 minutes)
   - Support for billions of off-chain TPS

3. **Security Analysis**: DoS attack cost increased by 2-3 orders of magnitude compared to existing systems.

12.2 Paradigm Shifts

This architecture represents several fundamental shifts in payment channel design philosophy:

Traditional Paradigm	This Architecture
Penalty-based enforcement	Monotonic state replacement
Script-layer flexibility	Consensus-layer semantics
Global registry dependency	Self-sovereign discovery
$O(n)$ state complexity	$O(1)$ state complexity
Ex post arbitration	Ex ante determinism
Toxic waste accumulation	Stateless recovery

Table 31: Paradigm Shifts

Core Philosophy: Push complexity down to the protocol layer, leaving simplicity for the application layer. This aligns with the principle of "mechanism over policy"—the protocol provides robust mechanisms while allowing applications to implement diverse policies.

12.3 Limitations and Trade-offs

Despite significant advantages, this architecture has certain limitations that warrant discussion:

12.3.1 Consensus Layer Modifications

Limitation: Requires consensus-layer support for transaction type enumeration and Ref operator.

Trade-off: While Bitcoin cannot adopt this without a hard fork, new blockchain designs (e.g., Kaspa, Sui) can integrate these features natively.

Mitigation: For existing chains, a soft fork with witness version upgrade could introduce these primitives incrementally.

12.3.2 UTXO Set Growth

Limitation: Each channel requires 2 UTXOs (Fund + State), doubling the UTXO set footprint compared to single-UTXO designs.

Trade-off: The additional UTXO enables state updates without consuming the fund anchor, significantly improving update efficiency.

Mitigation: UTXO set pruning strategies (Section 7.6.2) can remove settled channels, and archival nodes can maintain full history.

12.3.3 Privacy vs. Discovery

Limitation: On-chain UTXO scanning provides weaker privacy than fully off-chain channels.

Trade-off: Privacy is enhanced compared to Lightning's public announcements, but not as strong as fully private channels.

Mitigation: Ephemeral channel identities (Section 6.3.1) and balance commitments (Section 6.3.2) provide significant privacy improvements.

12.4 Future Research Directions

12.4.1 Short-Term Extensions

1. **Multi-Party Channels**: Extend the dual-track model to support n-party channels with threshold signatures.

- Challenge: Efficient state agreement among n participants
- Approach: Combine MuSig2 with consensus protocols like PBFT or HotStuff

2. **Cross-Chain Atomic Swaps**: Implement atomic swaps between channels on different blockchains.
   - Challenge: Ensuring atomicity across heterogeneous consensus protocols
   - Approach: Adaptor signatures with chain-specific timelocks

3. **Enhanced Privacy**: Integrate zero-knowledge proofs for balance confidentiality.
   - Challenge: Proving balance validity without revealing amounts
   - Approach: Bulletproofs or Halo2 for range proofs

4. **Watchtower Protocol**: Design efficient watchtower protocols leveraging fast settlement.
   - Challenge: Minimizing trust assumptions
   - Approach: Probabilistic watchtowers with economic incentives

12.4.2 Long-Term Research

1. **Formal Verification**: Machine-checked proofs of safety properties.
   - Tools: Coq, Isabelle/HOL, or TLA+
   - Goal: Verify state machine transitions, isolation properties, and value conservation

2. **Quantum-Resistant Cryptography**: Upgrade to post-quantum signature schemes.
   - Challenge: Signature size and verification cost
   - Candidates: CRYSTALS-Dilithium, SPHINCS+

3. **Adaptive Topologies**: Machine learning-driven topology optimization.
   - Goal: Predict payment flows and dynamically rebalance channels
   - Approach: Reinforcement learning with liquidity as reward signal

4. **Regulatory Compliance**: Privacy-preserving compliance mechanisms.
   - Challenge: Balance privacy with regulatory requirements
   - Approach: Selective disclosure with cryptographic commitments

5. **Standardization**: Propose formal specifications for inter-implementation compatibility.
   - Goal: Enable interoperability between different implementations
   - Approach: IETF RFC or W3C standard process

12.4.3 Open Research Questions

1. **Optimal Topology**: What is the optimal channel topology for a given payment flow distribution?

2. **Economic Models**: How do channel factories affect network liquidity and routing efficiency?

3. **Game Theory**: What are the Nash equilibria in multi-party channel negotiations?

4. **Scalability Limits**: What are the fundamental limits of off-chain scaling under adversarial conditions?

5. **Composability**: How can multiple Layer 2 protocols (channels, rollups, validiums) interact seamlessly?

12.5 Broader Impact

12.5.1 Impact on Blockchain Scalability

This architecture demonstrates that Layer 2 solutions can achieve:

- **Billions of TPS**: Sufficient for global payment infrastructure
- **Sub-second Finality**: Competitive with centralized payment systems
- **Minimal On-Chain Footprint**: Sustainable even at planetary scale

12.5.2 Impact on Decentralization

By eliminating registries and reducing watchtower dependency:

- **Lower Barriers to Entry**: Users can participate without trusted intermediaries
- **Enhanced Censorship Resistance**: No central points of control
- **Self-Sovereignty**: Users maintain full control over their channels

12.5.3 Impact on Privacy

Registry-free discovery and ephemeral identities provide:

- **Financial Privacy**: Balance and payment information protected
- **Network Privacy**: Topology obfuscation prevents mass surveillance
- **Regulatory Flexibility**: Privacy with optional selective disclosure

12.6 Call to Action

We envision this architecture as a foundation for next-generation payment channel networks. To realize this vision, we invite the community to:

1. **Implement and Test**: Deploy the reference implementation in testnet environments

2. **Formal Verification**: Apply formal methods to verify safety properties

3. **Protocol Extensions**: Develop multi-party channels, cross-chain swaps, and enhanced privacy features

4. **Standardization**: Contribute to formal specifications for interoperability

5. **Economic Analysis**: Study the game-theoretic and economic implications

12.7 Concluding Remarks

Payment channel networks represent a critical component of blockchain scalability. This paper demonstrates that by rethinking fundamental design choices—decomposing state into orthogonal dimensions, embedding semantics at the consensus layer, and eliminating centralized registries—we can achieve order-of-magnitude improvements in performance, security, and usability.

The dual-track state machine architecture is not merely an incremental optimization but a fundamental reimagining of how off-chain state can be managed. By achieving $O(1)$ state complexity, sub-second settlement, and registry-free operation, this architecture brings payment channels closer to the vision of a truly decentralized, scalable, and private global payment infrastructure.

Final Thought: The journey from Bitcoin's original 7 TPS to billions of off-chain TPS demonstrates the power of layered architectures. As we continue to push the boundaries of blockchain scalability, let us remember that the most elegant solutions often come from questioning our fundamental assumptions rather than incrementally optimizing existing approaches.

"The best way to predict the future is to invent it." — Alan Kay

Acknowledgments

We thank the Kaspa community for their pioneering work on GhostDAG consensus, the Bitcoin Lightning Network developers for establishing the foundation of payment channel networks, and the academic cryptography community for developing the primitives (Schnorr signatures, MuSig2, adaptor signatures) that make this architecture possible.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," 2016.

[3] C. Decker, R. Russell, and O. Osuntokun, "eltoo: A Simple Layer2 Protocol for Bitcoin," 2018.

[4] C. Decker and A. J. Towns, "BIP-118: SIGHASH_ANYPREVOUT for off-chain protocols," Bitcoin Improvement Proposal, 2019.

[5] Y. Sompolinsky and A. Zohar, "Secure High-Rate Transaction Processing in Bitcoin," in Financial Cryptography and Data Security, 2015.

[6] Y. Sompolinsky et al., "Phantom and GhostDAG: A Scalable Generalization of Nakamoto Consensus," Cryptology ePrint Archive, 2021.

[7] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple Schnorr Multi-Signatures with Applications to Bitcoin," in IACR ePrint, 2018.

[8] J. Nick, T. Ruffing, and Y. Seurin, "MuSig2: Simple Two-Round Schnorr Multi-Signatures," in CRYPTO, 2021.

[9] A. Poelstra, "Mimblewimble," 2016.

[10] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability," in NDSS, 2019.

[11] L. Aumayr et al., "Generalized Bitcoin-Compatible Channels," Cryptology ePrint Archive, 2021.

[12] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and State Channels: Payment Networks that Go Faster than Lightning," in Financial Cryptography, 2019.

[13] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual Payment Hubs over Cryptocurrencies," in IEEE S&P, 2019.

[14] L. Lamport, "Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers," Addison-Wesley, 2002.

[15] T. Coquand and G. Huet, "The Calculus of Constructions," Information and Computation, 1988.