# Towards Secure and Efficient Route Computation for Cross-Chain Message Delivery

by

Amin Rezaei

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2026

## Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

Demand for blockchain applications has led to a surge of new public blockchains. However, this fragments liquidity and pushes users to bridge across unfamiliar protocols, increasing risk and complexity. Cross-chain communication enables interoperability, allowing contracts to execute logic and move assets across chains. Yet current delivery solutions either support message passing only between directly connected chains, limiting connectivity, or are centralized and route through a single hub chain that introduces a single point of failure and requires trust in the hub operator.

Inter-blockchain communication can become more robust by leveraging concepts from traditional network architectures, including routing, name resolution, and policy-based message delivery. These mechanisms can increase connectivity by enabling chains that are not directly connected to communicate securely over multi-hop routes.

This thesis studies the problem of **policy-driven cross-chain routing**: Current cross-chain routing is largely ad-hoc and manual, and does not reliably respect users' security or cost preferences when no direct connection exists. Given a dynamic inter-chain topology and user policies (e.g., security thresholds, fee budgets, latency targets), we compute routes over **multi-hop Inter-Blockchain Communication (IBC)** while ensuring (a) security constraints are strictly enforced on-chain and (b) preference constraints (e.g., minimizing gas costs) are met with practical guarantees. This is challenging because the required inputs (e.g., fees, validator sets, congestion, and application-specific state) change independently on each chain, yet the resulting route and its policy compliance must be verifiable on the destination chain at a reasonable cost. We present a modular stack: a Transport Layer with Policy Enforcement Module, a Relayer Control Plane for route computation, and a Relayer Data Plane for execution, which separates concerns between policy specification, route computation, and delivery.

We introduce three routing methods: (1) Single-Relayer routing, which computes routes off-chain independently by off-chain relayer nodes, (2) zkRouter, which computes routes off-chain with a succinct zero-knowledge proof of policy compliance and (3) Relayer Network, a new collaborative overlay that distributes operational load (client updates, packet relaying) across relayers. Our prototypes demonstrate that our stack is practical and achieves higher decentralization, better connectivity, and greater scalability, enabling richer and safer cross-chain applications while preserving IBC's security assumptions and without significant fee overhead. Our evaluation shows: (1) near 90% connectivity vs. 15% for hub-and-spoke; (2) more than 30% connectivity after removing top four chains, reaching 50% with topology upgrades; (3) less than \$0.10 on-chain cost per message; (4) scales to more than $10^6$ messages maintaining low processing time.

**Acknowledgments**

I am grateful to my supervisor, Prof. Bernard Wong, for his guidance and support throughout my master's program and during this research. Working with him has been a great experience and has taught me a lot.

I also thank Solomon Levi Davidson, with whom I worked on Baton, the system this thesis builds upon. I am grateful for the feedback and discussions I received from the Shoshin group over the course of writing this thesis.

Finally, I thank my parents for their constant love and support, especially during the years I was away from them. I also thank my partner, Fatemeh Shafiei Ardestani, for her love and encouragement. I could not have asked for a better partner.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Blockchain adoption has grown at a staggering rate over the past decade. What started as a way of securely transferring assets between end users without relying on banks, exchanges, or other trusted intermediaries, has since expanded into a rich ecosystem spanning thousands of independent chains, with many supporting sophisticated financial transactions. This change to a multi-chain ecosystem was motivated in part by the conflicting requirements of different blockchain applications that cannot be supported by a single chain. Having multiple chains also alleviates congestion created by the increasing demands to transact on blockchains by users and institutions as each chain can transact independently.

With the emergence of a multi-chain ecosystem, providing interoperability across chains has become increasingly important. Currently, the most commonly used interoperability solution is to bridge assets, such as ERC-20 tokens, across chains by locking up assets on the source chain using a smart contract, and then minting a representation of the locked asset on the destination chain. According to DeFiLlama [23], as of January 2026, more than $45 billion USD worth of tokens are currently locked on cross-chain bridges.

However, securing blockchain bridges has proven to be challenging as more than $2.8 billion has been stolen from bridges [5]. Many of the exploits used social engineering techniques to acquire the small number of keys needed to take control of the bridge's smart contract. Furthermore, most bridges only provide interoperability between two chains. Bridges between smaller chains often do not exist, or have questionable reputation. As a result, users of smaller chains often need to bridge their assets more than once across low-reputation bridges to make their assets available at their desired destination, which significantly increases their risk exposure. In practice, most bridges rely on smart contracts that are controlled by a small committee of operators, requiring users to trust this group,

which undermines decentralization and introduces a strong new trust assumption.

The Inter-Blockchain Communication (IBC) protocol was designed to avoid some of the security issues that are present in committee-based bridges. IBC enables two chains to communicate through light clients that track the block headers and verify the consensus states of counterparty chains. Once the respective block headers are verified, transaction inclusion can be verified via Merkle proofs. This provides a comparatively secure approach since it offers equivalent security to that offered by the other chain, assuming proper bootstrapping of clients.

Unfortunately, out of the chains that support IBC, most only have IBC connections to a few popular chains. This limitation is driven in part by the high administrative and financial overhead to create and maintain IBC connections. Users who wish to communicate between chains where an IBC connection does not exist must find their own multi-chain route and send a series of messages that must interact with wallets or smart contracts on the intermediate chains. Not only is this approach challenging to undertake, it can also be problematic from a security perspective as the user may unknowingly choose to route through a malicious chain that can compromise the message. Furthermore, with limited information about the gas costs on different chains, the user may end up choosing a high-fee route.

Axelar [11] is a cross-chain messaging solution that addresses this issue by relaying all cross-chain messages through its own chain to create a hub-and-spoke topology. Although this provides connectivity between all chains connected to the hub chain, it introduces a significant new trust assumption. Furthermore, the hub chain can also limit cross-chain messaging throughput, which can in turn increase message-delivery costs as transaction fees increase with demand.

In many ways, cross-chain communication today is facing problems very similar to those that the Internet faced and solved for inter-network communication. This includes the need for decentralized route computation that meets application-specific security requirements, efficient delivery of messages without introducing new trust assumptions, and secure name resolution to help users identify remote services. At the same time, many of these problems have unique blockchain-specific assumptions and requirements that demand solutions that differ from existing networking solutions. Solving these cross-chain communication problems may enable blockchains to experience the same transformative growth and acceptance by the general public that the Internet had experienced.

The Baton framework provides decentralized multi-hop message routing and delivery for IBC-enabled blockchains. Baton enables users and applications to send packets to a destination chain even when no direct IBC connection exists, by routing over intermediate

chains while enforcing user-specified policies. At a high level, a sender specifies a destination and policy, relayers compute and execute a multi-hop route, and the destination chain verifies required security conditions before accepting the packet. Baton is organized into four layers: application services, transport (policy enforcement), a relayer control plane (route computation), and a relayer data plane (execution). **This thesis focuses on the control plane and its integration with policy and execution layers.** Specifically, we address: how routes are computed under user-specified policies, what network state is trusted during computation, and how destination chains verify packets before acceptance. We propose and evaluate three route computation designs: (1) off-chain router, (2) a zero-knowledge proof-based router (zkRouter), and (3) a collaborative relayer network. We analyze their trade-offs in security assumptions, cost, latency, and policy enforceability.

A **policy** specifies conditions that a route must satisfy (e.g., every intermediate hop must have Nakamoto coefficient $\geq 8$). We distinguish:

- **Security policies** (strict): must hold for packet acceptance; enforced deterministically on-chain via IBC verification plus any required auxiliary data (e.g., an intermediate chain's validator set, verified against the light client state, to compute Nakamoto coefficient).
- **Preference policies** (soft): should hold for best user experience (e.g., fee minimization); enforcement depends on the routing method.

All methods in this thesis enforce security policies on-chain, but provide different guarantees for preference policies; in particular, zkRouter can provide cryptographic guarantees for preference-policy compliance.

The route computation problem presents several distinct challenges that this thesis explores:

1. **Security:** A route computation method involves different trust assumptions (e.g., routing through a hub chain, letting a single relayer select a path subject to on-chain verification of security policies, or generating a zero-knowledge proof of policy compliance) , and since users are performing financial transactions, it is important to choose a method with appropriate security and low trust assumptions.
2. **Cost-Efficiency:** Operations on blockchain are all covered by gas fees. A route-computation method should be practical and affordable. This motivates designs where route computation is off-chain, but the destination still performs lightweight on-chain verification of security policies.

3. **Latency:** Financial operations on blockchains are usually time-sensitive (e.g., prices can change within minutes, invalidating quotes) and significant delays can render transactions irrelevant or failed and incur losses. Route computation must be fast enough to avoid this.

4. **Customizability:** It's important that routes are generated with regard to available information (e.g., gas fees, congestion, and application-specific state such as DEX liquidity) . While information about network topology is enough for finding routes between source and destination, richer information can enable new opportunities (e.g., price, reliability).

To address these challenges, this thesis proposes and evaluates three distinct architectures for route computation: an off-chain router, a zero-knowledge proof-based router (zkRouter), and a collaborative relayer network. By implementing these distinct approaches, we provide a comprehensive analysis of the trade-offs between security assumptions, cost, latency, and policy enforceability. This thesis makes the following contributions:

1. **Three route computation designs.** We design and implement three route computation designs: an off-chain router, a zero-knowledge proof-based router (zkRouter), and a collaborative relayer network, and integrate them with Baton's policy and execution layers.

2. **Implementation of a collaborative relayer network** that enables efficient multi-hop packet delivery with cryptoeconomic security guarantees.

3. **Evaluation against hub-and-spoke baselines.** We evaluate connectivity, decentralization, scalability under load, and on-chain verification overhead, comparing to direct IBC connections and hub-and-spoke approaches (Axelar and Cosmos Hub).

4. **Demonstration application.** We implement a StableSwap showcase to demonstrate how multi-hop routing reduces liquidity fragmentation; in our evaluation, 3-hop routing achieves near-ideal 1:1 USDT/USDC pricing even for $100 million trades, while a single-chain baseline returns less than $50 million USDC due to limited local liquidity (Figure 5.8).

The rest of this chapter summarizes on our three route computation designs and discusses their trade-offs in security, cost, latency, and customizability.

## 1.1 Off-Chain Router

Blockchains charge fees, known as *gas fees*, on each performed transaction based on the amount of computation and state reads/writes performed. Since the transaction becomes permanent on the network and the overhead is high, performing either complex on-chain computation or manipulating large states incur huge costs. Building an on-chain router (a module that maintains network-wide state and computes routes on-chain) is not a feasible effort because it requires tracking of a large amount of state across many chains in the network to guarantee user-specified security properties.

To avoid the limitations and high costs of an on-chain router, we explored and designed Off-Chain Router, a routing mechanism that is queried by relayers to retrieve the best route based on user-specified requirements. The router is a stateful program that periodically updates its internal state with the current state of the network and makes routing decisions based on existing IBC connections and the specified requirements. The router can be run by relayers themselves or by third parties. Route computation is done off-chain, while the destination chain still verifies on-chain that the chosen route satisfies the user's security policies before accepting the packet.

This approach is cost efficient because it has a tiny on-chain footprint and offers low latency. Security of this method is enforced on-chain, with strict verification of the source and intermediate hops on the destination chain. However, enforcement of preference policies of this method depends on monetary incentives for the relayers and is not enforceable.

## 1.2 Zero-knowledge Proof-based Route Computation (zkRouter)

Zero-knowledge Proofs (ZKPs) allow one party (the prover) to demonstrate knowledge of specific information to another party (the verifier) without revealing the actual information. Besides the privacy properties of ZKPs, verifier-side computational advantage has made them popular in the blockchain ecosystem; particularly **zk-SNARKs** (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge), provide efficient verification and have minimal computational and storage overheads. The elimination of the need for trusted intermediaries and also the computational advantage makes ZK proofs suitable for scalability solutions on blockchains, where they are used to prove that certain off-chain computation was correct and follows the defined rules in a cheap and efficient way. **Groth16** and

**PLONK** are two SNARK constructions which enable this practically through constant-size proofs: 6 field elements (∼200 bytes) for Groth16 and 16 field elements (∼512 bytes) for PLONK. Both systems utilize native blockchain curve operations for efficient on-chain proof verification.

We designed a zero-knowledge proof-based route computation method, **zkRouter**, which computes the routes based on the latest state of the network, provides a succinct ZKP of correct route computation and satisfaction of the user requirements. The destination chain succinctly verifies this proof upon receipt of a multi-hop packet and accepts the message only if verification succeeds. While this method provides excellent cryptographic security, high customizability of routing parameters, and on-chain cost efficiency, proof generation remains computationally intensive. We benchmark our approach to assess its feasibility and explore its potential for future deployment.

## 1.3 Relayer Network

As described previously, off-chain routers suffer from limited customizability while the zkRouter incurs high latency and off-chain compute overhead. Multi-hop IBC has a single-relayer-centric design, where one relayer is responsible for updating all light clients and delivering all packets along a multi-hop path. This design choice is made mostly to ease implementation and avoid fee division, even though the responsibility is easily divisible. These challenges led us to seek a new routing mechanism that:

- **is collaborative and scalable:** The end goal of routing is to deliver packets through intermediate chains from the source to the destination. Every chain on the path uses different rules, possibly different technologies, and mainly different gas currencies. The task is inherently multiparty and forcing all duties on a single entity not only hurts scalability but also disincentivizes participation due to administrative overhead.
- **offers a practical level of security:** The protocol should offer a security mechanism with reasonable, low trust assumptions (not necessarily trustless) to be able to provide practical blockchain use cases. The extent of routing use cases (e.g., fee- or liquidity-sensitive applications that depend on fresh on-chain data, such as DEX pool states) should not depend on non-deterministic, incentive-based models. In these settings, users need deterministic guarantees; best-effort incentives are insufficient when preferences depend on fast-changing state.

- **has low latency and is cost efficient:** The overhead of routing should be reasonable both in terms of latency and incurred costs to end users, to make it relevant in the blockchain space.

We designed the Relayer Network, a collaborative route computation and overlay network that allows relayers to work together to perform route computation and delivery of cross-chain user messages. This network is permissioned through staking and is secured through cryptoeconomic guarantees, holding faulty parties accountable.

# Chapter 2

# Background and Related Work

In this chapter, we provide an overview of blockchains, the Cosmos ecosystem (including the Cosmos SDK and the CometBFT consensus), the Inter-Blockchain Communication (IBC) protocol and its components, recent extensions like multi-hop IBC, and the fundamentals of zero-knowledge proofs (ZKPs). We conclude by discussing current limitations that motivate our work for policy-aware, efficient routing in IBC.

## 2.1 Blockchain

A *Blockchain* is a distributed system that maintains a decentralized ledger recording transactions in sequential blocks, each referencing the hash of previous block [45]. This chaining of blocks using cryptographic hash functions makes the ledger tamper-evident and altering any past block would break the link to subsequent blocks. In Bitcoin, participant nodes achieve consensus on a single transaction history based on computational work (Proof-of-Work) to extend the longest valid chain [45]. This results in an append-only ledger that is immutable under normal condition. An attacker would need to expend infeasible resources to re-write a confirmed history.

Beyond simple payments, modern blockchains support *smart contracts*, allowing programmable transactions. The idea was popularized by Ethereum in 2014 [18]: Ethereum introduced a general-purpose blockchain with a built-in virtual machine, enabling "programmable money" and applications like decentralized exchanges and financial instruments. These contracts run on-chain and can enforce complex logic for digital assets.

Blockchain consensus mechanisms have also evolved. Bitcoin's Proof-of-Work provides robust decentralization at the cost of high energy usage and probabilistic finality (multiple confirmations are needed for confidence) [34]. In contrast, many modern chains use Proof-of-Stake, where validators stake tokens and vote , by locking up their tokens and participating in block proposals and approvals according to the protocol's rules on blocks, achieving faster finality with requiring far less energy [34]. For example, Cosmos chains use CometBFT (formerly Tendermint) consensus, finalizing blocks in a few seconds [17]. Each approach has trade-offs in security, efficiency, and complexity, but all aim to maintain the core properties of a blockchain: a consistent, append-only ledger maintained without a central authority.

## 2.2 Cosmos Network and App-Chains

The Cosmos Ecosystem consists of interconnected blockchains to address interoperability and scalability issues that exist in blockchain networks. Instead of relying on centralized bridges and oracles, Cosmos allows connection between chains by its Inter-Blockchain Communication (IBC) protocol. The design encourages developing blockchains for specific application logic (a.k.a. App-Chains) and relies on connectivity, promoting the concept of "Internet of Blockchains". The Cosmos Hub is the central blockchain that connects to various blockchains that operate with their own consensus mechanisms and governance models.

This is possible because of Cosmos SDK, a modular toolkit for creating blockchains, abstracting away the low-level primitives like consensus and connections to other chains, allowing the blockchains to focus on the application logic. It's powered by CometBFT consensus algorithm, which is a Byzantine Fault Tolerant protocol based on the Proof-of-Stake mechanism, ensuring network security and reliability even in the presence of malicious actors. CometBFT finalizes blocks in round-based voting and tolerates up to $f < 1/3$ of validators being malicious while guaranteeing safety and liveness.

For example, *Osmosis* is a Cosmos SDK-based chain specializing in decentralized exchange. Launched in 2021 shortly after IBC became operational, Osmosis customized its chain parameters and modules for automated market making and became a major hub for liquidity in the Cosmos ecosystem [30]. It interconnected with dozens of other Cosmos chains via IBC, and as of 2024 Osmosis handled over 20% of all IBC transfer volume [30]. Many other projects (from DeFi to gaming to infrastructure) have adopted this app-chain model.

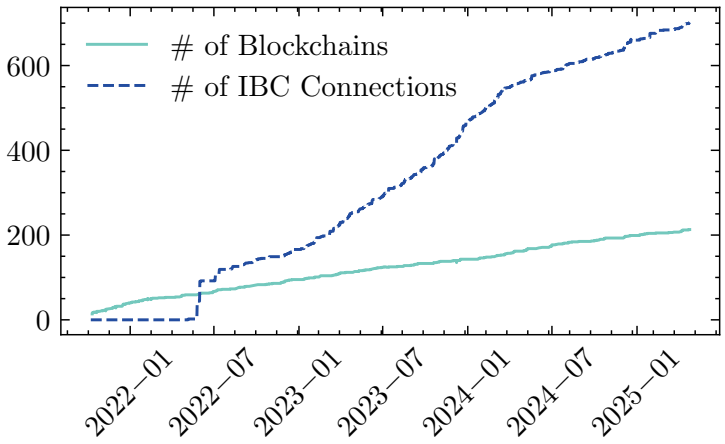



Figure 2.1: Growth in IBC-enabled chains and connections (2021–2025).

### 2.2.1 Inter-Blockchain Communication (IBC)

The Inter-Blockchain Communication (IBC) [2] protocol provides secure message passing and token transfer between blockchains in a trust-minimized way. Unlike centralized bridges, IBC uses light-client verification on-chain. Each chain running IBC maintains a light client of the other chain's state. When one chain sends a packet, it is accompanied by a cryptographic proof (a Merkle proof of inclusion in that chain's state). The receiving chain uses its light client to verify the proof against the known header of the sending chain [38]. If valid, the packet is processed. The only trust assumption is the honest majority of each chain's validators.

Figure 2.1 illustrates the growth of the IBC network from 2021 to 2025. In mid-2021, only a handful of Cosmos chains were IBC-enabled. By mid-2023, there were about 50, and by late 2024 the number surpassed 100. Connections grew even faster than the number of chains, showing super-linear scaling [46].

*IBC Architecture:* The protocol is specified in Interchain Standards (ICS). Core transport standards include ICS-2 (Clients), ICS-3 (Connections), and ICS-4 (Channels/Packets). On top, application standards like ICS-20 (token transfers) and ICS-27 (interchain accounts) are widely used [39]. Underpinning all proofs is ICS-23 (Merkle proofs). This modularity is akin to TCP/IP layering.

### 2.2.2 Relayers

IBC relies on off-chain *relayers* to deliver packets between chains. Relayers monitor one chain for outbound packets and submit them to the destination chain [28]. Running a relayer incurs costs (infrastructure and transaction fees). Early on, most relayers were run by foundations or validators as a public good [22]. To improve sustainability, the community introduced ICS-29 (fee middleware), allowing users to attach fees to IBC packets so relayers can be compensated [47]. However, adoption has been slow: as of early 2024, none of the top five IBC chains had ICS-29 enabled, though about two dozen smaller chains did [37].

### 2.2.3 Multi-Hop Extension

The current IBC protocol can only send messages between two directly connected chains. A multi-hop extension [48] has been proposed that can reuse existing connections between chains to send messages between chains that are connected through extra intermediate hops.

This standard relies on the transitive nature of trust between these chains and multi-hop verification of the consensus state along the path.

Verifying a message across multiple hops requires a chained proof showing that the first intermediate hop observed the message on the source chain, and each subsequent hop observed the previous hop's observation. Figure 2.2 demonstrates an example of how a cross-chain transaction from chain A to chain C is performed. First a relayer queries chain A for outgoing messages. If there are outgoing messages, it collects the current consensus state from chain A, and uses it to update the light client in chain B. It then queries chain A for the message commitment, and queries chain B for its consensus state, and delivers both to chain C, which allows chain C to verify that the message originates from chain A and is correctly recorded on the source chain.

## 2.3 Zero-Knowledge Proofs

Zero-knowledge (ZK) proofs let a prover convince a verifier that a claim is true without revealing why it is true. They provide correctness with privacy. Below we outline core properties, main systems used today, and how zkVMs prove program execution.

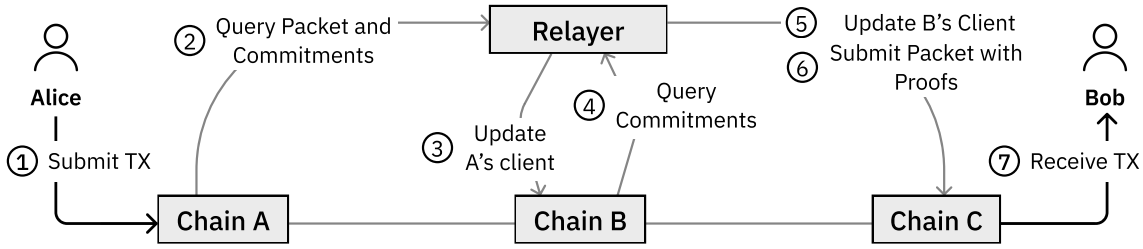



Figure 2.2: Sequence diagram illustrating a multi-hop packet flow.

### 2.3.1 Definition

A zero-knowledge proof is a way of proving the validity of a statement without revealing the statement itself, introduced by Goldwasser, Micali, and Rackoff [29]. The 'prover' is the party trying to prove a claim, while the 'verifier' is responsible for validating the claim.

The three fundamental characteristics that a ZKP should satisfy:

- **Completeness**: If a statement is true, then an honest verifier can be convinced by an honest prover that they possess knowledge about the correct input.
- **Soundness**: If a statement is false, then no dishonest prover can unilaterally convince an honest verifier that they possess knowledge about the correct input.
- **Zero-knowledge**: If the state is true, then the verifier learns nothing more from the prover other than the statement is true.

### 2.3.2 History

ZK began as interactive proofs [29]. NIZKs remove interaction using a setup (common reference string) [15], while Fiat–Shamir removes interaction in the random oracle model by hashing transcripts [26]. PCP results showed complex statements admit short, randomly checkable proofs [9]. Modern systems balance prover cost, verifier cost, setup assumptions, and proof size.

### 2.3.3 SNARKs

SNARKs (Succinct Non-interactive Arguments of Knowledge) encode a computation as arithmetic constraints and produce very small proofs that verify quickly. A common arith-

metization is the Rank-1 Constraint System (R1CS), which represents a program as constraints that must all hold.

In R1CS, each step of the computation becomes a constraint. A typical constraint has the form:

$$a(x) \cdot b(x) = c(x)$$

Translating high-level code into R1CS often dominates engineering effort.

Classic pairing-based SNARKs (e.g., Groth16 [31]) achieve sub-kilobyte proofs and millisecond verification on popular curves. The trade-off is a *trusted setup*: a ceremony produces a common reference string (CRS) with a proving and a verification key. Universal SNARKs (e.g., PLONK) reuse a single CRS for many circuits, reducing ceremony overhead at the cost of larger keys and proofs. In practice, SNARKs are preferred when on-chain verification cost is critical; proving can be heavy but benefits from batching, recursion, and hardware acceleration. For an accessible overview, see [52].

### 2.3.4 STARKs

STARKs (Scalable Transparent Arguments of Knowledge) avoid trusted setup and rely only on collision-resistant hashes and information-theoretic checks [14]. Computations are expressed as an Algebraic Intermediate Representation (AIR) and evaluated over a large execution trace. The prover commits to this trace and uses low-degree testing (e.g., FRI) to convince the verifier that all transition rules hold.

STARK proofs are larger (often tens to hundreds of kilobytes), but proving is highly parallel and verification is simple: mostly hash queries plus polynomial checks. Security is plausibly post-quantum. These properties make STARKs attractive when transparency and auditability matter more than minimal proof size. In practice, systems may also wrap a STARK in a SNARK to keep on-chain verification costs low.

### 2.3.5 zkVM

A zkVM (zero-knowledge virtual machine) proves correct execution of programs without custom circuit design. RISC Zero, for example, runs RISC-V binaries and produces a STARK receipt of the instruction trace [64, 55]. Developers write a guest program, compile to an ELF, run in the executor, and obtain a receipt that any verifier can check. An

ImageID (a hash of the program) binds the proof to the intended binary, and a public journal commits outputs. Similar efforts exist for zkEVMs and zkWASM. For zkRouter, a zkVM lets us prove off-chain route computation or batched light-client checks while chains verify a short proof on-chain.

## 2.4 Limitations

While IBC minimizes trust assumptions, it still has practical limits. New connections require creating light clients and completing multi-step handshakes, which adds coordination and on-chain cost. This slows adoption and encourages hub-and-spoke topologies, where most chains connect to a few large hubs.

Multi-hop IBC can reuse existing connections to reach more chains. However, it introduces liveness and trust dependencies on all intermediate hops and does not specify how to compute routes. Additional infrastructure is needed to select paths and enforce application policies.

## 2.5 Related Work

The field of cross-chain communication has received significant attention from both academia and industry given its potential for financial applications. The financial implications of these systems necessitate the design of protocols that are not only efficient but also secure, since compromises can lead to losses of millions of dollars in user funds.

Numerous surveys have been published on cross-chain communication protocols, offering comprehensive categorizations and analyses of various approaches [10, 58, 13, 19, 12, 32, 43, 54]. In this section, we review key protocols, their limitations, and their relevance to our system.

**Notary-Based Mechanisms:** A number of approaches rely on Notaries, systems that listen to the source chain for events and then confirm the events on the target chain. These solutions involve different degrees of trust assumptions, ranging from central entities, to multi-sig wallets and decentralized methods. Chainlink's Cross-Chain Interoperability Protocol (CCIP) [20] is built upon Decentralized Oracle Networks (DONs) acting as Notaries, but also relies on its Risk Management Network to detect malicious behavior in these DONs.

Hyperlane [35] and LayerZero [63, 42] also utilize the same concept by introducing more smart contracts, first one defines flexible inter-chain security modules (ISMs) while the latter uses decentralized verifier networks (DVNs) which provides better trust assumptions. Omnity (previously known as Octopus) [49, 50] deploys its main logic on ICP (Internet Computer) blockchain that uses its own unique cryptographic primitive known as Chain-Key Cryptography [27], a threshold signature method that can be created for any contract data by validators, to confirm events as simple as a signature verification.

This model is also widely adopted by many bridges which use multi-signature wallets to control the notary and upon breach lead to million-dollar losses. While decentralized solutions provide enhanced security, this approach still needs the additional trust assumption on the notary.

**Native Interoperability and Hub-and-Spoke Architectures:** The Cosmos ecosystem [41] uses the IBC protocol and light clients to connect chains via hubs (e.g., Cosmos Hub). Because interoperability is achieved via light-client verification, trust assumptions are minimized.

Axelar [11] is a blockchain built on the Cosmos SDK that runs as a zone within the larger Cosmos ecosystem. The chain provides an abstraction known as gateways, which can transfer tokens and execute smart-contract calls across ecosystems in a path-agnostic manner.

Polkadot [61] is another hub-and-spoke solution built around a Relay Chain that allows developers to build parachains (i.e., sidechains that are not fully independent). To join the network, a new parachain must participate in a slot auction and lock DOT to lease a slot. Reliance on a single hub for governance, security, and message passing limits scalability and introduces additional trust assumptions (e.g., throughput coupling to the hub).

**Routing-Based Solutions:** In these approaches, a new blockchain acts as a router for transactions between ecosystems, coordinating route computation and, in some cases, message verification.

Wang et al. [60] present a design that resembles Cosmos but emphasizes cryptoeconomic security over trustless relayers and light clients; Anlink [59], an extension of this work, improves scalability. Ding et al.'s InterChain [24] focuses on cross-chain asset transactions but requires some subchain nodes to connect to their router, introducing compatibility issues and certain trust assumptions. Kan et al. [40] design a multilayer system and implement a three-phase commit protocol across chains. Limitations of these approaches

include added trust assumptions through the router blockchain and a lack of practical implementations.

**Smart Contract-Based Solutions:** These solutions implement cross-chain transfer logic via smart contracts deployed on the source and destination chains. Two major methods exist: (1) systems that introduce a trust assumption on a third-party entity/chain to verify transfers [53]; and (2) on-chain light-client implementations that rely on trustless relayers [44, 62, 57]. Although the latter minimizes trust assumptions, verification costs are often high, especially when the source and destination blockchains are heterogeneous.

**Asset Transfer-Focused Solutions** Several systems focus on facilitating asset transfers and exchanges across blockchains. Some employ techniques similar to those described above [6, 7, 33], while others use cryptographic primitives such as Hashed Time-Locked Contracts (HTLCs) [8]. These designs are inherently limited by their inability to support other use cases, such as arbitrary message passing or smart-contract execution across ecosystems.

# Chapter 3

# System Architecture

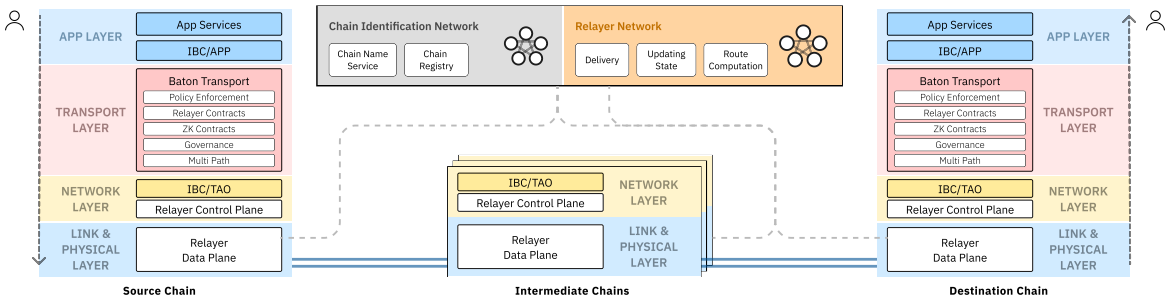



Figure 3.1: High-level system architecture.

In this chapter, we describe the system design and architecture of our policy-driven cross-chain routing stack. Figure 3.1 shows the high-level architecture of our system, which is organized into abstraction layers that are modeled after network stacks and work together to deliver cross-chain messages. Our system has four main layers:

- **Application Services** – ***App Layer*** offers application-level services for cross-chain messaging. It enables other modules to send messages across chains seamlessly without dealing with the underlying protocol. Applications can either use the high-level messaging API, which will be the case for most of the decentralized applications, or call the transport layer directly when they need to specify custom policies or routing behaviour.
- **Transport** – ***Transport Layer*** contains our core modules responsible for policies and delivery: (1) the Policy Enforcement Module enforces policies (e.g., security

thresholds, fee budgets, latency targets) on messages and verifies integrity of the message upon delivery; (2) Relayer Contracts facilitate operations of the Relayer Network, such as verifying messages and fee distribution; and (3) zkRouter contracts enable efficient verification of computed routes.

- **Relayer Control Plane (RCP)** – ***Network Layer*** is responsible for computing routes to the destination chain, based on the user's choice of routing method: local computation, retrieving a zero-knowledge proof, or via the Relayer Network. It makes routing decisions that take into account security requirements, gas costs, congestion, and relayers' gas-token availability.
- **Relayer Data Plane (RDP)** – ***Link and Physical Layers*** is responsible for watching for new outgoing IBC packets , updating intermediate client states, and submitting packets to the destination. These operations are done by using RPCs provided by chains.

Additionally, the Relayer Network which serves a role similar to a controller in an SDN and provides: (1) Chain ID Service, a mapping from chain identifiers to connection and client identifiers allowing the use of names for chains (2) Route Computation, which computes the best path between two chains based on specified policies, enabling flexible policies that would otherwise be impractical; and (3) Chain RPC Lookup, used by the RDP to retrieve reliable RPC endpoints for a given chain identifier.

In the following sections, we describe the functionality of the different modules in turn.

## 3.1 Policy Enforcement Module

A **policy** specifies conditions that a route must satisfy (e.g., every intermediate hop must have Nakamoto coefficient $\geq 8$). Policies are appended to messages and specify how the message must be routed and delivered (e.g., via specific chains, with latency or cost constraints). We distinguish between **security policies**, which must hold for packet acceptance and are enforced deterministically on-chain via IBC verification plus any required auxiliary data (e.g., an intermediate chain's validator set, verified against the light client state, to compute Nakamoto coefficient), and **preference policies**, which should hold for best user experience (e.g., fee minimization) where enforcement depends on the routing method.

The policy module provides two main functions. **First**, it allows a user to enqueue a new cross-chain packet with preferred policies for the route that the packet will take. For

Table 3.1: Transport-Layer API Interface

| **Call** | **Caller** | **Semantics** |
|---|---|---|
| *Core Transport Interface* | | |
| `SubmitPacket(pkt, policy)` | App | Enqueues a packet with an attached policy; returns a packet sequence number. |
| *Relayer Contracts Interface* | | |
| `EscrowFees(pkt_seqno)` | App | Escrows fees for the Relayer Network. |
| *Policy Enforcement Interface* | | |
| `SubmitRoute(pkt_seqno, route_info)` | RCP | Submits the computed route for the packet. |
| `UpdateConfig(config)` | RCP | Updates the configuration for Relayer Network contracts. |
| `OnNewPacket(pkt)` | IBC/TAO | Informs the Policy Enforcement Module about a new packet. |
| *Verifier Contracts Interface* | | |
| `VerifyPacketRoute(pkt_seqno, route_proof)` | Policy Enforcement | Verifies the packet and whether the routing rules were respected. |

example, a user might require that all of its messages hop through chains meeting specified security requirements (e.g., a chain with a Nakamoto coefficient greater than or equal to 6). **Second**, upon receipt of a new packet on the destination chain, the module verifies that packets comply with the specified policies.

Depending on the user's needs, different kinds of policies can be specified. These include, but are not limited to, route-computation methods, security requirements, delivery time, and fee constraints. We also support fine-grained custom policies, which may be optional or enforced depending on the route-computation method the user chooses.

### 3.1.1 Policy Definition

In IBC, channel versions can encode information about specific use cases or channel properties, and all endpoints must agree on the channel version during channel setup. We use channel versions as the primary reference for routing methods and security-policy enforcement. To ensure channel longevity, additional policies are typically excluded from versioning to allow efficient reuse of the channel for future messages. Figure 3.2 shows the version syntax and two examples.

### 3.1.2 Routing Method

**Definition 3.1.1** (Route)**.** A route in our system is a list of connection identifiers, one per hop along the path between the source and the destination. A long-lived multi-hop channel can be established by relayers based on a computed route and used to transport packets.

A routing method defines how the route is computed and what guarantees it provides. Our system currently supports the following three methods, which we describe in detail in Section 3.2:

1. **Single-Relayer routing**: a relayer computes routes based on policies specified by packet and submits them for verification.
2. **Relayer Network–based routing**: a collaborative network of relayers computes routes and delivers packets.
3. **Zero-knowledge router (zkRouter)**: route computation is handled off-chain with a zero-knowledge proof that is verifiable on-chain.

### 3.1.3 Security Requirements

In our design, security requirements are specified as policies and are **strictly** verified by the target chain. If a relayer attempts to deliver a message through a route that does not satisfy one of the security conditions, the message is rejected. Two example requirements:ds

1. **Minimum acceptable Nakamoto coefficient:** each connection hop on the computed route must have a coefficient greater than the specified value.

| **Syntax** | `[App Version]/[`$P_1$`]:[`$PV_1$`]/...` |
|---|---|
| Hops with more than 4 validators | `ics20-1/validators:4` |
| Use Relayer Network | `ics20-1/routing:rn-cosm..gz9yjv` |
| Use zkRouter | `ics20-1/routing:zk-cosm..fg3vqd` |

Figure 3.2: Syntax for channel versions.

2. **Minimum number of validators:** each connection hop on the computed route must have at least the specified number of validators.

### 3.1.4 Message Delivery Fees

Users have two ways to specify preferred delivery fees: (1) provide an upper-bound maximum fee; or (2) request a route with the minimum possible fee. The former is easy to enforce: if a relayer chooses a more expensive route, the relayer is not compensated beyond the cap. The latter is harder to enforce because optimality is not verifiable on-chain; however, by holding relayer fees in escrow it is possible to return a portion of the fee if another relayer later submits a cheaper route that satisfies the user's needs. This economic incentive discourages routing messages through unnecessarily expensive paths.

### 3.1.5 Message Delivery Time

Two policies can be defined for message-delivery time: (1) a timeout policy, in which the user specifies a deadline based on either wall-clock time or the destination's block height; and (2) a minimum-time policy, in which the user indicates a preference for shorter routes.

### 3.1.6 Policy Enforcement

#### Security Policy Enforcement

To enforce security policies on-chain, relayers need to retrieve the necessary proofs off-chain from the source and intermediate chains, and submit them to the destination for verification. These proofs are needed in addition to those needed for plain multi-hop messages,

and the relayer can determine which proofs are required based on the specified policies and the channel version.

To demonstrate the process, we walk through how a security policy is verified in a multi-hop message. We define $CS_C(B, h)$ as the consensus state of chain $B$ at height $h$ recorded on chain $C$, which has the following fields:

```
type ConsensusState struct {
    Timestamp time.Time
    AppState []byte
    NextValidatorsHash []byte
}
```

Take $A$ and $C$ as two chains connected in a multi-hop topology where $B$ is an intermediate hop between them. The relayer must provide information about the validator set of $B$ for $C$ to accept the message. Assuming $CS_C(B, h)$ is the minimum height at which multi-hop message $M$ is verifiable, any honest relayer can query the validator set of chain $B$ at block $h{+}1$ and submit it along with the other computed proofs and the packet to the destination. Let $\pi_{nakamoto}(validatorSet, n)$ be our policy function, which returns `true` if the validator set has a Nakamoto coefficient greater than or equal to $n$. The destination chain verifies (1) $\text{hash}(validatorSet) = CS_C(B, h).\text{NextValidatorsHash}$ and (2) $\pi_{nakamoto}(validatorSet, n) = \texttt{true}$, and accepts the message only if both hold.

#### Preference Policy Enforcement

Most preference policies are not directly enforceable in the single-relayer routing method because on-chain verification is either impossible or impractically expensive. We try to satisfy these preferences by incentivizing correct behavior and address verifiability in the other two routing methods discussed next.

## 3.2 Route Computation

In this section we discuss how routes are computed for each routing method described in Section 3.1.2.

### 3.2.1 Routing Algorithm

All of our route computation methods share the same routing algorithm; they differ only in how routes are computed and in the guarantees they provide for satisfying preference policies. Security policies are strictly enforced on-chain regardless of the method and all methods provide same level guarantees for them.

Figure 3.3 describes the high-level procedure for computing the route. As of now, we support at most one optimizing policy, which is common for blockchain use cases. In the future, a Multi-Constrained Shortest Path algorithm would better support richer optimizing policies.

```
def compute_route(network_state, src, dst, policies):
    for filter_policy in filter(policies, is_filtering_policy):
        network_state = filter_policy.filter(network_state)
    graph = construct_network_graph(network_state)
    # we support maximum of one optimizing policy
    optimizing_policies = filter(policies, is_optimizing_policy)
    if len(optimizing_policies) == 0:
        return dijkstra(graph, src, dst)
    op = first(optimizing_policies)
    op.assign_weights(graph)
    if has_negative_edge(graph):
        return bellman_ford(graph, src, dst)
    else:
        return dijkstra(graph, src, dst)
```

Figure 3.3: High-level implementation of the routing algorithm.

### 3.2.2 Single-Relayer Routing

For any given message, one relayer handles all necessary tasks, from computing the route to delivering the packet and submitting the acknowledgment. To compute routes correctly, the relayer periodically queries RPC endpoints for the chains that it is relaying to maintain the latest available network state. On a source chain that receives a request, the relayer computes possible routes based on the current state and selects a route that satisfies

the policies specified for the message. This approach communicates with the governance module to query and ensure correct mapping of chain identifiers to channel identifiers.

### 3.2.3 Relayer Network–based Routing

The Relayer Network (see Section 3.3) allows relayers to collaborate on route computation, chain-identifier mappings, and packet delivery. We organize the relayers as a separate independent chain that introduces cryptoeconomic security and fee distribution, enabling more flexible guarantees for non-security policies (e.g., gas minimization). It also provides cheaper chain-identifier mappings compared to the governance module approach. More details will be discussed in Section 3.3.

### 3.2.4 Zero-knowledge Routing

This method computes the route off-chain by employing zero-knowledge proofs that can be verified by the destination chain. Our zero-knowledge program is implemented in the RISC Zero zkVM [64] and: (1) takes as input the message with policies and the latest network state (queried from RPCs) along with the Merkle proofs of needed on-chain states (2) verifies the proofs and input validity, and (3) computes and outputs the route along with the block header that these inputs correspond to. The program is a RISC-V ELF that its hash is recorded on-chain, and each time the program is executed by the zkVM, it generates a compact proof of correctness along with the output. This approach flexibly adapts to different policies and can provide optimality guarantees. Proof verification occurs on-chain and takes milliseconds to complete wihtout needing a GPU, keeping on-chain costs low. However, generating proofs demands significant computational resources (e.g., GPUs).

## 3.3 Relayer Network

In this section, we first discuss the necessity of a collaborative approach in relaying messages. Second, we explain how we design an organized system, **Relayer Network**, that allows relayers to collaborate, compute routes, and share fees while keeping the original simplicity from the user's perspective.

Table 3.2: IBC/TAO Interface

| Call | Caller | Semantics |
|---|---|---|
| `SendPacket(pkt)` | Policy Enforcement | Enqueues a packet as ready to send. |
| `QueryNewPackets()` | RDP | Checks and retrieves the list of new packets ready to send. |
| `SubmitAck(pkt_seqno, delivery_proof)` | RDP | Submits acknowledgment of delivery for a specific packet. |
| `SubmitTimeout(pkt_seqno, timeout_proof)` | RDP | Submits a timeout along with the receipt for the timeout from the destination chain. |
| `UpdateClientState(client_id, new_state)` | RDP | Updates the corresponding client's consensus state after verification. |
| `QueryProof(key)` | RDP | Queries a proof of inclusion/absence with respect to the chain's state. |
| `DeliverPacket(pkt, proofs)` | RDP | Delivers the packet with all necessary proofs to the destination chain. |

### 3.3.1 Rationale

In original IBC, packet relaying uses a winner-takes-all approach: between two competing relayers, only the first one receives rewards. The multi-hop extension, while introducing several new responsibilities, assumes a single relayer handles all actions. Historically, many relayers were maintained by blockchain foundations. With the recently introduced fee-incentives module [47], we expect multiple relayers to join and compete.

The current approach is problematic for two reasons: (1) a relayer must monitor many chains and hold native currencies for each to submit transactions, which is impractical at scale; and (2) based on experiments in [21], competing relayers can worsen transaction throughput. These factors motivate a better-organized design that leverages multiple relayers to improve performance rather than wasting resources in a winner-takes-all scenario.

Additionally, this network can provide Chain Identification Services (see Section 3) as an alternative to the governance module for retrieving reliable chain-identifier-to-channel mappings.

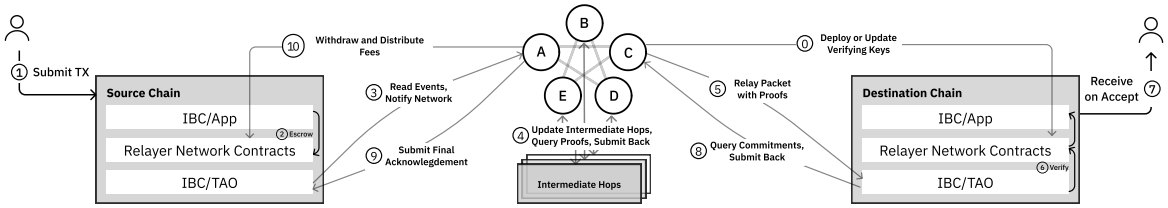



Figure 3.4: Relayer Network architecture.

### 3.3.2 New Relayer Roles

We introduce the following relayer roles in order to divide the responsibilities and make collaboration between multiple relayers possible:

**Multi-hop Channel Creators:** These entities create multi-hop channels based on computed routes between source and destination chains. They perform the IBC handshake to create a new channel between two chains.

**State-update Relayers:** To forward a packet along intermediate chains, a relayer must query a multi-hop proof of each chain's consensus state. Before that, each blockchain's light-client state must be updated to a height at which the new transaction on the source chain is visible and thus provable. The State-update Relayer updates the consensus state of light clients along the path in the proper order.

**Multi-hop Packet Relayers:** These relayers handle user packets end-to-end. They monitor the source and destination chains, receive queued messages, compute consensus-state and Baton-policy proofs, and submit them to the destination chain. They then monitor the inclusion of the transaction (or detect exclusion on failure) and relay an acknowledgment or timeout packet back to the source chain, completing the packet flow.

### 3.3.3 Design

Our Relayer Network is a cooperative network of relayers who collaborate to perform tasks more efficiently than they could independently. By splitting responsibilities across multiple relayers, we avoid the scalability issues described earlier and allow competing parties to share profits proportionally based on their contributions rather than winner-takes-all. We

achieve this with two core components: (1) a Coordination Network and (2) Management Contracts.

**Coordination Network:** A permissioned, Byzantine-fault-tolerant blockchain built with the Cosmos SDK. Relayers stake a minimum amount of tokens to enter the network and face penalties upon malicious acts or unavailability. Each relayer collects events from chains in which it participates and broadcasts them to the network. On detecting an event for a chain of interest, a relayer picks up the message, performs the necessary actions on the target chain, and submits the result with proofs back to the network. The next relayer observes the progress and continues the work. This repeats until the message is delivered and the acknowledgment packet is relayed back to the source chain. The Relayer Network also maintains the current state of the Cosmos network to support route computation; when new channels or chains are created, the network reaches quorum to update chain and channel mappings.

**Management Contracts:** Management contracts reside on chains that are sources or destinations (intermediate chains do not need to host them) and are deployed by Relayer Network participants. These contracts (1) receive escrowed fees from the user; (2) hold a list of public keys that can be updated by network quorum; and (3) verify each message on the destination to ensure that routing information approved by the Relayer Network is respected.

Figure 3.4 illustrates the architecture of the Relayer Network and the full flow of a packet collaboratively routed through it. Table 3.3 shows the high-level network interface.

### Membership

To become a member of the network, a relayer deposits and locks (stakes) a minimum required amount of funds. If a relayer misbehaves, other participants can propose slashing of that participant's funds. This deters misbehavior and the submission of invalid routes.

### Task Division and Participation Proofs

Upon joining the network, a relayer specifies the networks in which it specializes and runs clients. Our current implementation assigns tasks on a round-robin basis among relayers that include the relevant network in their preference list. Relayers must attend to assigned

tasks without delay and submit a participation proof, which is a receipt from the chain indicating the operation and the responsible relayer.

#### Fee Division

Fees are collected on the source chain from users and kept in the management contract. All participation events and fees paid by relayers are recorded on the network. When a relayer proposes fee collection on a specified chain, the split is calculated on-chain and signed by every participant. The signed proposal can then be submitted to the management contract, which automatically transfers fees to the corresponding participants.

#### Slashing and Misbehavior Proofs

A relayer is considered to be misbehaving if it:

- **performs poorly:** the absence ratio or latency exceeds a defined threshold, or the relayer fails to participate in assigned tasks.
- **submits suboptimal routes:** the relayer fails to follow user-specified preferences.

Because the network tracks the state of the Cosmos network, any participant can assemble a proof and propose slashing when misbehavior is observed.

## 3.4 zkRouter

zkRouter is a cryptographically secure and verifiable mechanism for multi-hop packet routing. Its goal is to ensure that routing decisions across multiple blockchains satisfy user-defined policies while remaining verifiable via zero-knowledge proofs. This makes it possible to verify policies that are otherwise hard to check on-chain (e.g., minimizing gas fees or maximizing trade output).

At a high level, zkRouter gathers state information from participating blockchains and computes optimal routes based on user-specified criteria such as hop count, gas fees, and validator stakes. It then produces a zero-knowledge proof that the chosen route adheres to the specified requirements and is valid based on the current network state.

Table 3.3: Relayer Network Interface

| Call | Caller | Semantics |
|---|---|---|
| *Execution Interface* | | |
| `RegisterRelayer(networks)` | RCP | Join Relayer network with the interested networks, fails if relayer doesn't satisfy staking requirements. |
| `ProposeNetworkChange (proposal)` | RCP | Propose a change in the state of the network, adding a chain, connection, or channel with necessary proofs. |
| `ProposeSlashing(relayer, proof)` | RCP | Proposes a participant as a slashing candidate based on given proofs of misbehavior. |
| `SignProposal(id)` | RCP | Vote in favor of the given proposal. |
| `SubmitIBCEvent(src, dst, event)` | RCP | Notify the network about an IBC event. |
| `SubmitParticipationProof (chain, proof)` | RCP | Submit a proof of participation in relaying a packet. |
| `ProposeFeeCollection (chain)` | RCP | Submit a proposal for withdrawal of collected fees on given chain. |
| *Query Interface* | | |
| `QueryRoute(pkt)` | RCP | Retrieves the computed route for a packet. |
| `QueryProof(pkt, type)` | RCP | Retrieves a computed proof needed for multi-hop relaying for a given packet. |
| `ResolveChainClient(src, target)` | RCP | Resolve client identifier for the target chain on source chain. |
| `QueryRPC(chain)` | RDP | Find a reliable RPC server for given chain. |

### 3.4.1 Components

#### Data Collection Module

This component gathers state data from each blockchain's native store, including network topology, gas prices, and validator stakes. These data are accompanied by Merkle proofs to ensure authenticity and integrity. The collected data are:

- **Client States:** for each chain, the state of each of its light clients is collected to enable cross-verification of connections.

- **Connection States:** for each chain, its connections are collected to construct the network graph.
- **Validator Sets:** the validator set for each chain, to support routing requirements based on validator security and stake.
- **Light Blocks:** when some chains do not have the latest state about their counterparts, these states are encoded as light blocks and verified with the same algorithm as light clients (inside the routing algorithm) to ensure proofs are verifiable everywhere.

**Key Optimizations:** Three major optimizations are taken into account to make proving efficient:

- **Modular System:** the system has two proofs: one to obtain the latest network graph, and another for routing based on that graph. Because most connections are long-lived, the network state does not change frequently; rather than updating it every time, the second proof verifies freshness based on on-chain validity timestamps.
- **Collection Order Heuristic:** the system finds a minimum dominating set of the network and collects information in an order that, while yielding the same result, minimizes the number of light-block verifications (an expensive operation).
- **Batched Merkle Proofs:** due to the storage mechanism on Cosmos chains (a provable variant of AVL), data are stored in a semi-sorted manner, yielding shared Merkle paths for many proofs. By batching, we avoid recalculating hashes and can verify multiple proofs at once.

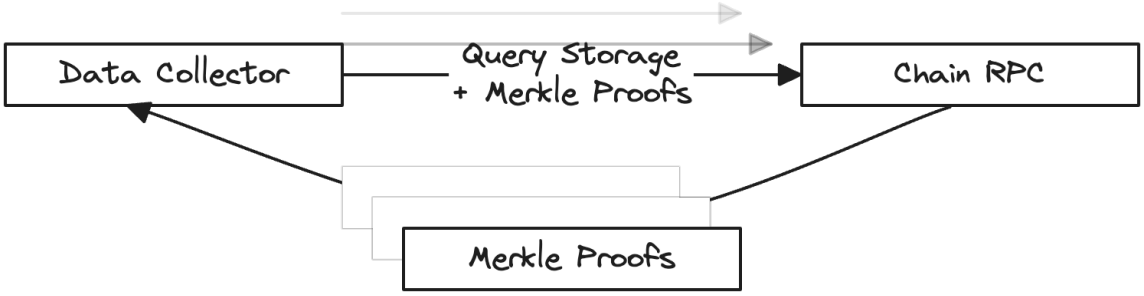



Figure 3.5: Data collection: IBC state.

Figures 3.5, 3.6, and 3.7 highlight the data-collection procedures.

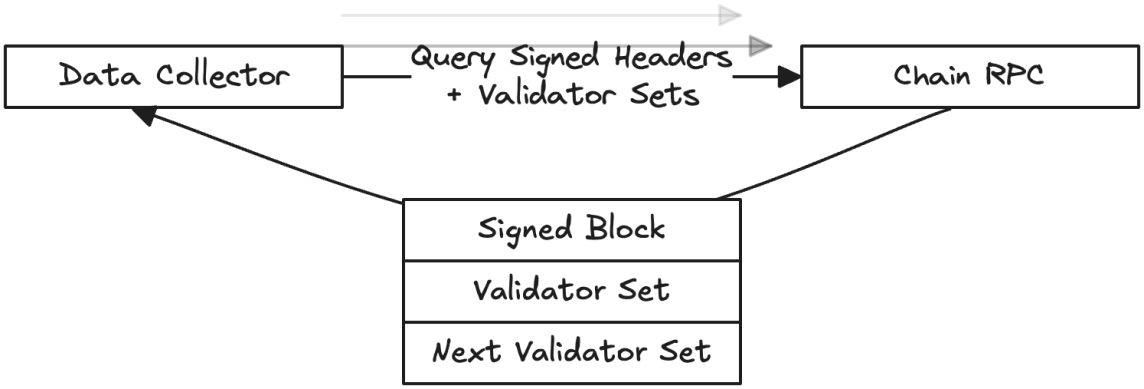



Figure 3.6: Data collection: light blocks.


Figure 3.7: Data collection: Merkle batches.

### Route Calculation

Using the collected data, zkRouter first verifies data integrity and then uses our route-computation algorithm to compute the optimal route for packet transfers. The route is determined based on criteria specified by the user, such as minimizing hop count, reducing gas costs, or selecting paths with the highest security.

The algorithm considers multiple potential paths and evaluates them according to the specified policies, ensuring that the final selected route aligns with the user's requirements. This process is highlighted in Figure 3.9.

### Proof Generation

Once the optimal route is determined, zkRouter generates a STARK (Scalable Transparent Argument of Knowledge) proof using the RISC Zero zkVM. This proof certifies that the route calculation was performed correctly and in accordance with the user-defined policies. To enhance efficiency, the STARK proof, which is initially large, is compressed into a SNARK (Succinct Non-interactive Argument of Knowledge) proof. The SNARK is much smaller and can be verified quickly, ensuring that the integrity of the routing decision can

be confirmed without significant computational overhead. Figure 3.8 demonstrates this process.

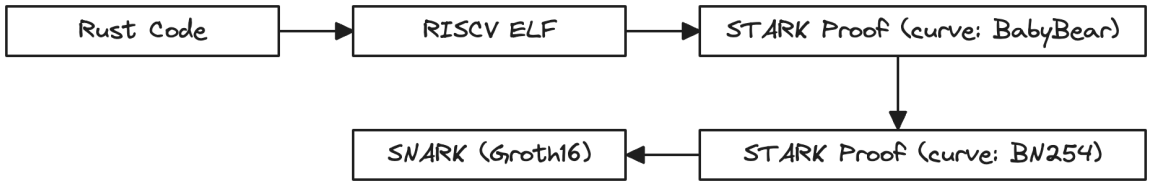



Figure 3.8: zkRouter proof pipeline.

#### Packet Transmission

Relayers, the parties responsible for packet transmission, can use our system to generate the route and proof, transmitting the packet to the destination chain, which can efficiently verify integrity and process the transaction.

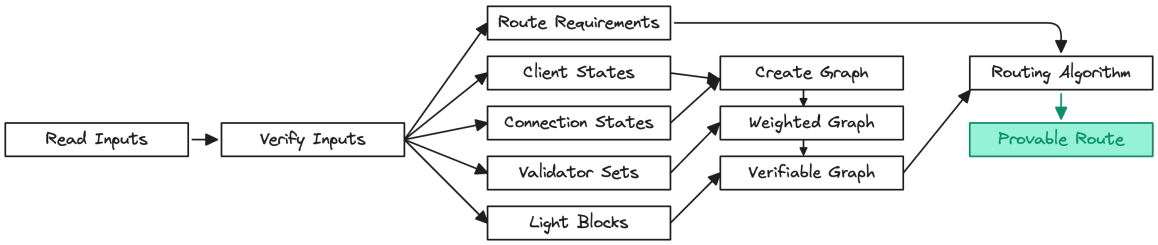



Figure 3.9: RISC-V program pipeline.

### 3.4.2 Decentralization and Security

Relayers in IBC introduce a point of centralization as they are needed to deliver messages and proofs to the destination. This is acceptable because even malicious relayers cannot deviate from user-specified security policies or forge packets. The Relayer network improves upon this by distributing tasks across different relayers and using a slashing mechanism for misbehavior.

By creating a CometBFT-based blockchain for the Relayer Network, the network is safe under $f < \frac{n}{3}$ Byzantine validators in a partially synchronous network model [25]. Participation is permissionless: any relayer may join by posting a bond on-chain.

Adversarial relayers may influence routing for fees but cannot violate security policies or alter the packet. Censorship or intentionally suboptimal routing is detectable and punishable through slashing. If an adversary compromises enough relayers to evade slashing, our on-chain fee escrow mechanism can disincentivize such attacks. The fee that the user pays for the multi-hop message is held in escrow on the source chain for a certain period, giving a third party a chance to submit evidence of misbehavior and claim the escrowed fee as a reward.

The IBC multi-hop specification does not constrain the security of intermediate hops. If an adversary compromises an intermediate chain, it can forge transaction on a source chain, forge proofs, or falsify packet information. This attack is specific to multi-hop IBC; in our system, it is possible if a chain that satisfies the security policies is compromised or malicious. However, this attack is unlikely, because it can only be done by chains that have high trust score and are rarely malicious. To mitigate this attack, we provide a multi-path module that sends messages across multiple independent routes and accepts the result only when a majority agree, providing tolerance against compromised intermediate chains.

## 3.5 Multi-chain Decentralized Applications

There are numerous decentralized applications that benefit from enhanced cross-chain connectivity, especially in the DeFi sector where performance relies on liquidity availability. We introduce the concept of multi-chain dApps: smart contracts or modules that utilize Baton to optimize performance. On any Baton-enabled blockchain, apps can access the Baton interface to perform cross-chain operations seamlessly.

### 3.5.1 Example Use Case: Multi-chain StableSwap

We showcase a StableSwap application based on existing constant-product market maker (CPMM) DEXes that can use cross-chain liquidity via Baton. CPMM exchanges price trades using the curve $x \cdot y = k$ based on demand and available liquidity. The main challenge is high slippage (price fluctuation) when trade size is large relative to pool liquidity.

The swap flow is: (1) the user submits a StableSwap request on the source chain, indicating the maximum acceptable slippage; (2) the StableSwap protocol exposes a decision function that relayers can query; (3) relayers compute a swap route that satisfies the conditions and submit it to the source chain, triggering multi-hop transactions; and (4) based on available liquidity, the trade may succeed fully or partially, and the user receives funds

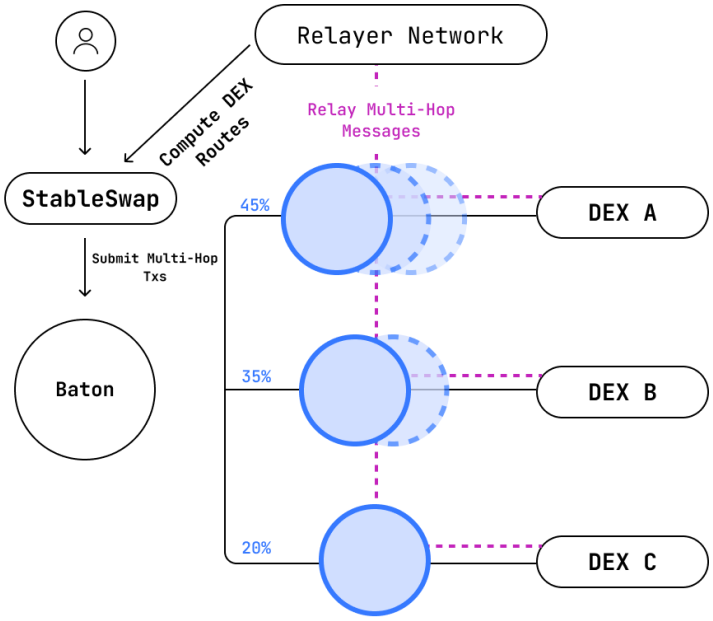



Figure 3.10: StableSwap architecture and flow.

accordingly. Figure 3.10 illustrates the flow. This use case requires relayers to query and relay DEX-specific state, such as liquidity across multiple chains, enabling the protocol to compute routes that satisfy user-defined slippage constraints.

# Chapter 4

# Implementation

We implemented a modular stack that powers our prototype and its evaluation: the Relayer Network (Cosmos SDK + CosmWasm), cooperation-aware relayers, portable Cosmos SDK modules, an off-chain router, zkRouter tooling, and evaluation and benchmark tooling. We designed the components to be reusable and composable, and intend to open-source them.

## 4.1 Relayer Network

The Relayer Network is a Cosmos SDK chain that integrates `wasmd` to host CosmWasm contracts. To keep upgrades safe and flexible, we implement the main relayer logic in CosmWasm (Rust) smart contracts. This lets us evolve coordination logic and policy checks without a full chain upgrade.

### 4.1.1 Core Contract

The core contract governs membership and accountability:

- Relayer lifecycle: join/leave and key management.
- Governance: proposals to update chain and client identifiers, and to authorize signing intents on target chains.
- Accounting: fee withdrawal proposals backed by participation proofs.
- Security: slashing proposals when policy violations are proven.

### 4.1.2 Management Contracts

We deploy lightweight management contracts on the source and destination chains:

- *Source*: escrow user fees and authorize withdrawals once the Relayer Network approves the route.
- *Destination*: track the Relayer Network verification key and verify signatures on routing decisions.

## 4.2 Relayer Implementation and Cooperation

We first extended the `ibc-go-relayer` (with Polymer Labs' multi-hop support) to enable cooperation:

- Specialization: relayers declare the chains on which they specialize.
- Scheduling: path separation with round-robin tasking.

**Finding the bottleneck.** Benchmarks showed limited throughput gains with more relayers. Investigation revealed excessive client updates: intermediate clients were updated even when not strictly required by the multi-hop specification. While this strategy is acceptable for single-hop IBC (header + proof in one transaction), in multi-hop each unnecessary client update adds multiple block intervals on the forward and backward passes, becoming the dominant cost at scale.

**Spec-aligned minimal updates.** To address this, we implemented the specification-aligned minimal-update algorithm and a new relayer that maintains compatibility with our cooperation modes and coordination. At a high level:

1. Pre-compute the minimal set of client updates needed to validate all proofs along the chosen path.
2. Batch updates where possible to amortize block delays.
3. Relay packets only when the destination verification predicates are satisfied, avoiding eager updates.
4. Acknowledge and back-propagate similarly.

## 4.3 Cosmos SDK Modules

We provide portable modules to enable policy enforcement and application experiments across Cosmos chains.

### 4.3.1 EVM Integration

To experiment with higher-level applications, we integrated the Ethereum Virtual Machine as a Cosmos SDK module by decoupling Geth's EVM from the surrounding chain logic. Unlike Evmos's tightly integrated design, this module is intended to be reusable: accounts can be backed by EVM contracts, enabling reuse of the Solidity ecosystem for complex DeFi logic.

### 4.3.2 Policy Enforcement Module

This module verifies routing decisions and enforces policies using a pluggable interface:

- Off-chain router decisions with data and checks.
- Relayer Network-signed route decisions.
- zkRouter-verified decisions (state proofs + route computation).

Security parameters (client freshness, path constraints, per-channel policies) are enforced uniformly, independent of the decision backend.

## 4.4 Off-Chain Router

We implemented a Python service that maintains a fresh snapshot of network state by polling RPC servers and consuming the chain registry. It exposes an HTTP API that relayers can query to obtain routes under user-provided constraints.

Implementation details:

- State refresh with backoff and health checks; caching to reduce RPC load.
- Route selection supporting shortest-path and policy-aware heuristics.
- Lightweight API for relayer consumption.

## 4.5 zkRouter Tooling

The zk tooling enables verifiable route computation:

- **zkProver**: Rust web service using RISC Zero; accepts a RISC-V ELF and parameters and produces a proof.
- **router-data-collector**: gathers registry and RPC data along with proofs [46, 4].
- **zk-router**: verifies network state, constructs the graph, and computes routes; compiled to RISC-V ELF for proving.

The verifier checks the proof and enforces policies, reducing the trust placed in the router runtime.

## 4.6 Evaluation and Benchmark Tooling

We built three complementary evaluation tracks:

- A SimPy model of Cosmos to study the capacity of network in congested scenarios
- A Cosmos network analysis tool to study systemic effects (additional links, failures, churn) across topologies.
- A real-world deployment with a multi-hop IBC load generator to measure throughput, success rate, and end-to-end latency.

# Chapter 5

# Evaluation

Our evaluation compares our system with a hub-and-spoke cross-chain message-passing system, the predominant approach for providing connectivity among Cosmos chains. We measure connectivity, decentralization, and scalability; evaluate required on-chain transaction fees and off-chain maintenance costs; and benchmark system components. In particular, we (i) load-test relayers (single and cooperative) for throughput and delivery rate under multi-hop traffic, and (ii) benchmark zkRouter's proving pipeline across devices, reporting per-stage runtimes and estimated cloud-provider proving costs. Finally, we analyze a DeFi use case to quantify the impact on asset liquidity when a protocol can trade tokens seamlessly across multiple chains.

## 5.1 Methodology

For our experiments, we examine the current network topology and effects of different parameters on several essential metrics: connectivity, decentralization, scalability, and costs.

**Data Collection:** We obtain a list of all blockchains in the Cosmos network, along with their RPC addresses, from the Cosmos Chain Registry repository [46]. For each blockchain, we connect to its RPC endpoint and collect data about the current state of the network. This includes connection, channel, and client information for the chain. We also query archival RPC endpoints to retrieve historical changes in connections over time. Finally, we reconstruct the current topology from the collected data. Historical IBC-connection

changes are captured up to January 2025, and other data are captured up to April 2025; both correspond to commit `addf89a9` in the Chain Registry repository.

**Simulation:** To evaluate scalability and StableSwap performance, we ran simulations using SimPy [3], due to the prohibitively high cost of running large-scale experiments on real blockchains.

**Relayer Load Testing:** To assess relayer performance, we generate multi-hop traffic at fixed rates and measure throughput and delivery rate under load. We send traffic for a bounded window (five blocks) and compute throughput as the number of packets divided by the elapsed time between the first packet observed and the last packet delivered. Details and results appear in Section 5.3.3.

**Zero-Knowledge Benchmarks:** We benchmark zkRouter's proving pipeline across diverse hardware (consumer laptop, cloud GPUs, and a CPU) and across five stages: verifying a storage Merkle proof, verifying a light block, computing a network-wide route, performing the BabyBear→BN254 curve transform, and generating a Groth16 SNARK that verifies the STARK. We report mean and standard deviation of runtime per stage; see Section 5.3.2.

**Cost Modeling:** For on-chain costs, we measure gas usage for manual multi-hop versus our system and amortize one-time channel setup over multiple messages. For proving costs, we estimate the dollar cost per 100M computation cycles using public cloud-GPU pricing and report results per device and provider.

## 5.2 Network Topology

### 5.2.1 Connectivity

As mentioned in Section 2.4, establishing a direct IBC connection between two chains often relies on an initial period during which light clients from the two chains are trusted to correctly exchange the Merkle root hashes of their chains' consensus states. This process typically involves manual intervention by the stakeholders/developers of the chains. As a

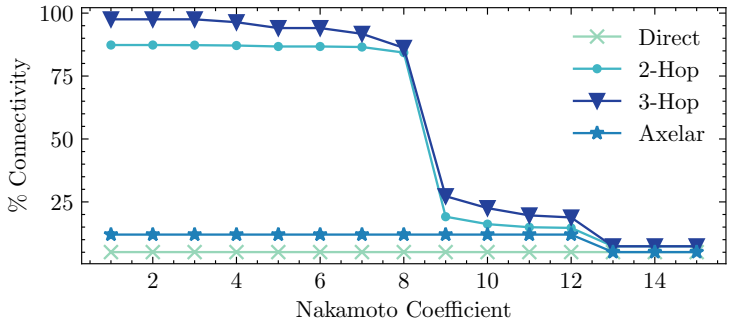



Figure 5.1: Connectivity achieved under a minimum Nakamoto-coefficient threshold (excluding lower-coefficient chains as intermediate hops).

result, increasing the connectivity between chains by adding direct IBC connections is generally a slow and measured process.

Therefore, one of the key metrics we measure is the degree of additional connectivity that the multi-hop IBC approach provides over single-hop IBC. We also compare with Axelar, which represents a hub-and-spoke model where the Axelar chain serves as a connecting hub for other chains. For this analysis, we use the IBC topology collected in April 2025.

Additionally, for multi-hop IBC and Axelar, we only use a chain as an intermediate hop if it meets a minimum security requirement as measured by the chain's Nakamoto coefficient [56]. Although the validity of the Nakamoto coefficient as a security metric is debated, it is simple to compute for any public chain and suffices for this comparative analysis. For Axelar, because the Axelar chain always serves as the intermediate hop, we apply the same security threshold to the Axelar chain itself.

Figure 5.1 shows that we can provide nearly 90% connectivity between chains while only allowing chains with a Nakamoto coefficient of 8 or higher to serve as intermediate hops. This compares to less than 15% connectivity for Axelar, which is limited by the number of direct IBC connections that Axelar chain has to other chains, and approximately 5% connectivity when relying only on direct connections. Connectivity drops precipitously at a Nakamoto coefficient of 9 and higher as that is a higher than the coefficient of some of the largest, most strongly connected Cosmos chains. These results illustrate the need for multi-hop message passing to realize the goal of providing connectivity between most chains.

### 5.2.2 Decentralization

Decentralization is one of the key defining properties of blockchains. Not only does decentralization offer stronger security compared to centralized approaches, it also provides higher availability as a chain can continue to function even if a minority of validators become unavailable. Our multi-hop routing, where any chain that meets a channel's security policy requirements can serve as an intermediate hop, offers a higher degree of decentralization compared to hub-and-spoke approaches where the failure of the hub chain can cause connectivity to revert back to the level offered by using only direct IBC connections.

To measure impact on decentralization, we perform an experiment where we measure the change in chain connectivity as we remove the most strongly connected chains, simulating the worst case scenario where those large chains become unavailable. Figure 5.2 shows results for 2-hop and 3-hop routing using both the current IBC topology, and an upgraded topology with additional connectivity between smaller chains.

For the current IBC topology, the loss of just the most strongly connected chain reduces connectivity to less than 40%. This is because most small chains are only connected to the largest chains, and the connectivity between small chains can be disrupted when a large chain fails even with multi-hop routing.

One approach to improve decentralization is to incentivize the creation of direct IBC connections between smaller chains. In our system, direct connections between two small chains with low Nakamoto coefficients can be bootstrapped using a multi-hop IBC connection with a high Nakamoto coefficient intermediate chain without requiring manual intervention by the chains' stakeholders/developers. Note that some applications may choose to not use connections bootstrapped in this way, as it relies on trusting the past security of a chain rather than its current security. In this analysis, we only consider applications that accept multi-hop bootstrapping of direct connections.

The upgraded lines in Figure 5.2 show the increase in decentralization by incentivizing each chain to create one additional direct connection to a random chain that it was not previously connected to with a Nakamoto coefficient of at least 6. Using 3-hop routing, we can still provide nearly 80% connectivity between chains when the most strongly connected chain becomes unavailable. In the unlikely event that the four most strongly connected chains become unavailable, we can still provide nearly 50% connectivity between chains using 3-hop routing.

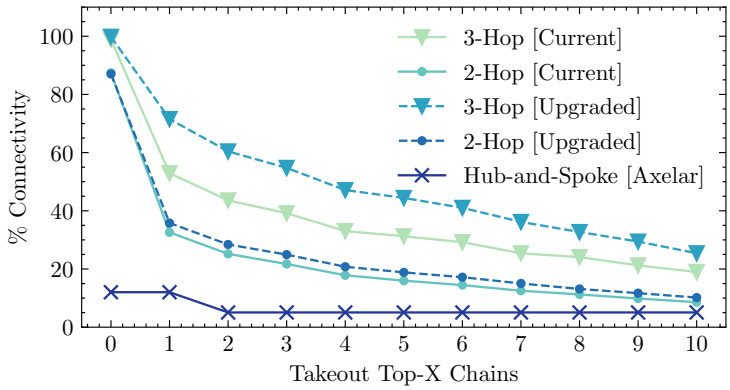



Figure 5.2: Change in connectivity after removing the top-X most connected chains.

Note that we do not measure the degree of decentralization of the relayer network. Although the relayer network serves an important purpose in our framework, the damage that can be caused by malicious behavior from a compromised relayer network is primarily limited to denial of service and higher fees as security properties can be verified by the source and destination chains.

### 5.2.3 Rate of Change in IBC Topology

One of the key roles of the relayer network is to monitor the IBC topology in order to compute routes when establishing a multi-hop channel. This can be challenging if there is a high rate of change in the IBC topology, which can include new connections being added, or existing connections being removed or becoming stale/frozen. A high rate of change can result in additional load on the relayers as they may need to recompute channels affected by topology change.

In this experiment, we measure the rate of changes in the Cosmos Network. It's essential to know how frequently a new connection is established between two chains or an existing connection has become stale/frozen. As relayers are the key players in the protocol that are computing the routes based on RPC requests to the network, it's necessary that this process doesn't lead to a situation where route computation is done with outdated state of the world.

Figure 5.3 shows the frequency of events that change the IBC topology over a period of three years. It shows that the number of topology changing events is relatively low, and although there is a slight growth trend, the rate of topology change events will likely

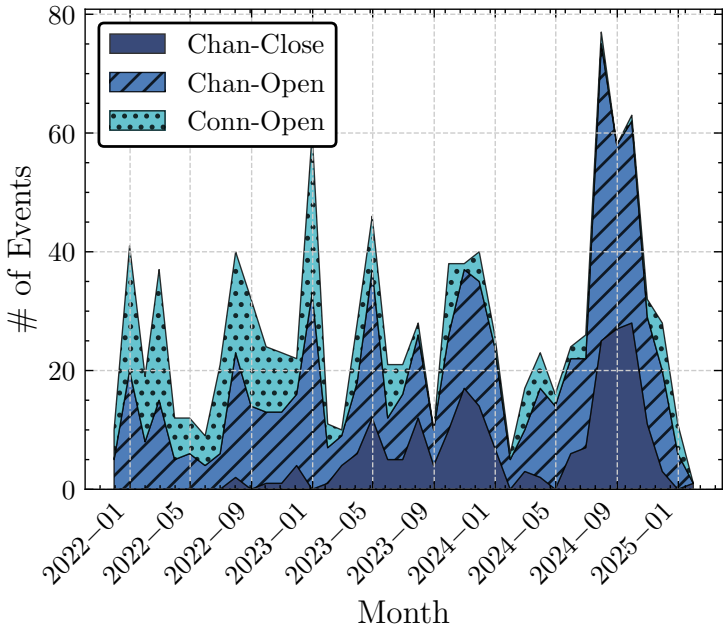



Figure 5.3: Connection-related changes in Cosmos Hub over time.

remain manageable for the foreseeable future unless some significant change occurs in the Cosmos ecosystem.

This clearly shows that it's only possible in the distant future that the overhead of querying the state of network becomes so high that it introduces a lag in the state of the world and demonstrates that our system will be effective even if network grows to thousands of blockchains. Even in that situation, our relayer network that cooperatively works on computing the routes and processing the packets would mitigate the problem and help the network scale.

## 5.3 System Performance

### 5.3.1 Scalability

Current hub-and-spoke based approaches require that messages are first sent from the source chain to the hub chain, and then sent again from the hub chain to the destination chain. This can limit scalability as the maximum rate of cross chain messages is bounded by

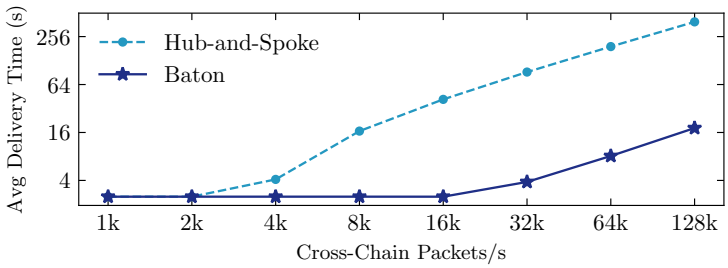



Figure 5.4: Message-delivery time: hub-and-spoke vs. our system.

the throughput of the hub chain. Most Cosmos chains, including Axelar, are built using the CosmosSDK [1] and use the Tendermint consensus algorithm [16]. Results from the original Tendermint whitepaper [16] show that it can sustain approximately 4000 transactions per second for a 64-node deployment. Therefore, the hub chain will experience high congestion once its throughput is saturated by cross-chain messages, which can result in high fees for hub chains with working fee markets, or failed transactions for those that do not.

Unlike hub-and-spoke approaches, our system does not need to send messages to intermediate chains. Rather, it only needs to update the consensus state of intermediate chains to generate multi-hop proofs. Furthermore, instead of a single hub chain, different multi-hop paths can be used for delivering cross-chain message. This spreads the load across chains that meet the security policy requirements.

To evaluate scaling, we simulate a network of 500 blockchains connected to a single hub and send cross-chain messages through it at varying rates. Packets are generated and assigned to paths based on a Zipf distribution. We measure the average end-to-end time from message inclusion on the source chain to receipt on the destination chain, comparing hub-and-spoke and Baton. Figure 5.4 shows the results: Baton handles up to 32k packets per second without latency impact compared to 4k for hub-and-spoke, and tolerates higher loads with reasonable latency. Congestion can be further mitigated in Baton if fee-minimizing routes are selected as applications will prefer to use low fee channels that avoid highly congested chains.

### 5.3.2 zkRouter Benchmarks

To assess the performance and efficiency of zkRouter, we benchmark each stage of the pipeline across a range of common compute devices.

**Devices.** We evaluate on the following hardware:

- **MacBook M3 Pro:** an ARM-based SoC with 18-core GPU (consumer-grade laptop).
- **NVIDIA T4:** a GPU commonly used in cloud environments, known for its balance between performance and cost-effectiveness.
- **NVIDIA L4:** a GPU optimized for AI inference workloads.
- **NVIDIA A100:** a high-performance GPU for large-scale AI and HPC applications.
- **Intel i7-1165G7:** a high-end consumer-grade CPU.

**Benchmarked stages.** On each device we measure:

- **Verifying a Merkle Proof (STARK scenario 1):** This step measures the time taken to verify the integrity of data using Merkle proofs, a critical operation within the zkRouter framework.
- **Verifying a Light Block (STARK scenario 2):** This scenario tests the verification of a light block, simulating a lightweight interaction with a blockchain's consensus state.
- **Estimating Router for Full Network (STARK scenario 3):** This scenario evaluates the full routing process across the network, incorporating all optimizations described in the zkRouter design.
- **Curve Transformation (BabyBear to BN254):** Evaluates the essential step of mapping to correct curve for SNARK proof
- **SNARK Groth16 Proof:** Evaluates generation of final SNARK proof

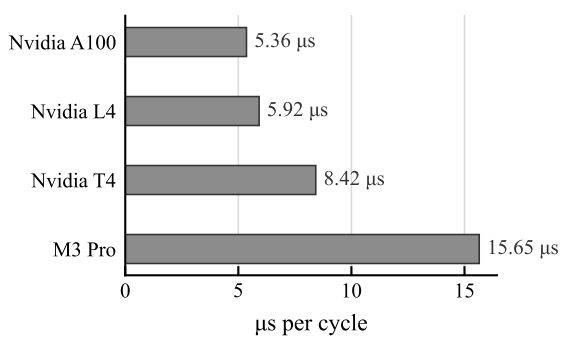
Table 5.1: Performance of zkRouter pipeline stages across compute devices

| Experiment | M3 Pro | NVIDIA T4 | NVIDIA L4 | NVIDIA A100 | Intel i7-1165G7 |
|---|---|---|---|---|---|
| STARK ***Storage Merkle Proof*** | $8.3 \pm 0.4$s | $4.5 \pm 0.1$s | $3.2 \pm 0.1$s | $2.9 \pm 0.1$s | $133.5 \pm 34.5$s |
| STARK ***1 Light Block*** | $656.5 \pm 30.4$s | $353.3 \pm 2.0$s | $248.3 \pm 1.3$s | $224.9 \pm 1.4$s | – |
| STARK ***Router*** | $1563.0 \pm 72.4$s | $841.2 \pm 4.7$s | $591.1 \pm 3.2$s | $535.4 \pm 3.3$s | – |
| Transform ***BabyBear to BN254*** | $32.6 \pm 0.3$s | $113 \pm 0.3$s | $83.2 \pm 0.1$s | $82.7 \pm 0.3$s | $130.3 \pm 32.0$s |
| SNARK ***Verifies STARK*** | $62.7 \pm 1.8$s | $7.2 \pm 0.4$s | $3.4 \pm 0.1$s | $3.0 \pm 0.1$s | $41.1 \pm 9.8$s |

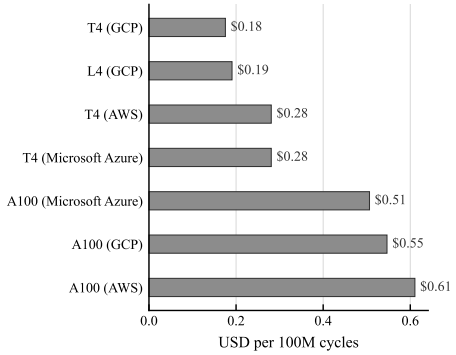
**Results.** The primary metric is the time to complete each stage on each device; Table 5.1 reports mean ± standard deviation across multiple runs. Proof computation time is reasonable given that the network graph changes infrequently. Route calculation has roughly the same number of computation cycles as a simple Merkle proof; a relayer therefore needs only seconds on an NVIDIA T4, or minutes on a strong CPU, to compute the proof.

The workload exhibits a high degree of parallelism, so performance improves with more powerful devices or by distributing work across a cluster. Figure 5.5a shows the time per computation cycle (µs) to generate a proof in the zkVM executing the RISC-V ELF.

We also estimate proving cost using cloud-GPU pricing. Figure 5.5b presents the cost for 100M computation cycles by GPU and cloud provider. In the most expensive setup, cost is approximately US$0.60 and amortizes further when the same proof is reused across transactions. Importantly, this cost is borne once for the whole network, not per pair of chains.[51]




(a) Time per computation cycle for proof generation (µs).




(b) Proving cost for 100M computation cycles by GPU and cloud provider (USD).

Figure 5.5: Proof performance and cost across devices and providers.

### 5.3.3 Relayer Benchmarks

To evaluate our relayer in both single and cooperative modes, we benchmark multi-hop message delivery under load. We generate cross-chain traffic at rates from 10 to 160 packets per second for five blocks and wait for the relayer(s) to deliver all packets. We report throughput as the total elapsed time from the first packet observed to the last packet delivered, divided by the number of packets; this reflects how many packets per second the relayer can assemble proofs for and, when necessary, update clients.

Figure 5.6 summarizes the results. Our Rust relayer scales nearly linearly with load, even in single-relayer mode. In contrast, the Go relayer is unable to keep up at higher rates, frequently crashing and because it does not implement a crash recovery mechanism, fails to resume delivery of previously observed packets after restart. At 160 packets per second, the single Rust relayer begins to saturate, while two cooperating relayers continue to scale, demonstrating the benefit of cooperative relaying. We were not able to test higher rates because the Cosmos chain RPC server became unstable under heavier load; we therefore used the maximum stable load in our environment.

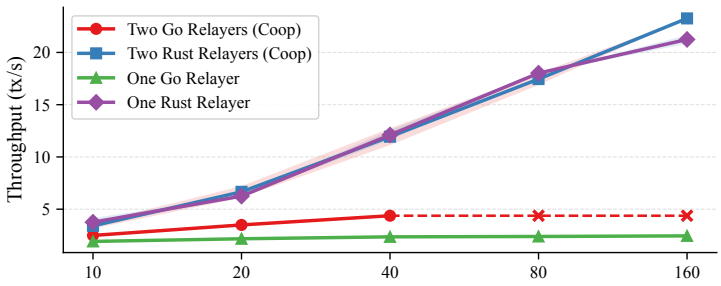



Figure 5.6: Throughput under multi-hop load by relayer configuration.

## 5.4 Costs

In this section, we compare the gas cost between sending a multi-hop message using our system and sending the same message manually using one or more direct IBC connections through the same intermediate chains. Our system has an initial setup cost for establishing a multi-hop channel for a source/destination pair. However, the created channel is usually long-lived and reused, so we amortized the cost of establishing the channel over multiple messages.

Figure 5.7 shows the cost of (1) manually sending the message using direct IBC connections, using our system (2) without including the cost of establishing the channel, and (3) with the cost of establishing the channel with an amortization factor of 1 message and 10 messages. As expected, the cost increases with higher hop count, but the rate of increase for our system is lower than manually sending the messages if the cost of establishing a channel is either not included or amortized over 10 messages.

We also measure the cost of performing on-chain verification of additional security policies. For this experiment, we send 12 messages along a 2-hop channel where the intermediate chain has 4 validators. We compare the cost of sending these messages against the cost of sending the same messages without the security policies. Since the gas cost depends entirely on the amount of data being processed, we found that the average additional gas cost per validator per hop is 15459 gas units. For the current state of the Cosmos network, where the network diameter is 3 and average number of validators is 54, this would result in $15459 * 3 * 54 = 2504358$ gas units which would be less than 0.076 USD using current price of ATOM, the gas token for many Cosmos chains.

These results show that our approach is cost effective, with fees that are low enough such that users will unlikely choose to perform manual single-hop transfers for fee-related reasons. For off-chain proving costs, see Figure 5.5b in Section 5.3.2.

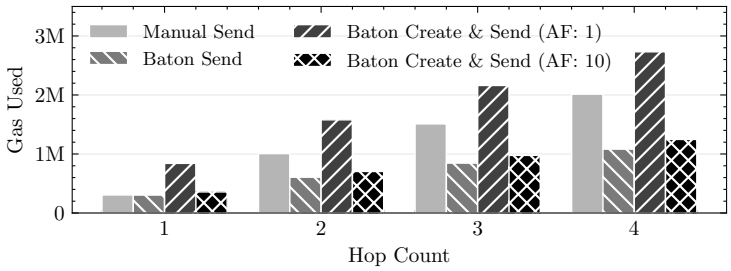



Figure 5.7: Gas costs: our system vs. manual multi-hop.

## 5.5 Application Impact

### 5.5.1 Use Case: StableSwap

As described in Section 3.5, we implemented a token swap protocol that breaks down a large swap into multiple smaller swaps performed on other chains using our customized routing. The protocol takes advantage of available liquidity across multiple chains to provide the user with a better price for their swap.

This experiment compares the financial gains of our swap protocol when trading USDT for USDC against different baselines, where USDC and USDT are both stablecoins pegged to USD. The available liquidity of USDC and USDT across Cosmos chains were collected in January 2025, and we scale the total available liquidity proportionately to $10 billion USD.

Figure 5.8 shows the amount of USDC received from trading a given amount of USDT using the different approaches. The ideal line illustrates perfect 1:1 pricing, which is the expected ratio from trading two stablecoins pegged to the same currency. The unified line illustrates the case where all available liquidity across all Cosmos chains is somehow unified to a single chain. The results are very close to the ideal line even when trading $100 million USDT tokens.

The 0-hop line illustrates the case where cross-chain trades are not performed, and the trade is satisfied only using the available liquidity of the local chain. For this experiment, we selected the Cosmos chain with the most available liquidity as the local chain to provide the best case scenario for the 0-hop approach. Trading $100 million USDT would return less than $50 million USDC because of the limited liquidity available in the local chain. Going from 0-hop to 1-hop significantly increases the available liquidity and the return in USDC from trading USDT. The 3-hop results are nearly identical to the unified results.

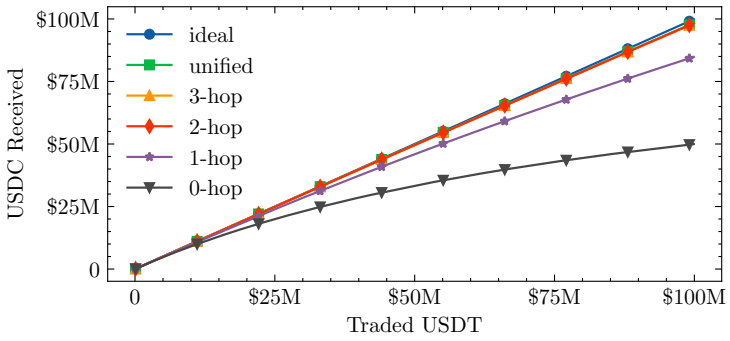



Figure 5.8: StableSwap performance between USDT and USDC.

One might argue that such large trades are not commonly performed on decentralized exchanges. However, larger trades and higher trade volumes often occur during significant market volatility. For example, decentralized exchanges experienced all-time high trade volumes on March 11, 2023 at $20 billion USD during the USDC depeg event as USDC holders exchanged their USDC for other stablecoins even when there was limited available liquidity [36].

# Chapter 6

# Conclusion

This thesis investigated policy-driven cross-chain routing over multi-hop IBC. We studied how to compute and enforce routes that satisfy application-defined security, cost, and latency requirements without introducing a new centralized trust anchor. We presented a modular stack that separates (i) policy specification and enforcement in the transport layer, (ii) route computation in a relayer control plane, and (iii) execution in a relayer data plane. Within this architecture we instantiated three complementary routing methods:

1. **Single-Relayer routing:** low latency and low on-chain cost; strictly enforces security policies at the destination via an IBC light client and Merkle proofs.
2. **zkRouter:** off-chain route computation with an on-chain-verifiable zero-knowledge proof that the chosen path satisfies user policies and matches the attested chain state at specified heights.
3. **Relayer Network:** a collaborative overlay with staking, slashing, and fee-sharing that provides cryptoeconomic guarantees for optimizing policies (e.g., fee minimization), distributes operational load, and aligns incentives.

Using the April 2025 IBC topology, our evaluation shows that the system provides more than 80% connectivity between chains even when intermediate chains must have a Nakamoto coefficient of 9 or higher, compared to less than 15% when using Axelar (a hub-and-spoke-based approach). Our results also show significantly higher decentralization and scalability compared to the baselines.

Interoperability at Internet scale requires routing that is policy-aware, verifiable, and practical. This thesis shows that it is possible to meet these goals without a new centralized hub: strictly enforce security policies at the destination, use ZK proofs or cryptoeconomic mechanisms for optimizing preferences, and keep verification cheap. Future work includes richer optimizing policies (e.g., MCSP), adaptive policy updates, and reducing on-chain verification costs. Overall, this work moves inter-blockchain communication closer to Internet-grade interoperability and aims to provide blockchains with the same type of transformative growth that the Internet achieved by seamlessly connecting different networks.

# References


[1] Cosmos sdk. https://github.com/cosmos/cosmos-sdk.

[2] Inter-blockchain communication. https://www.ibcprotocol.dev/.

[3] Simpy. https://gitlab.com/team-simpy/simpy/.

[4] Tendermint rpc docs. https://docs.tendermint.com/v0.34/rpc/.

[5] 7 Cross-Chain Bridge Vulnerabilities Explained — Chainlink. *https://chain.link/education-hub/cross-chain-bridge-vulnerabilities*, September 2024.

[6] 0x. 0x official website. https://0x.org/, 2025. Accessed: 2025-01-31.

[7] 0x Project. 0x white paper. Technical report, 0x Project, January 2025. Accessed: 2025-01-31.

[8] ARK Association. Ark official website. https://ark.io/, 2025. Accessed: 2025-01-31.

[9] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM*, 45(1):70–122, 1998.

[10] André Augusto, Rafael Belchior, Miguel Correia, André Vasconcelos, Luyao Zhang, and Thomas Hardjono. Sok: Security and privacy of blockchain interoperability. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3840–3865, 2024.

[11] Axelar. Axelar: Connecting applications with blockchain ecosystems. https://www.axelar.network/whitepaper [Accessed 22/08/2024], Jan 2021.

[12] Rafael Belchior, Jan Süßenguth, Qi Feng, Thomas Hardjono, André Vasconcelos, and Miguel Correia. A brief history of blockchain interoperability. *Commun. ACM*, 67(10):62–69, September 2024.

[13] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *ACM Comput. Surv.*, 54(8), October 2021.

[14] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Technical Report 2018/046, IACR ePrint Archive, 2018.

[15] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 103–112, 1988.

[16] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains.* PhD thesis, University of Guelph, 2016.

[17] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. In *arXiv:1807.04938*, 2018.

[18] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. Whitepaper, 2014.

[19] Vitalik Buterin. Chain interoperability. Technical report, R3, September 2016. Accessed: 2025-01-31.

[20] Chainlink. Introducing the cross-chain interoperability protocol (ccip) for decentralized inter-chain messaging and token movements. https://blog.chain.link/introducing-the-cross-chain-interoperability-protocol-ccip/ [Accessed 21/08/2024].

[21] João Otávio Chervinski, Diego Kreutz, Xiwei Xu, and Jiangshan Yu. Analyzing the performance of the inter-blockchain communication protocol. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 151–164, 2023.

[22] Cosmos Hub Community. Ibc relayer gas cost restitution plan. Forum post, 2023.

[23] DefiLlama. Defillama. https://defillama.com/ [Accessed 13/09/2024].

[24] Donghui Ding. Interchain : A framework to support blockchain interoperability. 2018.

[25] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[26] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[27] DFINITY Foundation. Chain key cryptography. https://internetcomputer.org/how-it-works/#Chain-key-cryptography, 2025. Accessed: 2025-01-31.

[28] Christopher Goes. Relayer algorithms. https://github.com/cosmos/ibc/tree/main/spec/relayer/ics-018-relayer-algorithms [Accessed 16/09/2024].

[29] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–304, 1985.

[30] David Goosenberg. Osmosis: The interchain dex. Blog, 2024.

[31] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016.

[32] Panpan Han, Zheng Yan, Wenxiu Ding, Shufan Fei, and Zhiguo Wan. A survey on cross-chain technologies. *Distrib. Ledger Technol.*, 2(2), June 2023.

[33] Thomas Hardjono. Blockchain gateways, bridges and delegated hash-locks, 2021.

[34] Ziad Hussein, May A. Salama, and Sahar A. El-Rahman. Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms. *Cybersecurity*, 6(1):30, 2023.

[35] Hyperlane. Hyperlane official website. https://hyperlane.xyz/, 2025. Accessed: 2025-01-31.

[36] Ledger Insights. Usdc depeg highlights importance of fed to banking system. https://www.ledgerinsights.com/usdc-depeg-fed-federal-reserve/.

[37] Interchain Foundation. Channel upgradability and fee middleware. Blog, 2023.

[38] Interchain Foundation. Ibc protocol developer portal, 2024.

[39] Interchain Standards Team. Ibc protocol specifications (ics). GitHub cosmos/ibc, 2023.

[40] Luo Kan, Yu Wei, Amjad Hafiz Muhammad, Wang Siyuan, Ling Chao Gao, and Hu Kai. A multiple blockchains architecture on inter-blockchain communication. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 139–145, 2018.

[41] Jae Kwon and Ethan Buchman. Cosmos. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md, Jan 2019. [Accessed 22/08/2024].

[42] LayerZero. Layerzero official website. https://layerzero.network/, 2025. Accessed: 2025-01-31.

[43] Li Li, Jiahao Wu, and Wei Cui. A review of blockchain cross-chain technology. *IET Blockchain*, 3(3):149–158, 2023.

[44] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, and Yih-Chun Hu. Hyperservice: Interoperability and programmability across heterogeneous blockchains. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 549–566, New York, NY, USA, 2019. Association for Computing Machinery.

[45] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Self-published whitepaper*, 2008.

[46] Cosmos Network. Cosmos chain registry, 2024. Accessed: 2024-08-16.

[47] Cosmos Network. Ibc specification: Fee payment (ics-029), 2024. Accessed: 2024-12-07.

[48] Cosmos Network. Ibc specification: Multi-hop communication (ics-033), 2024. Accessed: 2024-08-16.

[49] Omnity Network. Omnity network official website. https://www.omnity.network/, 2025. Accessed: 2025-01-31.

[50] Omnity. Omnity: The cross-chain endgame for a modular blockchain world. Technical report, Omnity Network, January 2025. Accessed: 2025-01-31.

[51] Paperspace. 2023 gpu pricing comparison: Aws, gcp, azure & more, 2023. Accessed: 2024-08-16.

[52] Maksym Petkus. Why and how zk-snark works: Definitive explanation. arXiv:1906.07221, 2019.

[53] Aleksei Pupyshev, Elshan Dzhafarov, Ilya Sapranidi, Inal Kardanov, Shamil Khalilov, and Sten Laureyssens. Susy: a blockchain-agnostic cross-chain asset transfer gateway protocol based on gravity, 2020.

[54] Ilham A. Qasse, Manar Abu Talib, and Qassim Nasir. Inter blockchain communication: A survey. In *Proceedings of the ArabWIC 6th Annual International Conference Research Track*, ArabWIC 2019, New York, NY, USA, 2019. Association for Computing Machinery.

[55] RISC Zero. Risc zero developer docs v3, 2025.

[56] Balaji S. Srinivasan. Quantifying decentralization, 2017. Accessed: 2023-10-25.

[57] Drew Stone. Trustless, privacy-preserving blockchain bridges, 2021.

[58] Shankar Subramanian, André Augusto, Rafael Belchior, André Vasconcelos, and Miguel Correia. Benchmarking blockchain bridge aggregators. In *2024 IEEE International Conference on Blockchain (Blockchain)*, pages 37–45, 2024.

[59] ZhongAn Tech. Anlink whitepaper. https://alicliimg.clewm.net/049/389/1389049/1484820492640c2baf37ea3e4f9fd77bd52c2a1e9bbbe1484820484.pdf, 2017. Accessed: 2025-01-31.

[60] Hui Wang, Yuanyuan Cen, and Xuefeng Li. Blockchain router: A cross-chain communication protocol. In *Proceedings of the 6th International Conference on Informatics, Environment, Energy and Applications*, IEEA '17, page 94–97, New York, NY, USA, 2017. Association for Computing Machinery.

[61] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. https://assets.polkadot.network/Polkadot-whitepaper.pdf, Nov 2016.

[62] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 3003–3017, New York, NY, USA, 2022. Association for Computing Machinery.

[63] Ryan Zarick, Bryan Pellegrino, Isaac Zhang, Thomas Kim, and Caleb Banister. Layerzero. https://layerzero.network/publications/LayerZero_Whitepaper_V2.1.0.pdf, Jan 2024.

[64] RISC Zero. Risc zero api documentation, 2024. Accessed: 2024-08-16.