

Secure and Efficient Message Routing and Delivery Across Blockchain Networks

by

Solomon Davidson

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, 2024

© Solomon Davidson 2024

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

The increasing demand for blockchain applications and services has led to a surge of new public blockchains to support this growth. Although the availability of additional blockchains enables the support of more applications and users, it also fragments liquidity, which can create a challenging user experience. Cross-chain communications can tackle this issue by providing interoperability for smart contracts across blockchains. However, current cross-chain delivery solutions either only support message passing between directly connected chains, which significantly limits their connectivity, or are heavily centralized, requiring messages to be routed through a single hub chain.

In this thesis, we present Baton, an interoperability protocol for efficient and secure cross-chain message routing and delivery. Baton is built on the Inter-Blockchain Communication protocol (IBC). IBC is a protocol for general message passing between blockchains. Developers independently connect chains over IBC as they see fit, and Baton can route messages across the IBC topology to allow pairs of chains to communicate. This reduces the need for additional trusted connection setup periods, which are vulnerable for exploitation. Routes are computed off-chain and verified on-chain to minimize the cost of route computation. Unlike IBC, where users must manually specify routes, users can send cross-chain messages just by specifying a destination chain. Baton enforces cryptoeconomic incentives to ensure that near-optimal routes, based on a user specified metric, are maintained.

Unlike packet routing in traditional networks where the packets have to traverse one or more transit networks, Baton delivers messages directly from the source to the destination and only uses the state of the intermediate chains to construct a proof to verify the authenticity of the message. Baton can also include information about chain security properties within these proofs, such as how stake is distributed. These user-specified security policies disallow messages from being sent along routes with less secure blockchains. Baton can send messages along multiple disjoint paths that connect the same chains. By verifying that messages sent along disjoint paths are the same, Baton can detect compromised chains and fabricated messages.

Cosmos Hub and Axelar are hub chains that connect to other blockchains, and that enable wide-spread interoperability by forwarding messages between pairs of chains. If Cosmos Hub or Axelar become unavailable, only 32.8% and 48.2% of chain pairs can communicate, respectively. In contrast, Baton can maintain connectivity between 57.5% of chain pairs when the top 10 most connected blockchains become unavailable. Our evaluation shows that Baton can deliver up to 20.6% and 3.26% more messages than if blockchains were arranged in a mesh topology or a hub-and-spoke topology respectively, given the same number of cross-chain proof transactions.

Acknowledgements

I would like to thank my supervisor Dr. Bernard Wong for his support and guidance throughout my master's program, as well as with this research and my thesis. It has been a great experience to be able to work with him. I would also like to thank Amin Rezaei, who also worked with me in developing Baton. I am grateful for all the feedback I received as part of the Shoshin group over the course of writing this thesis.

I would also like to thank my brother and sister Samuel and Simone Davidson, as well as my parents Ray and Janet Davidson. Their continual love and support has been a blessing. Finally, I would like to thank my beautiful wife Aileen Agada-Davidson. Her love and encouragement has been wonderful, and I could not have asked for a better partner.

*“Commit everything you do to the LORD. Trust him, and he will help you.”
-Psalms 37:5 (NLT)*

Table of Contents

Author’s Declaration	ii
Abstract	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	xi
1 Introduction	1
2 Background	6
2.1 Blockchain Preliminaries	6
2.2 Gas, Tokens and Wallets	7
2.3 Leader Election, Consensus Protocols and Finality	8
2.3.1 Leader Election	8
2.3.2 Consensus on Blockchain State	9
2.3.3 Finality	10
2.4 Blockchain Interoperability	11
2.5 IBC	12
2.5.1 Overview	13

2.5.2	Client Updates	16
2.5.3	Merkle Proofs	17
2.5.4	Proofs of Inclusion and Exclusion	18
2.5.5	Channels	20
2.5.6	Multi-Hop Proofs	21
2.5.7	Relayer Incentives	22
2.6	Traditional Networking Solutions and Blockchains	22
2.6.1	Packet Routing	23
2.6.2	Data Integrity	24
2.6.3	Name Resolution	25
3	Related Work	27
3.1	Relays	27
3.2	Side Chains	29
3.3	Notary Schemes	29
4	System Design	32
4.1	Overview	32
4.2	Baton User	36
4.3	Relayer	39
4.4	Baton Participant	41
4.4.1	Processing Channel Requests	41
4.4.2	Specifying Security Policies	42
4.4.3	Creating A Channel	43
4.4.4	Adding New Security Policies	44
4.4.5	Module Policy Requirements	45
4.5	Validating IBC Channels as Part of Baton	46
4.5.1	Validation Governance Vote	46

4.5.2	Incentives for Correctly Identifying Blockchains	47
4.5.3	Validating Channels With Recurring Chains	47
4.5.4	Rewarding the Participant	48
4.5.5	Limitations of Rewarding Participants for Channel Use	49
4.6	Splitting Messages Along Multiple Disjoint Paths	50
4.6.1	The Security Risk Compromised Chains Pose	50
4.6.2	The Multi-Path Handshake Algorithm	50
4.6.3	Catching Misbehaviours	53
4.6.4	Limitations When Selecting Disjoint Paths	53
5	Evaluation	55
5.1	Experimental Setup	56
5.2	Decentralization	58
5.3	Congestion	60
5.4	Single-Hop Channels Vs. Multi-Hop Channels	61
5.4.1	Establishing Channels	62
5.4.2	Sending and Verifying Messages	64
5.5	Including Policy Proofs	66
5.6	Implication of Network Topology Gas Cost	68
6	Conclusion	71
	References	73

List of Figures

1	Two blockchains connected using IBC. The relayer forwards updates between the light clients within both chain's IBC module. The connection tracks the state updates between pairs of light clients. The channel pairs a port and connection on one blockchain with a port and connection on the other blockchain. Channels also enforce exactly one delivery of all packets sent along it. Applications on either chain communicate by sending packets along a channel.	14
2	Example Merkle tree	17
3	Merkle proof for "Key2/Value2"	18
4	Channel end data structure.	19
5	IBC-enabled blockchains store the consensus states of its neighbour and the corresponding connection end information in its key/value store. This means that the consensus state of a blockchain can be used to prove it is connected to a neighbouring chain, and that it knows that neighbour's consensus state. Ultimately, the source chain's consensus state is used to prove the existence of the message being sent.	21
6	An overview of Baton's infrastructure	33
7	User will request a channel that connects the source and destination chain, that meets certain security requirements and that is optimized according to a specified metric. A protocol participant will compute an appropriate route offline.	34
8	The protocol participant will create an IBC channel that corresponds to the computed route.	35

9	The user can now send their message on the newly created channel. Seeing that blockchains along the channel must adhere to security policy X , the relayer will append proofs of property X to the original multi-hop proof. . .	36
10	Baton user's flow of operations	37
11	The cost matrix on the left is used to compare the cost of Channel-1 and Channel-2. Channel-1 has cost 4 and Channel-2 has cost 5. Even though both channels connect the same endpoints, Channel-1 will be selected. . . .	38
12	Channel request format	42
13	Syntax for channel versions in Baton. The application version is first, followed by a forward slash, then by zero or more security policies. Policies are also delimited by forward slashes. Each policy as a policy type, followed by a colon, followed by a requirement.	42
14	Relayer's view when relaying messages along a channel	44
15	Life cycle of validating a Baton channel.	46
16	Two channels, both sharing the same blockchain as the third intermediate chain.	47
17	Light clients for the validated channel are updated such that blockchains B and E have the same height stored for C and C' respectively.	48
18	Overview of multi-path handshake algorithm	51
19	Percentage of blockchain pairs that are still connected even with the top X most popular blockchains removed. X ranges from 0 to 10.	59
20	The maximum amount of transaction on a particular block height for the most congested blockchain. We send a total of 50000 messages, where the send rate per blockchain pair ranges from one message every 25000 milliseconds, to a message every 1563 milliseconds. Each chain's block time is 4 seconds.	61
21	Average block time required to establish channels between pairs of chains. Since each hop is created independently, the best case scenario for standard IBC is to create every single-hop channel concurrently.	62
22	Gas required to establish channels between pairs of chains.	63
23	Average block time required send a simple message between two chains. . .	64

24	Average gas used by relayers when sending a simple message between two chains.	65
25	Blockchain network topologies.	68
26	The average amount of receive and acknowledgment messages processed for every update client message. We send messages uniformly at random between pairs of chains. In each scenario, there were 40 relayers relaying packets.	69

List of Tables

1	Types of security policies	43
2	Timeout parameters used for CometBFT consensus protocol.	57
3	Expected additional cost per message for validator count policy.	67
4	Number of 1-hop, 2-hop and 3-hop channels for each topology. Also includes the expected number of hops and expected gas cost per message if messages are uniformly sent between chain pairs.	70

Chapter 1

Introduction

The number of public blockchains has grown at an increasing rate year-over-year. While early blockchains were used purely as immutable ledgers for transactions between peers [43], newer blockchains allow users to develop and interact with Turing complete programs called smart contracts [10]. Blockchain applications have therefore 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 end-users and institutions, as each chain can transact independently.

It is beneficial for blockchains to interoperate, as smart contracts on different chains can be used in conjunction to yield better results for the end-user. Consider decentralized exchanges (DEXs), which are smart contracts that allows blockchain users to exchange one digital asset for another [3]. The exchange rate for these assets is usually determined by the relative amount of each asset on that particular chain. Liquidity is a measure of the quantity of a digital asset stored in a DEX, and it indicates a user’s ability to buy and sell that asset without drastic fluctuations in pricing. A user may want to use a DEX where a particular asset has a lot of liquidity, as they can make larger exchanges without changes in prices. It can be problematic for a user to be confined to one DEX, as changes in liquidity can cause assets to be volatile. Users can mitigate this by transferring digital assets, and other arbitrary data, between DEXs as needed. That said, blockchain interoperability has grown increasingly important as the number of blockchains has increased. We will use the following definition of blockchain interoperability which is presented by Ren et. al: [49] *“the ability to flexibly transfer assets, share data, and invoke smart contracts across a*

mix of public, private, and consortium blockchains without any changes to the underlying blockchain systems”.

Ren et. al define four system characteristics that categorize different aspects of blockchain interoperability: Decentralization, Locking Mechanism, Verification Mechanism and Communication [49]. Decentralization refers to a protocol operating with multiple independent users and organizations. The locking mechanism refers to how assets are securely managed during cross-chain operations. For example, when transferring tokens from one blockchain to another, the original tokens might be sent to a smart contract operated account on the source chain. A synthetic version of that token can then be minted on the destination chain. The verification mechanism refers to how one chain knows what has happened on another chain. When considering the previous example, the destination chain needs to verify that the original tokens have been sent to the smart contract’s account before minting the synthetic tokens. Finally, communication describes the direction of communication as either unidirectional or bidirectional.

It matters how these aspects of an interoperability protocol are designed and implemented, as a vulnerable interoperability protocol can be exploited to steal assets from honest users. Securing interoperability is difficult however, as more than 2.8 billion dollars have been stolen from bridges [1]. Many of the exploits used social engineering techniques to acquire the small number of keys needed from third parties to take control of the bridge’s smart contract. Therefore if the correctness of cross-chain operations depends on a third party, the third party becomes an attack vector that can be exploited to produce invalid cross-chain operations. We can mitigate this problem by not having the correctness of cross-chain operations depend on third parties.

The Inter-Blockchain Communication (IBC) protocol was designed to avoid some of the security issues that are present in other protocol designs, such as smart contract-based bridges. While many interoperability protocols are individualized to specific pairs of chains [49], IBC is designed after TCP/IP [21]. A light client is an algorithm that can verify the correct execution of a consensus protocol by validating block headers or signatures. Light clients can also verify the Merkle proof of inclusion for committed messages in the respective blockchain’s state. An IBC relay is an entity that forwards messages and light client updates between blockchains. Relayers cannot alter messages, as they are signed by the source chain’s validators and verified by the destination chain. IBC allows chains to communicate by having them maintain light clients for each other. Assuming the light client for the other chain has been correctly bootstrapped, then the security of IBC messages should be equivalent to the security offered by the other chain. Therefore, the correctness of cross-chain operations within IBC is dependent solely on the security guarantees of the chains involved. Furthermore, IBC allows for bidirectional communication and is

extensible.

While there are over 100 active chains that support IBC, most chains only maintain a few IBC connections to popular chains. A smart contract can only communicate over IBC with neighbouring chains to whom they are directly connected. If a user, such as a chain operator, would like to communicate between chains that do not already have an IBC connection, they would need to bootstrap a new connection manually. This requires an initial trusted period during which the user obtains the consensus states of the chains they wish to connect, and the light client builds on that initial state. To reduce the number of opportunities where an IBC connection can be compromised, it is best to limit how often new IBC connections are established. When looking at the interoperability landscape, Cosmos Hub [18], Axelar [5], and Avalanche [50] solve this problem by becoming a central hub blockchain to which sovereign chains can connect. All cross-chain messages must pass through the hub. Unfortunately, hubs are a centralized solution and are a single point of failure.

Alternatively, IBC allows messages to be sent across channels that span multiple connections. When sending messages on multi-hop channels, the light client for each chain along the route must be updated to track the state of the previous chain. Following that, relayers must submit state proofs of the intermediate blockchains to the destination chain so that the receiving smart contract can prove the existence of the message on the source chain. Sending cross-chain messages is manual, tedious and requires consulting external sources to determine which channels lead to which chains. While IBC channels have IDs, these IDs do not tell the user what the destination blockchain is. The user must retrieve this information from an external resource such as Mintscan.io [22]. Since channels can traverse different intermediate blockchains before arriving at the same destination chain, users must also be aware of which intermediate chains a channel spans. Otherwise, blockchain users may accidentally choose a channel through intermediate chains that are malicious or compromised.

In many ways, cross-chain communication today is facing the same problems that the Internet had faced and solved for inter-network communication. 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.

In this thesis, we present Baton, an interoperability protocol for efficient and secure cross-chain message routing and delivery. Baton is built on top of IBC, and is therefore compatible with any IBC enabled blockchain. Rather than requiring users to create new connections between chains, Baton can deliver messages across the existing topology of IBC enabled chains. By routing cross-chain messages along near-optimal paths, Baton

makes it easier for all blockchain pairs to interoperate. Baton can prove that blockchains along a route maintain certain security properties, therefore providing verifiable baseline security guarantees. Furthermore, Baton can use multiple disjoint routes to detect and circumvent compromised blockchains.

One of our contributions is that Baton can verify the security properties of blockchains along a route. By doing so, Baton disallows messages from being sent along channels that traverse less secure blockchains. Users will assign security policies to channels and applications running on Baton chains. This notifies IBC relayers to submit, along with the original message, proofs of these security properties for all intermediate blockchains along the route. When the message is delivered, the destination chain will validate these additional proofs. The results from verifying these proofs can be cached and used for later messages, as we show that relevant security properties often change infrequently. Thus, the cost of including policy proofs is insignificant when amortized across cross-chain messages.

Another contribution is Baton’s ability to route cross-chain messages along near-optimal paths. Participants in the protocol create IBC channels, and register the endpoint chains in the blockchain’s state. Thus when users specify the destination chain by name, Baton will have the message routed to the correct blockchain. This eliminates the need for users to consult external sources to learn which channel leads to which chain. Furthermore, protocol participants are incentivized to ensure their channels are optimal, as they are compensated whenever others use channels they have established as part of the Baton protocol. Participants in the protocol compute routes off-chain, and verify these routes on-chain when creating multi-hop IBC channels. Similar to how DNS servers must be consulted in order to learn which domain names map to which IP addresses, external sources must be consulted to determine which chain IDs map to which blockchain RPC endpoints. When a blockchain is being bootstrapped, governance voting, which is when stakeholders vote on important chain updates, can be used to establish mappings between chain IDs and channels. We demonstrate that Baton incentivizes rational stakeholders to assign blockchains their correct IDs. Beyond bootstrapping, Baton uses existing channels as out-of-band communication to prove that newly created channels traverse particular blockchains.

In addition to routing messages to the correct destination, Baton also uses proofs to ensure the integrity of messages is maintained upon delivery. Every adjacent pair of chains along a route maintains light clients for each other, and each chain’s consensus state can be used to prove the existence of the consensus state of the chain before it. By recursively constructing Merkle proofs, the destination chain is eventually able to prove the existence of the source chain’s consensus state. It can then use the source chain’s consensus state to verify the Merkle inclusion proof of a message, and process the message accordingly.

A third contribution is a mechanism that allows Baton to detect compromised blockchains. It has been shown that it is hard, if not impossible, to detect misbehaviour on one blockchain solely by observing another blockchain [53]. This is true because malicious validators can run a consensus protocol correctly to commit incorrect state transitions to a blockchain. Validating state transitions is expensive and requires verifying every transaction in a blockchain. That said, most interoperability protocols, including IBC, are susceptible to these types of attacks. Furthermore, IBC does not provide a standard mechanism to mitigate these attacks. Baton lowers the risk of these types of attacks by sending messages along multiple disjoint routes. Baton can reconcile multiple channels as connecting the same endpoint blockchains. This is done using an algorithm we call the “multi-path handshake”. If a blockchain along one of the channels becomes compromised in the future and fabricates a message, Baton can send that message again along a disjoint channel. By comparing both messages, the destination chain will learn that a blockchain along one of the channels is compromised.

We evaluate Baton against three blockchain topologies: A natural topology, a mesh topology and a hub-and-spoke topology. We demonstrate that Baton is decentralized, as Baton can maintain connectivity between 57.5% of chain pairs when the top 10 most connected blockchains are made unavailable. This beats current approaches, as only 32.8% and 48.2% of chain pairs can communicate if Cosmos Hub or Axelar, respectively, become unavailable. We show that Baton introduces far less congestion when compared to the current deployment of IBC on Cosmos based blockchains. We argue that, when given cryptoeconomic incentives, participants in the protocol will generate near-optimal routes between chains. We show that the multi-hop channels that Baton uses are faster and more cost effective than traditionally used single-hop channels. We show that the additional gas cost for including relevant security policy proofs is small. A proof that three intermediate chain have ≥ 100 validators, for example, costs $\sim \$ 0.029$ USD when amortized over 20 messages, and $\sim \$ 0.000232$ USD when amortized over 2500 messages. In our evaluation, we show that Baton can deliver up to 3.26% and %20.6% more messages given the same number of light client updates when compared to a hub-and-spoke and mesh topology respectively. This demonstrates that Baton is no less efficient than prevalent approaches.

Chapter 2

Background

In this section, we present the necessary background for Baton. We start by describing the fundamentals of how blockchains work. Following that, we introduce the concept of blockchain interoperability. Finally, we explain how the Inter-Blockchain Communication (IBC) interoperability protocol works, as Baton is built on top of IBC.

2.1 Blockchain Preliminaries

Online activities that involve payments and other financial operations must be performed securely to ensure that all parties are paid the agreed upon amounts. Unfortunately, a lot of these sensitive activities rely on the integrity of trusted third parties, such as banks and other private companies. Blockchains were developed to remedy this problem by removing the need for trusted third parties [43]. Instead of having a single entity facilitate and validate operations, many independent parties can attest to the validity of the operations. This decentralization makes it much more difficult for corrupt parties to censor, manipulate or otherwise compromise financial operations. Blockchains have since been adapted to not just ensure the integrity of financial operations, but of all types of data in general. Consequently, blockchains are useful because they provide a way to implement immutable databases or ledgers.

A blockchain is fundamentally an ordered, distributed ledger that is managed by entities called validators. When someone would like to add data to the ledger, they will broadcast that data to the validators. Validators periodically run a leader election protocol to take turns adding to the ledger. When it is a validator's turn, they will aggregate some of the

data they have seen into a structure called a block. That block will then be broadcast to the other validators. Each validator uses a shared set of rules to verify that the data within the block is allowed to be included in the ledger. An example rule might be that a user cannot send another user more money than they have. A validator will either append that block to their local view of the ledger, or they will omit the block. Following that, validators will run a consensus protocol to come to an agreement on the new state of the ledger. As this process repeats, more and more blocks are chained on to the ledger. This is what gives them the name of “blockchains”. When a blockchain can be accessed by anyone, it is called either a public or a permissionless blockchain. Otherwise, it is called a private or a permissioned blockchain.

Blockchains borrow from database terminology in that the individual chunks of data that users would like to have included in the ledger are called transactions. A transaction can be any arbitrary data, and it is up to the blockchain’s validators to correctly interpret the meaning of a transaction. Processing a transaction is fundamentally the same as updating the ledger.

Users can submit source code as transactions. Source code added to the blockchain in this way are called smart contracts. When a transaction targets a smart contract, validators must execute the corresponding source code using that transaction as input. The result of executing that smart contract will be reflected as an update in the ledger.

Many blockchains treat their state as a key/value store. In those cases, transactions and smart contract invocations will add and update values in the key/value store. Going forward, we will assume that all blockchains have a key/value store. The height of a blockchain is the total number of blocks that have been chained together thus far. In order to know the state of a blockchain at a particular height, you must process all of the transactions and smart contract invocations in order starting from the first block, called the genesis block.

2.2 Gas, Tokens and Wallets

Transactions are constantly being submitted to blockchains, and validators are constantly validating transactions and broadcasting messages to each other. Since compute resources and network bandwidth are limited, there is a limited amount of transactions that can be added per second. Therefore, it is important that users do not maliciously exhaust resources by spamming the blockchain with transactions. Blockchains enforce a cost to submitting transactions in order to prevent DoS attacks and spamming.

When the validators determine whether transactions should be included in the block, they will also compute costs for each transaction. This is called the gas cost. The gas cost is often computed based on factors such as the amount of computational resources required for a smart contract invocation, and the amount of blockchain state accessed. The sum of the gas costs for all transactions in a block is called the block gas, and usually there is a limit to how high this can be.

A token is a denomination of cryptocurrency that is tracked within the blockchain’s state. Every unit of gas represents a certain amount of tokens. Older blockchains like Bitcoin use an “unspent transaction output” model for tracking tokens [43]. However, we will assume an account-based model going forward, as it more prevalent in newer blockchains. Users will have a certain amount of tokens allocated to their accounts. All account information is stored within the blockchain’s state. When a user submits a transaction, they must pay a fee in tokens. While a blockchain may have many token denominations, the denomination used for paying gas fees is called the gas token. Users must sign transactions using the private keys associated with the accounts from which gas tokens are deducted. The device that manages these private keys is called a blockchain wallet.

2.3 Leader Election, Consensus Protocols and Finality

There are many possible valid blocks that can be constructed and used for any given height. Since blockchains are intended to be immutable however, validators must all agree on what the specific block should be at each height. Consequently, every blockchain has a protocol that the validators follow when selecting blocks. First, validators will determine a small set of leaders that are allowed to broadcast a block for a particular height. Following that, validators will come to a consensus on whether or not a block broadcast by a leader should be included in the ledger. According to a survey done in 2023, there were around 30 consensus protocols deployed in production blockchains, each with its own security and performance implications [34]. Once a block has been added to the ledger, there will be a finality period at which point the block is considered immutable.

2.3.1 Leader Election

The leader at a particular height is responsible for choosing the transactions that are included in the block for that height. This means that a leader has the liberty to selectively

omit transactions, or to only include transactions that benefit them. Transaction censorship is prevented by continually selecting leaders from a set of non-colluding validators. This is usually achieved by requiring validators to prove ownership of a scarce resource. Since it is difficult to monopolize ownership of a scarce resource, it is difficult to monopolize block creation, and therefore difficult to consistently censor transactions. We present two of the most common algorithms for leader election: Proof-of-Work and Proof-of-Stake.

For proof-of-work (PoW), validators must construct a block that hashes to a range of values. Since it is computationally infeasible to determine the pre-image of a hash, validators must randomly change a block's fields until it hashes to an acceptable value. A validator implicitly becomes a leader once it has constructed a valid block. The scarce resource is compute power in this case, and the most well known PoW blockchain is Bitcoin [43].

Proof-of-stake (PoS) is another leader election algorithm. For PoS chains, validators will perform a secure multi-party computation to determine when each validator can add to the blockchain. The amount of turns a validator gets is proportional to the tokens they have staked. Users may share their tokens with validators in order to increase the probability of that validator being chosen, as well as to share in the reward that validator gets from adding blocks. They will also share in the punishment if a validator's tokens are revoked for proposing invalid blocks. Users that do this are called delegators, and they, along with the validators, are stakeholders in the protocol. Ouroboros [38] was the first provably secure PoS protocol, and other PoS protocols have been derived from it. Some examples of PoS chains are Ethereum [32], Cardano [35] and Polkadot[47].

2.3.2 Consensus on Blockchain State

Once validators receive a block from the leader, they must agree on whether or not to include this block as part of the ledger. This is done using a consensus protocol. We present commonly used consensus protocols.

Longest Chain Rule

In general, proof class leader election algorithms such as PoW and PoS have the potential to generate multiple leaders, and therefore multiple legitimate blocks, for the same slot. For example, consider the scenario when two validators for a PoW chain generate two acceptable hashes at approximately the same time. Different validators will compute different, yet valid, blockchains. This is called a fork. When this happens, validators are

typically allowed to choose which ledger they would like to build upon and broadcast in the future. Ultimately, the longest chain is considered to be the true ledger. This is called the “longest chain rule”.

Voting

For voting class protocols, validators will collectively vote whether or not blocks should be added to the ledger. An example protocol is CometBFT [9]. CometBFT adapts the Practical Byzantine Fault Tolerance consensus algorithm [13] for the blockchain environment. The Cosmos Hub is a popular blockchain that uses CometBFT. Another voting class algorithm is HotStuff [56]. HotStuff uses leader election and threshold signatures to vote on blocks. As a consequence, HotStuff requires less messaging than CometBFT.

Due to the messaging complexity of voting protocols, blockchains using these consensus protocols typically only support a few hundred validators. In contrast, blockchains that follow the longest chain rule can support thousands of validators.

Paxos-based Consensus

The Paxos class of protocols include protocols such as Paxos [41] and Raft [45]. These protocols are not resilient to Byzantine nodes. Therefore, Paxos class protocols are not well suited for public chains. However, due to being crash tolerant, they may still be used for private chains. For instance, Raft based blockchain technology has been shown to be an effective solution in healthcare environments [2].

2.3.3 Finality

Finality is the guarantee of when a transaction in a blockchain is considered immutable. Voting class and Paxos class blockchains have instant finality. Since validators vote on each block, a transaction is accepted by all validators once it has been added to the blockchain. Some blockchains rely on the “longest chain rule” when determining which blockchain is valid. In those cases, a transaction is considered immutable once a sufficient number of blocks have been appended to the block containing that transaction. As long as the network is not partitioned, it is difficult for multiple equally sized blockchain forks to continually grow at the same rate. At some point, a leader will broadcast a single valid chain to every other validator. Therefore, the deeper a block is in the chain, the smaller the probability of it being replaced by the longest chain rule. Alternatively, blockchains might use a finality

protocol such as Casper [11]. In that case, validators will vote every N blocks to establish checkpoints before which transactions are immutable.

In order to incentivize validator participation, validators are generally paid for every finalized block they have added. A fee is usually taken from the gas fees paid by users that have submitted transactions in that block. There may also be a small amount of gas tokens minted and sent to the validator's account.

2.4 Blockchain Interoperability

Blockchain interoperability refers to the ability for blockchains to share data and exchange assets without modification the underlying system logic. Ren et al. performed a survey on blockchain interoperability protocols where they aggregate and taxonomize interoperability protocols [49]. They define blockchain interoperability as follows:

Definition 2.4.1 (Blockchain Interoperability). The ability to flexibly transfer assets, share data, and invoke smart contracts across a mix of public, private, and consortium blockchains without any changes to the underlying blockchain systems

A blockchain interoperability protocol is required for transferring tokens and other digital assets between chains. Interoperability typically involves first locking an asset on the source chain. In the case of cross-token transfers, this may involve sending those assets to a smart contract account, or burning those tokens so that they are out of circulation. In the case of general messaging, this may involve adding an entry to the underlying key/value store. Following that, a participant in the protocol, such as a program monitoring each blockchain, needs to tell the destination chain that the asset is locked. Finally, the destination chain will perform the required state transitions. This might involve generating synthetic versions of the original token in a process called minting. This could also involve invoking a smart contract to perform additional state updates.

Ren et al. identify that an interoperability protocol must always perform a verification process whereby someone verifies that there is a cross-chain transaction on the source chain [49]. This happens in one of three ways:

1. **Observation:** A trusted third party observes blockchain state by querying a full node. Full nodes connect to the blockchain network and maintain the entire state of the chain. Validators must run a full node.

2. **Simplified Payment Verification (SPV):** A process that can determine whether the underlying consensus protocol was correctly followed by reading the most recent headers of a blockchain.
3. **Certification:** A process that can determine whether the underlying consensus protocol was correctly followed by verifying validator signatures.

Verification through observation is generally simple to implement, as it usually just requires the receiving blockchain to check that the data it receives is signed by trusted third parties. Chainlink’s Cross-Chain Interoperability Protocol (CCIP) is an example where verification by observation is used [14]. A set of trusted third parties, called a decentralized oracle network (DON), will query a chain’s state from full nodes. Each party in the DON will attest to their observations by signing the data and broadcasting it to others in the DON. Eventually, the attested data will be committed to the destination chain. There is no single point of failure, since a DON is composed of many independent third parties. There is still centralization however, as Chainlink manages the economic and cryptoeconomic incentives that inhibit third party misbehaviour. Third parties that lie are financially punished, and their reputation is tainted such that they may no longer be allowed to participate.

A disadvantage of verification through observation is that it relies on trusted third parties. This introduces centralization which goes against core blockchain principles. Since observation does not verify the correct execution of a blockchain’s consensus protocol, the security of cross-chain operations is dependant on the security of the observers. SPV and certification verification methods verify the consensus protocol. As a result, cross-chain operations using these classes of verification are dependent on the security of blockchains. In order to deceive one blockchain into validating the incorrect state of the counterparty blockchain, one would need to compromise the counterparty’s validators such that they produce and potentially sign incorrect block headers.

2.5 IBC

Baton’s goal is to be a decentralized protocol for efficient and secure cross-chain message routing and delivery. In the previous section, we provided classifications for how interoperability protocols allow blockchains to verify each other’s states. We explained that verification through SPV or certification are better suited for decentralization than verification through observation. Therefore, the protocol that Baton is built upon must allow

for these types of verification. Furthermore, in order to deliver messages across a large network of blockchains, the protocol must be general enough that it can be implemented on many blockchains. These requirements led us to build on top of IBC.

2.5.1 Overview

The Inter-Blockchain Communication protocol (IBC) is an interoperability standard that is designed to be extensible and robust. Blockchains that connect to each other using IBC are able to verify each other’s state via SPV or certification. This is achieved using light clients. A light client is a program that exists on a blockchain. When a user invokes a light client with a transaction containing a blockchain’s state and an update message, that light client can verify whether the validators for that blockchain have reached a consensus on the new state. Different types of light clients will accomplish this in different ways. In our case, this involves checking whether or not a sufficient number of validators signed the new block header.

While there are numerous interoperability protocols, many of them are specific to pairs of chains [49]. IBC is inspired by TCP/IP, which is intended to allow for communication between all varieties of state machines. Just like with TCP/IP, individual chunks of data sent between blockchains are called packets. IBC packets have source and destination ports, and only one blockchain application can bind to a particular port at any given time. IBC packets have sequence numbers to enforce exactly once delivery. IBC also does not make any assumptions about the data being delivered, and leaves it up to the underlying application to process it correctly. As long as blockchain’s implement the IBC standard, they can communicate with each other.

Figure 1 provides an overview of IBC. Two chains will run IBC as a module; a module is a state-machine, such as a smart contract, that runs within the larger state-machine that is the blockchain. Each blockchain has a light client, denoted just as client in the diagram, that can interpret and validate the consensus state updates of the counterparty chain. Since blockchains might have light clients for multiple consensus protocols, IBC connections represent specific pairings of light clients between two blockchains. A channel is used to indicate a single flow of messages between applications running on each blockchain.

A relay is a program that periodically forwards client messages between the light clients on either chain. We explain relayers’ role in Baton in greater detail in section 4.3. Once the light client validates the counterparty chain’s states, the message will be forwarded to and processed by the application on the destination chain. Provided below are definitions for common terminology in IBC.

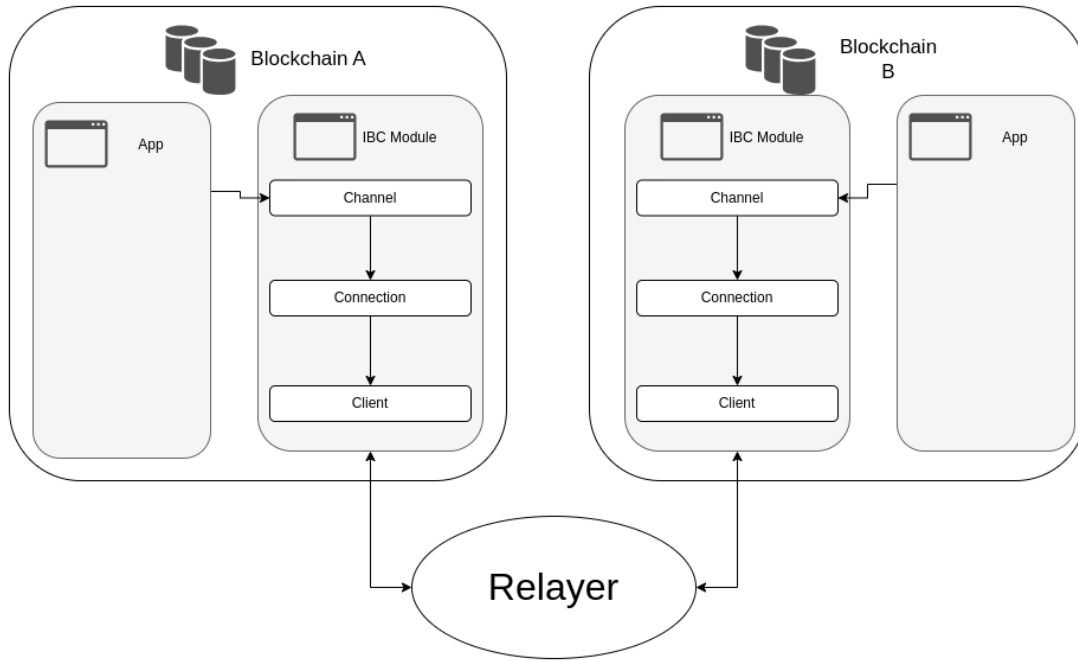


Figure 1: Two blockchains connected using IBC. The relayer forwards updates between the light clients within both chain’s IBC module. The connection tracks the state updates between pairs of light clients. The channel pairs a port and connection on one blockchain with a port and connection on the other blockchain. Channels also enforce exactly one delivery of all packets sent along it. Applications on either chain communicate by sending packets along a channel.

Definition 2.5.1 (Packet). A basic unit of data that is sent from an application on a source chain, and delivered to an application on a destination chain.

Definition 2.5.2 (Message). A single object whose end goal is to trigger a state transition. A packet can consist of one or more messages.

Definition 2.5.3 (Consensus State). The trusted view of a blockchain at a particular height. The consensus state can be used to prove the inclusion or exclusion of a key/value pair in the blockchain's state.

Definition 2.5.4 (Misbehaviour). A misbehaviour is the proof needed by the light client to determine whether a violation of the consensus protocol occurred [21].

Definition 2.5.5 (Client Message). A type of message that relayers can submit in order to update a light client to track a new consensus state for the counterparty chain. A client message may contain a misbehaviour which should freeze the client. A frozen client can no longer process messages [21].

Definition 2.5.6 (Light Client). Let C be a blockchain. Let s_C be the consensus state of C at block height h , and s'_C be the consensus state at block height h' where $h' > h$. Let $u_{s'_C}$ be a client message that contains s'_C . A light client is an algorithm that when given s_C and $u_{s'_C}$, can determine whether or not C correctly followed its consensus algorithm when updating its state from s_C to s'_C .

Definition 2.5.7 (Validity Window). Let C be a blockchain and let h and h' be two block heights on C such that $h' > h$. A validity window is any consecutive sequence of blocks $[h, h']$ where there exists a client message u that can be used to validate the consensus state at h' given the consensus state at h .

Definition 2.5.8 (Connection). Let C_a and C_b be two blockchains such that C_a and C_b are each other's counterparty. A connection is a pair of clients that periodically receive client messages for the counterparty chain. A connection remains open as long as the consensus states for all consecutive updates are validated by the client and fall within a validity window.

Definition 2.5.9 (Module). Any application, such as a smart contract, that runs on a blockchain.

Definition 2.5.10 (Port). Unique identifiers used to determine the intended destination module of an IBC message on the receiving chain. Only one module can bind to a port at a time, but a single module can bind to multiple ports.

Definition 2.5.11 (Channel). A logical mapping between ports and a sequence of connections, also called connection hops, between modules on two different chains. Channels enforce exactly once delivery of packets between modules, and may optionally enforce that packets are delivered in order.

Definition 2.5.12 (Consensus State). The trusted view of a blockchain at a particular height. Given a client message, consensus states must provide enough information to perform client updates and generate new consensus states [21].

Most IBC-enabled blockchains in production are currently built using the Cosmos SDK and run CometBFT. In this thesis, we will assume all participating chains are running CometBFT. The same abstract reasoning holds if CometBFT were to be swapped with another consensus protocol.

2.5.2 Client Updates

Every block for CometBFT chains contains the full list of validators. A subset of the validators will sign the header for any given block. Each validator’s public key is stored on-chain so their signatures can be verified. Each validator also has voting power that is proportional to the amount of tokens they have staked. A block is accepted when more than 2/3rds of the voting power signs the block.

Since CometBFT performs PBFT to achieve consensus, it is required that less than 1/3rd of the voting power is malicious [9]. Therefore, given a trusted header and a new header, a node can show that the consensus protocol was followed if:

1. More than 1/3rd of the voting power that signed the new header also signed the trusted header.
2. More than 2/3rd of the voting power signed the new header.

The first condition ensures that an honest validator from the trusted header also signed the new header. The second condition follows directly from the PBFT algorithm. The new header becomes the next trusted header.

The distribution of staking power constantly changes as validators acquire and spend tokens. The validator set also churns, as validators may choose to leave or join the protocol. In order to satisfy the first condition, client updates must be frequent enough such that a

subset of validators that signed the trusted header still have more than 1/3rd of the voting power when signing the new header. In section 5.5, we provide empirical evidence that Cosmos Hub, which is a popular IBC-enabled blockchain, experiences very little churn. We have observed this to be true for many other IBC-enabled blockchains as well. Therefore, client updates are usually submitted only when new packets are sent, since the update contains a consensus state that can be used to prove the existence of that packet. Very rarely are client updates needed to account for validator churn.

Additionally, every block also contains the Merkle root hash of the validators for the subsequent block. This is called the “expected validators hash”. If the new header’s height is one greater than the trusted header’s height, the light client will check that the validators from the new header hash to the expected validator hash from the trusted header.

2.5.3 Merkle Proofs

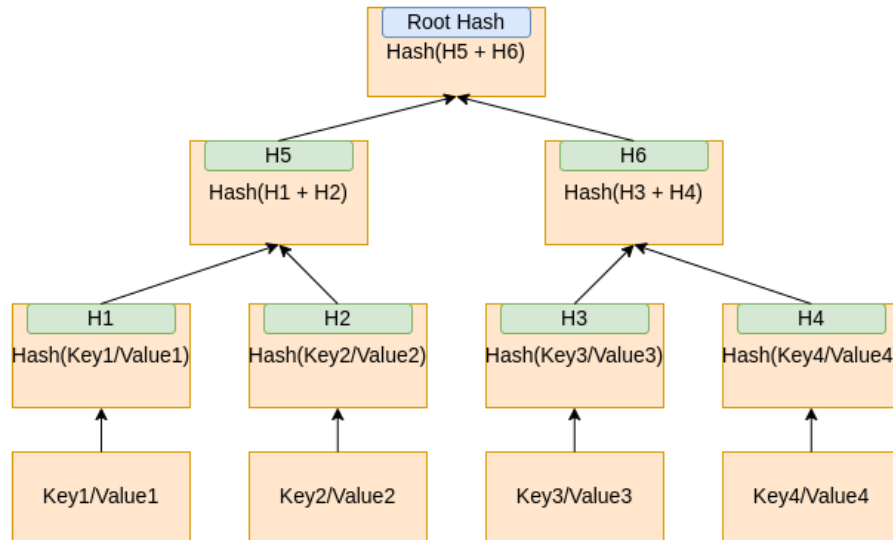


Figure 2: Example Merkle tree

Figure 2 provides an illustration of a Merkle tree consisting of four key/value pairs. A Merkle tree uses a single hash, called the root hash or the Merkle root, to represent all of the key/value pairs. In order to generate the root hash, the keys/value pairs must first be placed in order. This ordering must be deterministic such that only one correct ordering

exists. After that, each of the key/value pairs are hashed. Each of the resulting key/value hashes are placed in groups of two based on their order, and concatenated as pairs. The concatenated hashes are hashed again to form another sequence of hashes. The process of hashing the concatenation of adjacent hashes is repeated until a single, root hash is left.

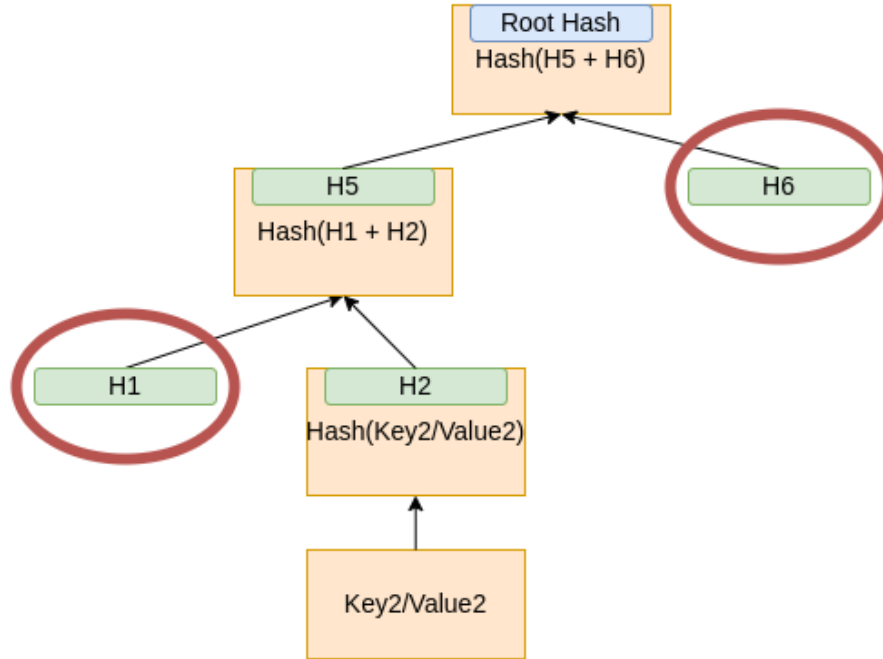


Figure 3: Merkle proof for “Key2/Value2”

Performing a Merkle proof involves reconstructing the expected root hash when given a key/value pair, and some other hashes. In the example given in Figure 3, “Key2/Value2” can be shown to be a part of the tree by reconstructing the root hash when given $H1$ and $H6$. Since it is computationally infeasible to determine a different key/value pair that can also be used to construct the root hash with $H1$ and $H2$, it is overwhelmingly likely that “Key2/Value2” is part of the tree. Our system uses the SHA256 hash function.

2.5.4 Proofs of Inclusion and Exclusion

IBC assumes that blockchains have fast, guaranteed finality. Since proof class chains can sometimes have their blocks reorganized, a finality layer must be applied in order for those

Channel End
State
Ordering
Counterparty Port
Counterparty Channel
Version
Upgrade Sequence

Figure 4: Channel end data structure.

chains to be IBC compatible. Finality prevents the situation where a transaction is proven to exist at a given time, only to have that transaction no longer exist later.

CometBFT blockchains organize their state as a Merkle tree, and the Merkle root for the current state is stored in every block. Proving that an entry exists on chain is equivalent to performing a Merkle proof. Since the keys are ordered, it is also possible to prove that a value does not exist on-chain. In that case, you prove the existence of the two values that appear on either side of where the non-existent value would be. By doing so, you can show that a value does not exist in the Merkle tree.

There are two steps for proving that there is a message on the source chain:

1. **Client Update:** Update the destination chain's light client with a new header. This header will include the Merkle root for the source chain's state.
2. **Prove Message:** Provide a Merkle proof for the existence of a message in the source blockchain's state.

2.5.5 Channels

Channel Overview

Once messages are proven to exist on a sending chain, those messages must be submitted to the correct module on the receiving chain. Channels provide a logical mapping that tells the receiving blockchain where to send incoming messages. Modules will bind themselves to ports, and channels will connect ports on the sending and receiving blockchains. Figure 4 shows the structure of a channel endpoint. A channel will have an endpoint on both chains.

The main channel states are init, try open, open or closed. The ordering denotes whether or not messages sent along a channel are ordered. The counterparty port and the counterparty channel represent the port id and the channel id respectively of the channel end on the counterparty chain. The version field shows the channel's version, and it is up to the underlying module to interpret what to do with the version. The upgrade sequence is incremented whenever a channel is updated.

Channel Open Handshake

A four-way handshake is used to establish channels. A blockchain user will submit a *MsgChannelOpenInit* message on the source chain. Since a Baton channel can span multiple blockchains, the user will specify a sequence of connections for the channel. The user will also specify the channel version, port, ordering and counterparty port. The source blockchain will create a corresponding channel end, assign it a channel id, and give it the init state.

Following that, the relayer will submit a *MsgChannelOpenTry* on the destination chain. the destination chain will validate that the expected channel end exists on the source chain. Following that, it creates its own channel end with the appropriate version, port, ordering and counterparty port. The channel ends on either chain must have the same port, ordering and version. The destination chain will also assign that channel an id and give it the try open state.

Next, a *MsgChannelOpenAck* message is submitted on the source chain. The source chain will verify that the destination chain created a channel end to match its channel end. If the validation passes, the channel's state will be changed to open.

Finally, a *MsgChannelOpenConfirm* is submitted on the destination chain. The destination chain verifies that the channel end on the source chain is open. It will then open

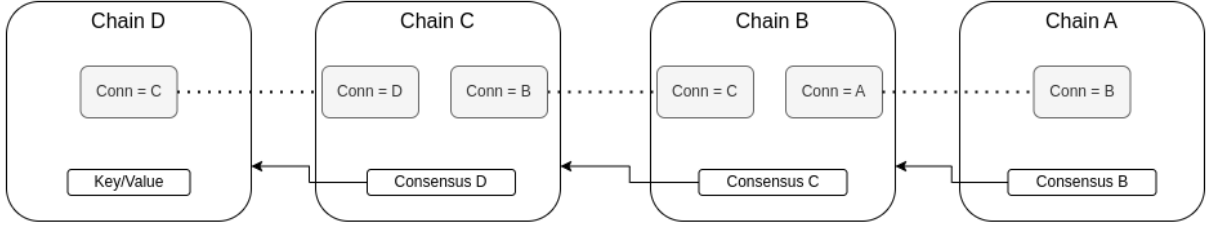


Figure 5: IBC-enabled blockchains store the consensus states of its neighbour and the corresponding connection end information in its key/value store. This means that the consensus state of a blockchain can be used to prove it is connected to a neighbouring chain, and that it knows that neighbour’s consensus state. Ultimately, the source chain’s consensus state is used to prove the existence of the message being sent.

its side of the channel. The channel handshake is now complete, and the channel is ready for use.

Closing Channels

IBC allows a channel to be closed if messages should no longer be sent along that channel. When this happens, both channel ends have their state changed to closed. Many applications, such as cross-chain token transfers, do not allow channels to be closed. Consequently, it is rare for channels to be closed in practice.

2.5.6 Multi-Hop Proofs

An IBC channel can be composed of multiple connection hops such that it spans multiple chains. Multi-hop proofs must be used in those instances to validate the source chain’s state on the destination chain.

Multi-hop enforces transitive security by recursively performing Merkle proofs. Consider Figure 5 as an example. The proof construction starts off the same way as in the single-hop case. A Merkle proof will be generated for some key/value pair on *ChainD*, and *ChainC* will learn *ChainD*’s corresponding consensus state after a client update. We call this the key proof.

ChainC now needs to prove to *ChainB* that it is connected to *ChainD*. Therefore, a Merkle proof must be generated for the the connection between *ChainD* and *ChainC*.

There will also be a proof for *ChainC*'s commitment to *ChainD*'s consensus state. These two proofs will be generated using the same Merkle root from *ChainC* at a height h . Once those proofs are generated, a relay must update *ChainB*'s light client to *ChainC*'s header at height h .

This process continues recursively until a relay updates *ChainA*'s light client for *ChainB*. At this point, the entire multi-hop proof will be submitted to *ChainA*. *ChainA* will first prove its connection to *ChainB*, then *ChainB*'s connection to *ChainC* all the way to *ChainD*. Finally, *ChainA* will validate the key-proof.

2.5.7 Relay Incentives

Since it costs relayers gas tokens to perform client updates and submit proofs, there needs to be incentive for them to do this. Otherwise, they would just be spending tokens with no reward. IBC leaves it up to the blockchain modules to determine how to pay relayers for cross-chain transactions (ics-029) [21]. Consequently, general relay incentivization is out of scope for Baton. Therefore, we present IBC's recommended compensation approach for relayers.

IBC assumes that relayers will have accounts on both the source and destination chains. The destination chain should have a mapping from destination account to source account for the relay. When a relay submits a message on the destination chain, the acknowledgment will contain that relay's account on the source chain. Gas tokens will be transferred both to the relay that submitted the acknowledgment on the source chain, and to the account given in the acknowledgement. That way, the relayers involved in both directions of communication will be compensated on the source chain.

2.6 Traditional Networking Solutions and Blockchains

Blockchain interoperability protocols and traditional networking protocols address a similar problem. Both blockchains and computers are state machines that operate multiple internal applications. Without external intervention, applications on blockchains and computers are isolated from applications on different blockchains and computers. In either case, users want applications to be able communicate with each other. Therefore, the goal of blockchain interoperability protocols and computer networking protocols are to route and deliver messages such that applications on different state machines can communicate. Despite this, there are critical differences that inhibit the use of traditional routing and

packet delivery solutions for blockchain interoperability. We describe those challenges in this section, as they motivate our design decisions for Baton.

2.6.1 Packet Routing

It is a challenge to establish a direct connection between all pairs of blockchains, because blockchains differ in design, implementation and operation [49]. It is time consuming and monetarily costly to update each blockchain to be able to interoperate with every other blockchain’s infrastructure. Even when blockchain infrastructures are compatible, a trusted third party is needed when establishing new connections. This introduces an attack vector that can be exploited every time a connection is created. Therefore, network-wide interoperability requires messages to be routed across multiple existing IBC connections. Routing protocols such as OSPF and EIGRP require routers to disseminate connectivity information so that each router can learn the topology of the network. Such routing protocols are expensive when directly applied to blockchain interoperability due to their high communication complexity.

One consideration we made when designing Baton was whether or not blockchains should compute routes on-chain. Since all blockchain transactions have monetary costs, communication between blockchains is more expensive than general communication between computers. Communication between autonomous systems (ASs) is facilitated by the Border Gateway Protocol (BGP). With BGP, messages are disseminated between neighbouring ASs to determine policy adherent routes. Within an AS, link state or distance vector protocols such as OSPF, EIGRP or RIP are commonly used. By considering the current state of IBC enabled blockchains, we can estimate how costly these routing protocols would be. Currently, there are approximately 1000 IBC connections across 50 blockchains [22]. This means that approximately $50 * 1000 = 50000$ messages would be sent to disseminate connectivity information about every blockchain in the network. OSPF refloods connectivity information every 30 minutes by default to account for changes. EIGRP and RIP also periodically resend messages. That said, a similar protocol for blockchain interoperability may also need connectivity information to be periodically resent. Considering that the mean amount of cross-chain transactions per day across all blockchains is roughly 56000 [22], a routing protocol for blockchain interoperability could introduce a relatively high amount of transactions and require blockchain users to pay much higher fees. In addition to the cost of messaging, it also costs gas to perform computations on-chain. Therefore, it is better to avoid having blockchains compute the shortest paths, as OSPF would require for example.

Rather than using a link-state or distance vector routing protocol, ASs can instead determine routes using software-defined networking (SDN). However since SDN uses controllers to determine routes between nodes in the network, each node ultimately needs to trust that the controllers are providing them with accurate information. This introduces another challenge, as blockchains typically do not trust information that cannot be verified on-chain. A blockchain’s validators will verify the state changes caused by incoming transactions. Given an ordered sequence of transactions and a blockchain state, there will be only one possible valid new blockchain state. Performing state transitions based on external information is a challenge, since validators will come to different conclusions depending on where and when they retrieve that information. Furthermore, the correctness of state transitions is no longer dependent on the security of the blockchain, but on the security of the external sources. SDN is commonly used in networks where the controller is assumed to be trustworthy. While SDN would be a cost effective solution for networking across blockchains, as it does not require a lot of messaging, it introduces a central point of failure that may be less secure than the blockchains themselves. This could lead to routing attacks where user’s messages are directed to unintended destinations.

2.6.2 Data Integrity

In traditional computer networking, endpoints use a protocol like TLS to exchange the public keys that will be used when verifying signatures. By doing so, the recipient can verify whether or not the source sent the given messages. Verifying that the source chain sent certain messages is more nuanced with blockchains. With IBC, this requires verifying that the validators for the source chain have correctly executed the consensus protocol. Unfortunately, there are many different consensus protocols that every blockchain would need to know how to verify [24]. Blockchains would need to constantly be upgraded to support different consensus protocols and cryptographic schemes, but it can take months to update blockchain software [20]. In contrast, it is typically easier to update client software to support the signature schemes used by servers. Hence, protocols like TLS expect that there is a signature and encryption scheme that both the client and server support that will allow them to communicate. We cannot expect this for blockchains.

Additionally, a blockchain’s validator set is constantly changing. This means that validator set membership information would need to be reliably updated in order for blockchains to continually verify data integrity. Avalanche’s solution to this problem, for example, is to have a central blockchain that maintains validator membership information for every chain, and that is validated by validators from all other blockchains [50]. This central blockchain, called the P-Chain, has a similar function as a certificate authority.

The limitation of this approach however is that it introduces a single attack vector that if exploited, will compromise all cross-chain communication.

2.6.3 Name Resolution

Another challenge with blockchain interoperability is the ability to uniquely identify a particular blockchain, and to distinguish it from other blockchains. Not being able to do so can have negative security and financial implications for users that are sending cross-chain messages. Route selection in traditional networking has little impact on security, since routers typically cannot alter packets that are in transit to a destination. Furthermore, traditional routing protocols do not take into account the financial cost of selecting one route over another.

In contrast, if any one of the blockchains along a multi-hop IBC channel is compromised, that chain could perform arbitrary state transitions. The corresponding state proofs could be used to fabricate harmful cross-chain messages that would be processed by the destination chain. This can be a serious problem because, since IBC is permissionless, anyone can spawn a blockchain and establish a multi-hop IBC channel with their chain as one of the intermediate chains. As messages are being sent on a channel, there is also a monetary cost that the relayer must pay when submitting the required state proofs to the intermediate blockchains. Since users may need to compensate the relayer as described in Section 2.5.7, the amount the user must pay to have their message delivered can depend on the route that message takes. Therefore, it is beneficial to be able to identify blockchains along a route to avoid using channels that are insecure, or that are financially costly.

Blockchains do not inherently have names that uniquely distinguish them from other blockchains. Currently, users must consult an external registry to learn which RPC endpoints are for which blockchains [17]. These registries play a similar role as DNS servers, as they resolve blockchain names to RPC endpoints like how DNS servers resolve domain names to IP addresses. That said, users can verify the identities of the blockchains along an IBC channel by referencing a blockchain registry. Unfortunately, blockchain registries have limitations in that they are central points of failure. If exploited, messages can be routed to the incorrect destination chain, or users may send a message along an IBC channel that contains a compromised blockchain.

Baton does not rely on external sources to identify blockchains along an IBC channel. Rather, Baton uses cryptoeconomic incentives to ensure that each blockchain's validators correctly identify other blockchains. This ensures that blockchain name resolution is correct as long as the validators do not collude to provide false information. Since we grant that the

source and destination chain must be honest, it is reasonable to assume that the validators on either chain will not collude. Blockchain name resolutions are stored on-chain for the user's convenience.

Chapter 3

Related Work

Baton is intended to allow sovereign blockchains to interoperate, where a sovereign blockchain is any independent blockchain built to serve a designated purpose. In this section, we present systems that provide interoperability across a network of sovereign blockchains. These protocols allow for general messaging, and can be divided into the following categories: Relays, Side Chains, and Notary Schemes.

3.1 Relays

Relays are protocols where a third party is responsible for reading the state of the source blockchain, and submitting that state to the destination blockchain. The endpoint blockchains are responsible for validating each other's state. This is commonly accomplished by running a light client on either chain, as a light client can validate the block headers and validator signatures of the counterparty blockchain.

Cosmos Hub is the first blockchain built using the Cosmos SDK, and is a hub for IBC-enabled blockchains [18]. Sovereign blockchains, called zones, can permissionlessly connect to Cosmos Hub. In order for zones to communicate with each other, they must do so in two phases. The source zone will send an IBC message to Cosmos Hub, then Cosmos Hub will send an IBC message to the destination zone. Consequently, Cosmos Hub keeps track of the cross-chain assets for each of the connected zones. Cosmos Hub is a relay protocol, as IBC relayers forward state information between endpoint blockchains that is ultimately verified on-chain. Cosmos Hub has limitations in that it is both a performance and security bottleneck. All cross-chain messages between zones must be committed on Cosmos Hub, and if Cosmos Hub's validators are compromised, all cross-chain messages can be altered.

The Octopus Network is a multi-chain network that is designed to bootstrap and run application-specific blockchains [44]. The main component of the Octopus Network is the Octopus Relay; the Octopus Relay is a set of smart contracts on the NEAR blockchain [48]. These smart contracts implement IBC, thus allowing IBC-enabled blockchains to communicate with each other using NEAR. IBC-enabled blockchains may also communicate with Ethereum based blockchains using the Octopus Relay, as The Rainbow Bridge protocol running on NEAR allows it to interoperate with Ethereum [8]. Since cross-chain communication involves transacting on the NEAR blockchain, this approach introduces a performance and security bottleneck. Furthermore, it can take several hours for Rainbow Bridge operations to complete, where it takes seconds to a few minutes for IBC operations.

Polymer enhances Ethereum based chains with IBC to provide modular security and permissionless cross-chain communication [40]. Many Ethereum based blockchains are layer-2 chains. Layer-2 chains are independent blockchains that periodically settle their state to a layer-1 chain, such as Ethereum. This allows users to transact on a faster and less expensive chain, rather than transacting on Ethereum directly. As layer-2 chains broadcast their state to Ethereum, Polymer Hub, which is also a layer-2 chain, will also collect these states. Polymer Hub aggregates and commits this data to its own state. Since Polymer is IBC-compatible, other IBC-enabled blockchains can interoperate with Polymer Hub. This allows IBC-enabled chains to also interoperate with Ethereum-based chains as a consequence. Polymer Hub is limited in that it relies on the security of the sources from which it receives each layer-2's state. The performance of Polymer Hub is also limited by the layer-2 chains' settlement speed, which can sometimes be in excess of 7 days [46].

Hyperlane is a permissionless interoperability framework that is deployed as smart contracts to blockchains [36]. Two of the core types of smart contracts are mailboxes and inter-chain security modules (ISMs). When someone would like to perform a cross-chain operation, they must submit the message along with the destination chain ID to a mailbox. A relayer will read the message from the mailbox, and retrieve the associated proof by invoking the appropriate ISMs. The relayer will submit the message and the proof to the mailbox on the destination chain. The destination chain will process the message after having verified the proof. Since Hyperlane smart contracts can be deployed permissionlessly, users will need to refer back to Hyperlane's official website to determine the correct smart contracts and chain IDs to use. In contrast, Baton uses cryptoeconomic incentives to ensure that the correct routing information is stored on-chain.

3.2 Side Chains

Side chain protocols revolve around central blockchains that participate in governing activity on other blockchains, called side chains. As data is transmitted from chain to chain, validators will validate state transitions for the central chain and their respective side chains. The central chain therefore behaves as a liaison for the side chains.

Polkadot is an example of a side chain protocol [54]. Polkadot’s hub chain is called the “relay chain”. Developers can create their own blockchains, and connect them to the relay chain. These blockchains are called “parallelized” chains, or parachains for short. Parachains are secured by the relay chain, as they share validators. Parachains can communicate with each other using Polkadot’s custom messaging protocol called Cross-Consensus Message Passing (XCMP). Validators that participate on both the relay chain and the parachain are called collators. With XCMP, collators for the source parachain will update the relay chain so that the collators for the destination parachain can validate and process cross-chain messages. A limitation with Polkadot is that there is a limited amount of parachains that can connect to the relay chain. So, parachains must pay for a slot, and the lease must periodically be renewed. Furthermore, parachains share state with the relay chain. If the relay chain has to revert its state, parachains must also revert their state [54].

Avalanche is a platform composed of subnets, where a subnet is a set of validators that validate a sovereign blockchain. A validator can be a part of multiple subnets, and every validator needs to be part of the P-Chain. The P-Chain is a central blockchain that tracks validator set information for each of the subnets. Blockchains communicate with each other using Avalanche’s custom messaging protocol called warp messaging. With warp messaging, validators will use BLS signatures to sign messages. This signed message is then submitted as a transaction on the destination chain. Since every validator is a part of the P-Chain, the validators of destination chain know the public keys for the validators of the source chain. This allows the destination chain to validate the integrity of a message by verifying the BLS signature. A limitation of Avalanche is that the P-Chain is a security bottleneck. Compromising the P-chain can allow a malicious actor to falsify public keys and forge messages.

3.3 Notary Schemes

For notary scheme protocols, trusted third parties will validate the state of both the source and destination blockchains. This is commonly done by having a threshold of the notaries

sign the blockchain’s state. The source and destination blockchains are aware of who the notaries are. Therefore, an endpoint chain will consider a counterparty chain’s state to be valid as long as it can verify that the notaries attest to it being valid.

Chainlink’s Cross-Chain Interoperability Protocol (CCIP) is an interoperability protocol for general messaging [14]. Cross-chain messages are sent with the help of groups of trusted third parties called decentralized oracle networks (DONs). The committing DON will scan the source chain for cross-chain message transactions, and store the commitment for those messages on the destination chain. The Risk Management Network, also a group of trusted third parties, will vote on whether or not the cross-chain message exists on the source chain. If the vote concludes that the message does exist on the source chain, that message is then considered blessed. Otherwise, it is cursed. When the executing DON sees that a transaction has been blessed, it will submit a Merkle proof for that transaction on the destination chain. The destination chain will verify the proof and process the message accordingly. A limitation with CCIP is that Chainlink ultimately decides who can be an Oracle. This is a centralized approach where Chainlink is responsible for enforcing economic and cryptoeconomic incentives that keep oracles from lying. Since Chainlink is not permissionless, newer blockchains must contact Chainlink to be supported.

Axelar provides another blockchain interoperability solution as a “decentralized cross-chain read/write oracle” [5]. Axelar’s validators are expected to run light clients for other blockchains in the network. Collectively, Axelar’s validators will use threshold signatures to submit transactions on other blockchains. When someone would like to perform a cross-chain transaction, they submit that transaction to an account or smart contract on the source chain that is controlled by Axelar’s validators. Axelar’s validators will perform the necessary queries and operations on the destination. Finally, the results will be signed and submitted to the source chain. Since cross-chain operations must be signed by a high threshold of validators (upwards of 80%), cross-chain operations are secure as long as Axelar’s validators do not collude. Axelar is limited in that there is currently a maximum of 138 validators that can sign and facilitate cross-chain operations [22]. Millions of dollars have been stolen in the past due to poor key management [7, 52], so Axelar may be susceptible to similar types of attacks.

LayerZero is a permissionless interoperability protocol that provides censor-resistant general cross-chain messaging [39]. The three core infrastructures in LayerZero are the LayerZero endpoint, the executor and the decentralized verifier networks (DVNs). DVNs are independent third party oracles that observe LayerZero enabled chains’ state. For example, a DVN can be a Chainlink decentralized oracle network. Users must specify a set of DVNs that they would like to verify their cross-chain messages. When the user submits the cross-chain message on the source chain, the executor will read the message

from an endpoint, package it, and submit it as a packet on the destination chain. While this is happening, the selected DVNs will generate a proof for the packet and submit that proof on the destination chain. The destination endpoint will verify the packet proof, and forward the verified packet to the intended application. A limitation of LayerZero is that its security depends on the DVN honestly reporting chain data.

Chapter 4

System Design

4.1 Overview

Baton is a blockchain interoperability protocol built on top of IBC [21] that efficiently and securely routes and delivers messages across a network of blockchains. Since IBC enabled blockchains communicate via direct connections with each other, prevailing approaches to wide spread interoperability, such as Cosmos Hub [18] and Axelar [5], have a hub-and-spoke architecture. Cosmos Hub and Axelar are hub blockchains where other blockchains must directly connect to them, and communicate through them. Unlike with Cosmos Hub and Axelar, Baton does not require chains to be connected to a hub to communicate. Rather, all pairs of chains running Baton can communicate as long as they are indirectly connected. Baton provides wide spread interoperability in a decentralized way as a consequence, thus avoiding the pitfalls of central points of failure, high transaction fees due to transactions competing for limited space, and throughput bottlenecks that centralized hub-and-spoke approaches introduce.

Figure 6 provides a diagram of Baton’s components. A Baton User is any individual that would like to transfer data from a source blockchain, called Blockchain A in the diagram, to a destination blockchain, denoted as Blockchain B. Users interact with blockchains by using software known as wallets, which query validator RPC endpoints. The diagram also depicts the Baton participant, where participants are individuals that transact on a blockchain to collectively operate the protocol. In order to use Baton, blockchains must run three programs called the Baton module, the Splitter module, and the IBC module. Participants in the protocol interact with the Baton module to establish an on-chain record of the paths through blockchains that each IBC channel takes. The Splitter module acts as

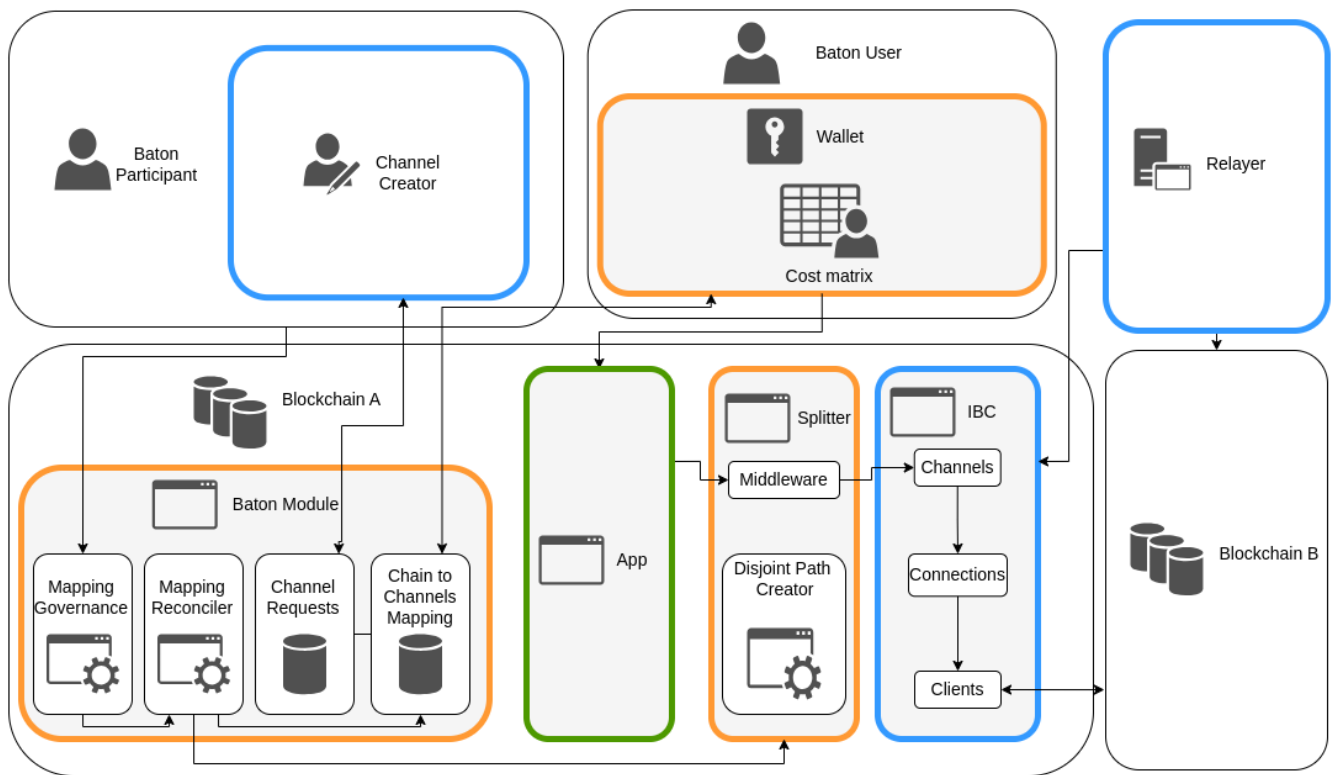


Figure 6: An overview of Baton's infrastructure

middleware between existing applications and the underlying IBC module, and can send messages along multiple disjoint channels to the same destination chain. The IBC module implements the IBC protocol, which allows blockchains to interoperate [21]. The relay is an external program that monitors the IBC module of a blockchain for outgoing messages, and submits the appropriate transactions to the IBC module on the counterparty chain. Components outlined in orange are newly added by Baton. Components outlined in blue exist as a part of standard IBC, but have been modified. The application component in green is unmodified by Baton.

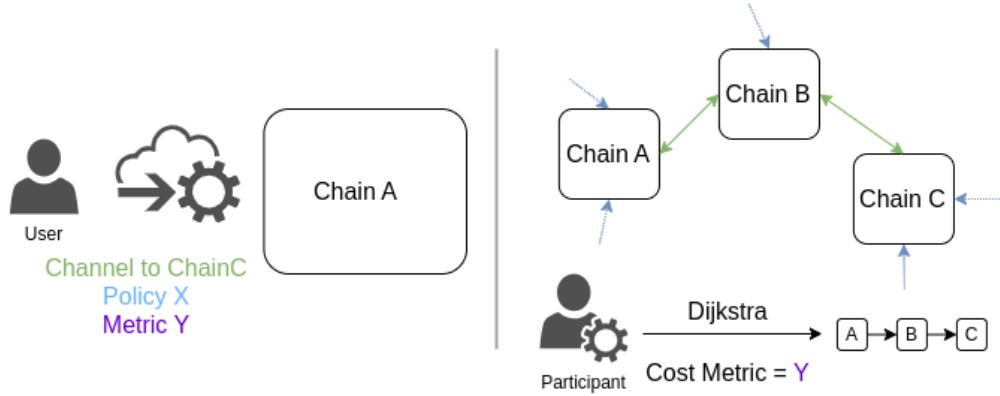


Figure 7: User will request a channel that connects the source and destination chain, that meets certain security requirements and that is optimized according to a specified metric. A protocol participant will compute an appropriate route offline.

The protocol starts with a user that would like to send a message from an application on one blockchain to an application on another blockchain. In order for this to happen, there must be an IBC channel that spans a sequence of connections from the source blockchain to the destination blockchain. Initially, there will not be any IBC channels connecting the apps on both chains. Figure 7 shows that the user will submit a request to the Baton module on the source chain to have a channel created. The user will also submit security requirements that the blockchains along the channel must adhere to, as well as a metric that can be used to measure the cost of a route. Participants in the Baton protocol will see this channel request, and search for a route that satisfies the user’s requirements. The participant does this by querying blockchain RPC endpoints to construct a local view of the network connectivity. The participant can compute an optimal route using an algorithm like Dijkstra’s shortest path algorithm [25], and will use the user specified metric as a way to measure the route’s cost. Example metrics are the hop count and the estimated gas

cost for light client updates.

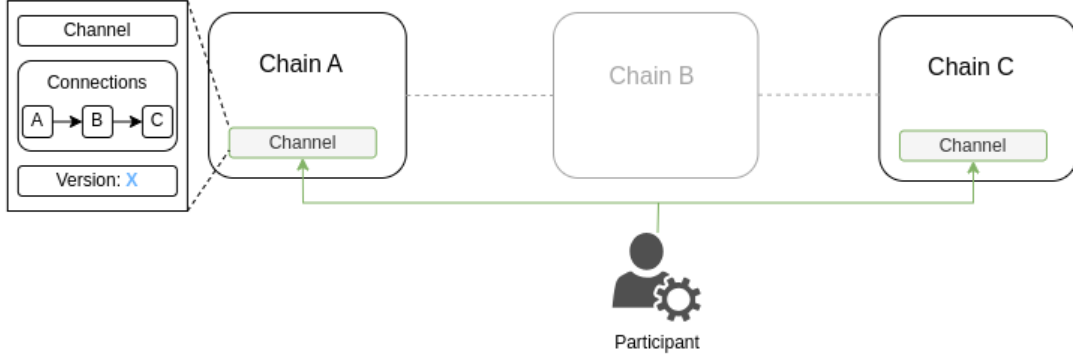


Figure 8: The protocol participant will create an IBC channel that corresponds to the computed route.

Figure 8 illustrates participant creating a multi-hop IBC channel that spans the computed route, as described in Section 2.5.5 and Section 2.5.6. Since IBC is a permissionless protocol, anyone can create a multi-hop channel that spans blockchains they also created. This is a problem because a blockchain that is operated by a single entity can perform arbitrary state transitions to fabricate cross-chain messages. Consequently, The participant can request that the validators on the source chain identify and permit the use of the blockchains along a route by triggering a governance vote. A channel is considered validated once all of its blockchains have been identified.

A user can query the source blockchain’s RPC endpoint for all validated channels that are destined to a specified blockchain. The user will filter out every channel that does not adhere to the desired security policies, and will select the least costly channel according to their cost metric. Figure 9 depicts the user sending their message along a chosen channel. After seeing the user’s message, the relayer will construct a multi-hop Merkle proof that will allow the destination chain to prove the existence of the message on the source chain. The relayer will also include proofs that the intermediate blockchains adhere to the channel’s security policy requirements. This proof is ultimately submitted to and verified by the destination blockchain.

When there are multiple disjoint channels that connect the same endpoint blockchains, messages can optionally be sent on each channel. Then, the destination chain can compare the messages sent along each channel to check whether or not they match. This functionality is handled by the Splitter module. When used as middleware, the Splitter module will send cross-chain messages along multiple disjoint paths to the same destination chain.

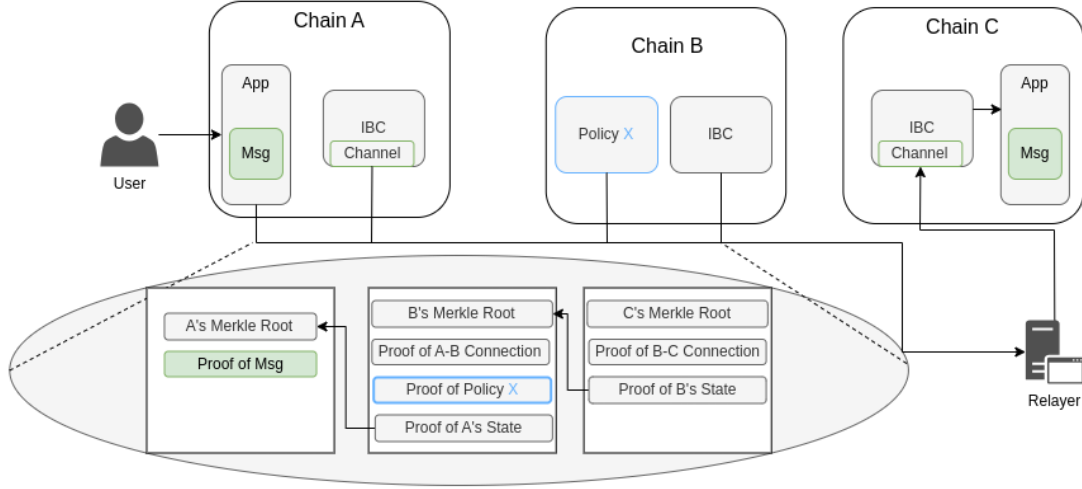


Figure 9: The user can now send their message on the newly created channel. Seeing that blockchains along the channel must adhere to security policy X , the relay will append proofs of property X to the original multi-hop proof.

While this incurs additional gas costs, it increases the security of cross-chain messaging. If an intermediate chain on one of the paths maliciously alters a message, this misbehaviour can be detected.

4.2 Baton User

A Baton user is anyone who would like to submit cross-chain transactions. Gas tokens are taken from a user's on-chain account to pay for cross-chain transactions. Each account has an associated public key, and users must sign transactions to prove ownership of the account. It is common practice to use a wallet to manage keys and sign transactions. The wallet is what allows the user to transact on blockchains in accordance with the Baton protocol. We can assume that the wallet is honest, as it runs on the user's device and is under the complete control of the user. We use the Cosmos SDK command line interface to sign and submit transactions, and incorporate BASH scripts to automate the CLI's interactions with the blockchains.

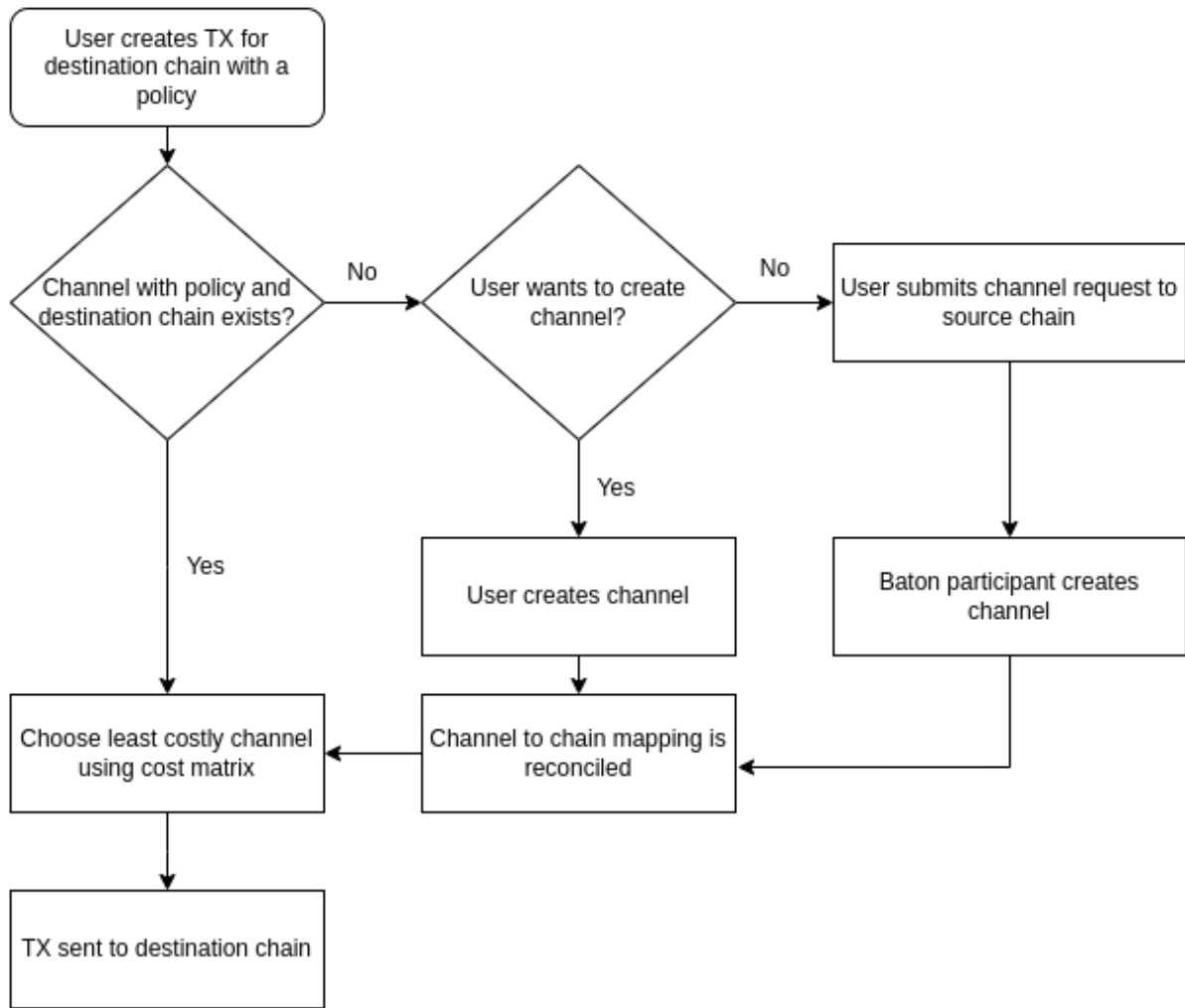


Figure 10: Baton user's flow of operations

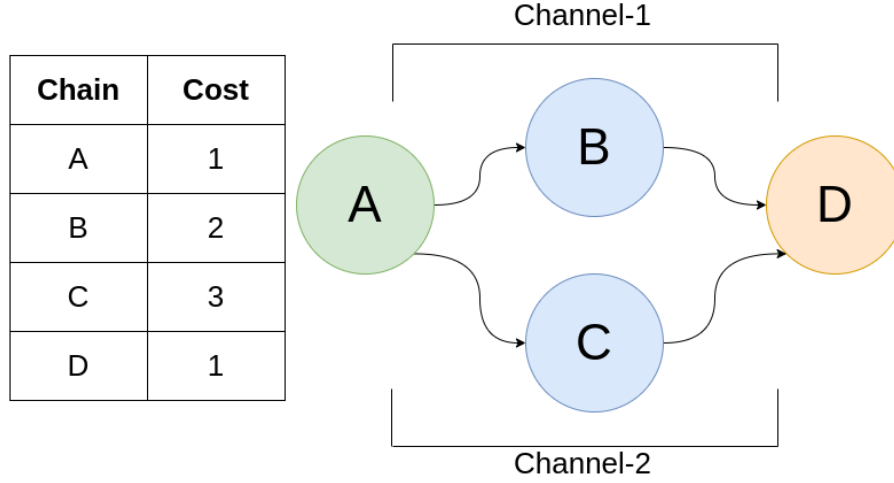


Figure 11: The cost matrix on the left is used to compare the cost of Channel-1 and Channel-2. Channel-1 has cost 4 and Channel-2 has cost 5. Even though both channels connect the same endpoints, Channel-1 will be selected.

Figure 10 shows the complete flow of operations from a user’s perspective. Users will start by specifying a transaction, a destination chain ID and a security policy that every blockchain processing the transaction should adhere to. The wallet will query the source chain through its public API server for all channels that lead to the destination chain. Following that, the wallet will filter the channels to only keep those that adhere to the user’s requested security policy. Finally, the wallet must determine which of the remaining channels to use. Baton allows users to determine for themselves how to compare channels. This is done by constructing a matrix that associates blockchains with user specified costs. This cost can represent whatever the user would like, such as the relative monetary cost of transacting on one chain over another. When multiple acceptable channels exist, the wallet will select the channel with the lowest cost. Furthermore, the wallet can maintain multiple cost matrices for different use cases. Figure 11 demonstrates how costs are computed.

If there are no acceptable channels for the user, a new channel must be created. Fortunately, failing to find a channel will not cost the user any gas since queries do not change the state of the blockchain. The user may choose to create a channel and validate it as part of the Baton protocol, or submit a request to have someone else create and validate the channel. Section 4.4 explores this process in detail.

4.3 Relayer

A Baton relayer is a third party process that is incentivized to forward messages back and forth between blockchains. Blockchains do not have access to the network layer in and of themselves. They can only process data that has been submitted to them as transactions. Relayers periodically read the transactions targeting the IBC module on one blockchain, and submit the corresponding transactions to another blockchain. By doing so, we can have blockchain communicate.

Algorithm 1 Standard Relayer Algorithm

Require: $C : \text{Set}<\text{Chain}>$

```
for chain of  $C$  do
  for counterparty of  $C$  do
    if chain  $\neq$  counterparty then
      for channel of getChannels(chain, counterparty) do
        datagrams = chain.pendingDatagrams(counterparty, channel)
        for localDatagram of datagrams[0] do
          lProof, lUpdate = counterparty.genProof(localDatagram, channel)
          chain.submitDatagram(localDatagram, lProof, lUpdate)
        end for
        for cptyDatagram of datagrams[1] do
          cProof, cUpdate = chain.genProof(cptyDatagram, channel)
          counterparty.submitDatagram(cptyDatagram, cProof, cUpdate)
        end for
      end for
    end if
  end for
end for
```

Algorithm 1 presents the standard relayer algorithm [29]. The relayer can decide how often to run this algorithm, although it usually runs every time a new block is added to one of the chains being monitored. The relayer first iterates over every pair of chains for which there is a channel, and reads the pending IBC messages on both chains. The relayer will then query the Merkle proofs for each of the pending messages, as well as the client messages required to update both light clients with the counterparty's consensus state. The relayer will finally submit the IBC messages, proofs, and client update messages to the respective chains. Once the receiving IBC module proves that the message exists on

the sending chain, that message will be forwarded to the application that is bound to that channel's port.

IBC requires that no exactly-once or deliver-or-timeout safety properties of IBC depend on the relayers [21]. Consequently, relayers should not be able to alter messages that have been committed on either chain. As described in section 2.5.2, a light client will accept a consensus state update as long as that state has been signed by more than 2/3rds of the voting power. Section 2.5.4 explains that the consensus state contains a Merkle root hash that can be used to prove the inclusion of IBC messages on the counterparty chain. When a channel spans multiple connections, multi-hop proofs are required by the destination chain in order to prove the inclusions of messages. Section 2.5.6 describes how multi-hop proofs are constructed and verified. Section 2.5.7 explains how relayers can be compensated for the gas paid when submitting transactions.

Algorithm 2 Baton Relay Algorithm

Require: $C : \text{Set}\langle \text{Chain} \rangle$

```

for chain of  $C$  do
  for counterparty of  $C$  do
    if chain  $\neq$  counterparty then
      for channel of getChannels(chain, counterparty) do
        datagrams = chain.pendingDatagrams(counterparty, channel)
        policy = channel.getPolicy()
        for localDatagram of datagrams[0] do
          lProof, lUpdate = genProof(localDatagram, channel)
          policyProof = genPolicyProof(localDatagram, channel, policy)
          lProof.append(policyProof)
          chain.submitDatagram(localDatagram, lProof)
        end for
      for cptyDatagram of datagrams[1] do
        cProof, cUpdate = genProof(cptyDatagram, channel)
        policyProof = genPolicyProof(cptyDatagram, channel, policy)
        cProof.append(policyProof)
        counterparty.submitDatagram(cptyDatagram, cProof)
      end for
    end for
  end if
end for

```

Algorithm 2 presents an additional step that Baton adds to the relayer algorithm, where the relayer will send policy proofs along with the message. As will be explained in further detail in Section 4.4.2, channels can be assigned security policies upon creation. Security policies tell the relayer that each chain along the channel must have certain properties. These are properties that indicate how difficult it would be to compromise a blockchain. An example property is the number of validators. It generally becomes increasingly difficult to compromise a blockchain the more validators it has, because that would require exploiting more independent parties. When a relayer sees that a channel has been given a certain security policy, it knows to append additional Merkle proofs of inclusion for the state information that demonstrates that each blockchain has that property.

4.4 Baton Participant

A Baton participant is any individual capable of submitting transactions to the blockchain and who participates in the Baton protocol. Baton is permissionless, meaning that anyone can decide to become a Baton participant simply by interacting with the protocol. They are responsible for collectively creating and validating the multi-hop channels that messages are sent on. In doing so, they facilitate decentralized, network-wide cross-chain message routing and delivery. In this section, we present how participants learn which channels need to be created. We describe how they apply security policies to modules and channels, and explain the steps they must take to validate those channels as part of Baton.

4.4.1 Processing Channel Requests

When a user attempts to submit a cross-chain transaction, it is possible that there are no channels that meet that user's requirements. In that case, the user may opt to submit a channel request to the Baton module. Baton participants will periodically read channel requests to determine if there are any channels that need to be created. Figure 12 shows the format of a channel request. The destination chain field tells the participant which chain should be at the other end of the channel. The policy field is the channel version that must be used when creating the channel. The creation tip and the validation tip represent the amount of gas tokens that will be paid to the participant that created and validated the channel, respectively. The channel ID field starts off blank, and will be atomically set once the corresponding channel has been validated. The memo field can be used to inform participants if there are any additional properties the channel should have. For example, the user may want a channel to the destination that has as few hops as possible.

Channel Request	
Timestamp	
Destination Chain	
Policy (channel version)	
Creation Tip	Validation Tip
Channel ID (initially blank)	
Memo	

Figure 12: Channel request format

4.4.2 Specifying Security Policies

IBC channels are given a version as a string when they are created. IBC allows this version to be anything the user wants, as interpretation of a version is left up to the underlying modules. We provide the syntax that Baton uses to encode security policy information in channel versions.

`<app version>/<policy 1>:<metric 1>/<policy 2>:<metric 2>/...`


Figure 13: Syntax for channel versions in Baton. The application version is first, followed by a forward slash, then by zero or more security policies. Policies are also delimited by forward slashes. Each policy as a policy type, followed by a colon, followed by a requirement.

Figure 13 shows the syntax for Baton policies. The first term represents the version for the underlying application. Following that is a sequence of policies delimited by forward

Policy Type	Requirement Type	Usage	Proof Strategy
validators	integer	The requirement indicates the minimum amount of validators a blockchain must have.	Where n is the given requirement, provide n proofs of inclusion for n distinct validators.
hops	integer	The requirement indicates the maximum amount of connection hops a channel can have.	Where n is the requirement, ensure that the number of connections hops for the channel is $\leq n$.

Table 1: Types of security policies

slashes. Each policy contains two parts, and each part is separated by a colon. The first part is the policy type. These types are predefined and are expected to be known and understood by the source chain, the destination chain and the relayers involved. The second part is a requirement that is used to determine whether or not a blockchain adheres to the given policy. Requirements are policy specific. Table 1 describes two policies that were enforced while evaluating Baton. Later in Section 4.4.4, we describe how to enforce new policies.

4.4.3 Creating A Channel

The Baton participant follows the IBC specifications for creating a multi-hop channel (ics-33) [21]. Since ics-33 is a draft specification, it is not finalized and there are no publicly available, production ready implementations. Consequently, we built upon a prototype implementation that was provided by Polymer Labs [40]. Baton modifies this process by including the security policy information from the channel request in the channel version. Consider the following example:

“ics-20/validators:8”

This channel version encodes two things.

- **ics-20:** As per the IBC specifications [21], this is the official version for cross-chain token transfer applications. This indicates that the underlying module performs token transfers.

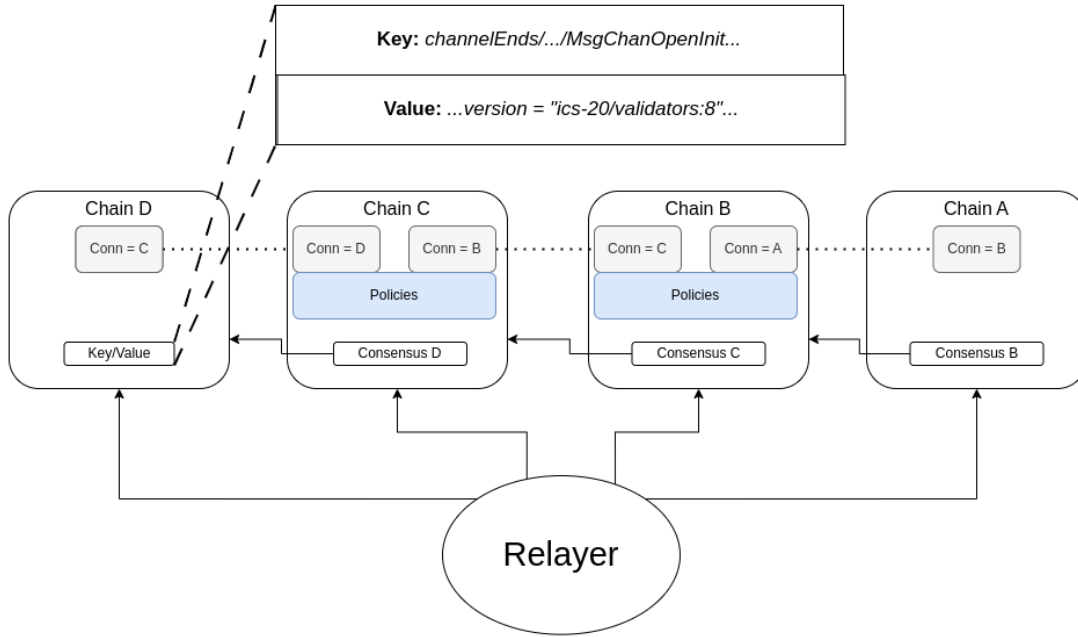


Figure 14: Relayer’s view when relaying messages along a channel

- **validators:8:** This indicates that all chains along the path must have at least 8 validators.

Figure 14 illustrates the process of relaying for channels that enforce security policies. Relayers know the version of the channels they relay for, as IBC requires that this information can be queried from blockchain server nodes. When the relayer sees “validators:8” in the channel’s version, it knows to append the additional policy proofs to the message. When the message is ultimately submitted on the destination chain, that chain knows to verify the additional proofs because it also sees the channel version.

4.4.4 Adding New Security Policies

All security policies must be provable either by inspecting a blockchain’s state, such as counting the number of validators, or by inspecting the proof itself, such as verifying hop count on the channel path. For policies that inspect blockchain state, relayers and blockchains need to know how to construct and verify the specific policy proofs. Sometimes the relevant security parameters already exist in the blockchain’s state. In that case, the

relayer must query the Merkle proofs for that additional data, and append it to the original message proof. In our experiments, we prove how many validators there are by querying validator set information from Cosmos’s staking module. However, other types of proofs might require updating existing modules.

For example, a useful security parameter in blockchains is the Nakamoto coefficient [51]. The Nakamoto coefficient represents the minimum number of validators that can collude to compromise a blockchain. Proving that the Nakamoto coefficient is some value n requires knowing the stake of the top $n - 1$ validators, and knowing the total amount of tokens staked. While the staking module does keep the order of validators in terms of staking power, the staking module does not provide an RPC endpoint that can be queried for a proof of this order. Therefore to prove a Nakamoto coefficient of n , one could modify the staking module so that a list of the top $n - 1$ validators of that blockchain can be queried from it.

Section 2.5.3 describes how Merkle proofs are constructed, and explains how to verify a Merkle proof for a given key/value pair. The relayer will query an RPC endpoint for each of intermediate blockchains to receive the corresponding Merkle proofs for the Nakamoto coefficient. That proof will be submitted and verified on the destination chain along with the original proofs as described in Section 2.5.6. Once verified, the destination chain must unmarshal and interpret the proof. In this case, the destination chain will read that the sum of the tokens staked by the top $n - 1$ validators is less than 1/3rd of the total tokens staked.

4.4.5 Module Policy Requirements

While the messages sent along a particular channel will only be processed when all the blockchains along that path adhere to that channel’s policies, there can be multiple channels connected to a module. Certain modules, such as those that perform token transfers, send scarce digital assets between chains. By lying about the amount of an asset they possess on the source chain, an adversarial user can benefit by acquiring more of an asset than they otherwise should on the destination chain. In order to prevent exploits such as this from happening, all channels connected to a module must be secure.

Baton allows a module to set baseline security requirements such that all channels that connect to it must meet those baselines. An example requirement is, all else being equal, a blockchain with a Nakamoto coefficient of 8 is no less secure than a blockchain with a coefficient of 7. In this example, a module can require that all channels have a policy where the Nakamoto coefficient is at least 7. The relayer will use the same methods presented in

Section 4.4.3 and Section 4.4.4 to append the policy proofs to the channel open handshake messages. Each module can therefore verify that their baseline security requirements have been met, and will refuse to complete the handshake if the requirements are not met.

4.5 Validating IBC Channels as Part of Baton

4.5.1 Validation Governance Vote

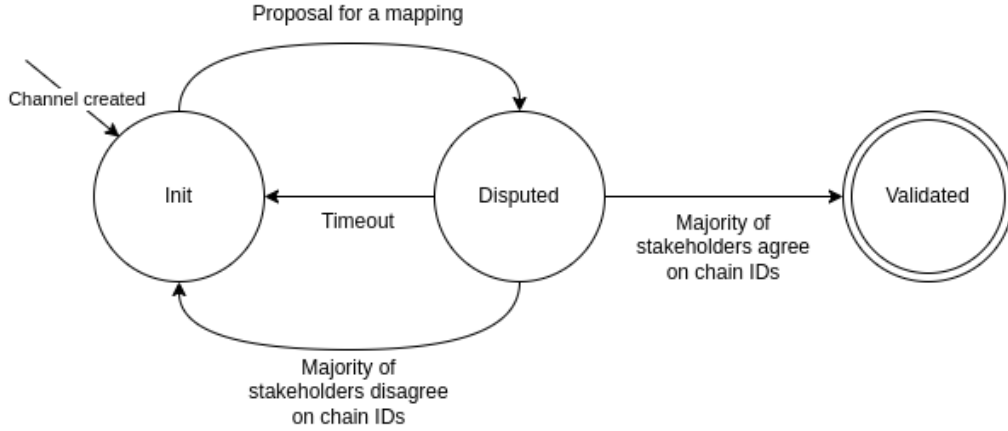


Figure 15: Life cycle of validating a Baton channel.

Figure 15 demonstrates how a participant can validate a channel by identifying the blockchains along that channel. When a Baton participant would like to validate a channel, using Figure 14 as an example, they will propose a governance vote for whether or not that channel has hops $ChainD \rightarrow ChainC \rightarrow ChainB \rightarrow ChainA$. The participant must simultaneously deposit tokens to the Baton module as collateral. Stakeholders on $ChainA$ will vote yes or no for whether or not the channel traverses those chains. A stakeholder is any individual who is either a validator, or who has delegated tokens to be staked and used by validators. If the governance proposal is accepted, the participant's tokens will be refunded, the channel will be mapped to $ChainD \rightarrow ChainC \rightarrow ChainB \rightarrow ChainA$ and the channel ID field of the respective channel request will be updated. If the proposal is rejected, the participant will lose their tokens and no mapping will be created.

4.5.2 Incentives for Correctly Identifying Blockchains

Section 2.6.3 explains that it is important for users to correctly identify the intermediate blockchains along a multi-hop IBC channel to avoid accidentally using insecure or costly routes. Consequently, there must be incentive for stakeholders to vote on the correct chain ID mappings for Baton channels. Fortunately, these mappings are determined by the source and destination chains which are assumed to be honest. In general, higher activity on a blockchain results in higher gas fees for users and greater rewards for stakeholders. It is therefore in the stakeholders' best interest to encourage blockchain activity, which means ensuring that the blockchains along a channel are correctly identified.

4.5.3 Validating Channels With Recurring Chains

Although governance votes allow stakeholders to collectively agree on the IDs of blockchains along a multi-hop channel, governance votes can take months to complete [20], and require manual intervention from many independent parties. Preferably, a vote should not be required every time a channel is created. While the same blockchain can be used as an intermediate hop for multiple channels, Baton fortunately requires only one governance vote to permanently identify that blockchain. In this section, we present how a blockchain that has been identified in a previously validated channel can be re-identified as part of a new channel without requiring a governance vote. We demonstrate how Baton prevents IDs from being reused for multiple different blockchains.

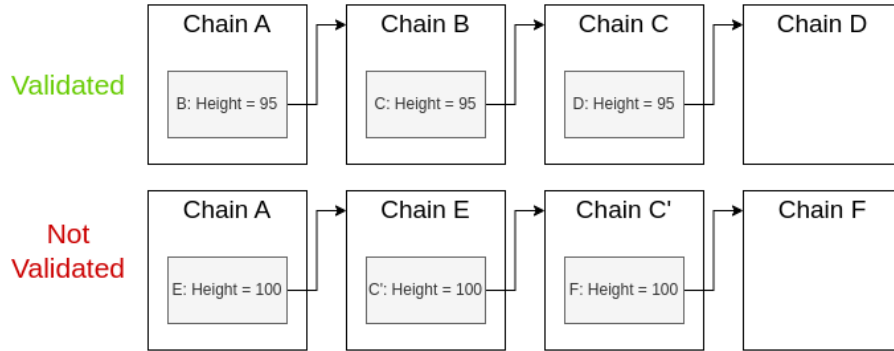


Figure 16: Two channels, both sharing the same blockchain as the third intermediate chain.

Figure 16 illustrates two channels, where only the first channel has been validated as a Baton channel. Since the blockchains along a validated channel have been identified, a

Baton participant can use the validated channel to identify the chains that are part of the second channel. In this example, Baton can prove that C and C' are different blockchains by supplying differently signed headers for C and C' at the same height. Proving that C' and C are the same involves supplying same signed header from both blockchains at the same height.

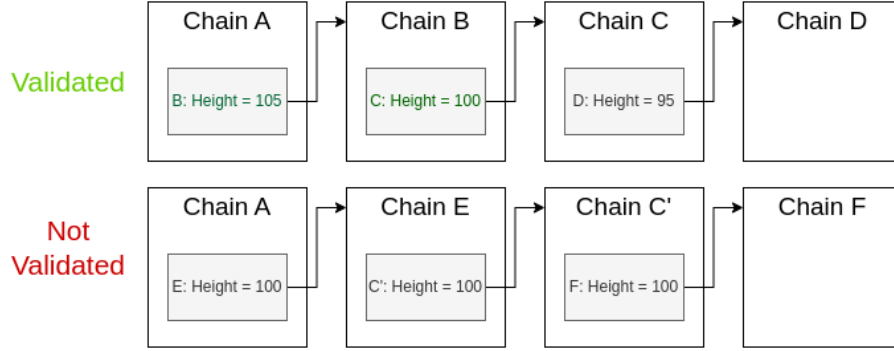


Figure 17: Light clients for the validated channel are updated such that blockchains B and E have the same height stored for C and C' respectively.

When a Baton participant on *ChainA* wants to identify blockchain C' on the second channel, they must first deposit collateral to the Baton module. Following that, the participant will update the light clients on either channel so that the neighbours of C and C' closest to *ChainA* view the same height of C and C' . Following that, a multi-hop proof along both channels up to C and C' can be submitted on *ChainA*. If both proofs are valid, *ChainA* will then compare the consensus states of C and C' . If they differ, the participant's deposit will be lost. If the consensus states are the same, the Baton module on *ChainA* will assign C' with the ID *ChainC*, and the participant's deposit will be returned. This process is repeated for each intermediate chain until they are all identified.

4.5.4 Rewarding the Participant

When validating a channel, a Baton participant will need to perform numerous transactions on each of the blockchains along that channel. These transactions can be costly. Upon completing the validation process, the participant will be rewarded the validation tip from the corresponding channel request on the source chain. Unfortunately, this tip might not cover the entire cost of validating the channel. Without sufficient compensation, there are no cryptoeconomic incentives for participants to validate channels. Furthermore, there are

no cryptoeconomic incentives to ensure these routes are near-optimal according to a user-specified metric, whether it be the hop count or the gas fees. In this section, we explain how Baton incentivizes participants to validate near-optimal channels.

A participant’s account information is stored on-chain since they must submit transactions to the Baton module when validating a channel. Whenever a user sends a message along a channel, a small fee can be taken from that user and paid to the participant that validated that channel. That way, participants will be compensated for all traffic that travels along channels they have validated. Participants compete with each other to have their channels used, so they need to make their channels as appealing as possible to blockchain users. Participants can do this by ensuring their channels are near-optimal according to the users’ needs. Baton users will have different definitions of optimal. For instance, one user may want a channel with as few hops as possible, and another user may want a channel that has the lowest fees. Baton users can specify this information in the memo field of their channel request. Baton participants can read the memo field to determine what users may consider to be optimal. This information will inform the participant’s decisions when creating and validating new channels.

4.5.5 Limitations of Rewarding Participants for Channel Use

Although creating and validating a channel only costs a participant a fixed amount of tokens, that participant may collect an arbitrarily large amount of fees when their channel is repeatedly used. This could lead to situations where a participant passively earns tokens long after they have contributed any value to the protocol. There are circumstances where blockchain users, such as validators, continuously earn tokens. However, this usually comes in return for providing perpetual value, such as the added security that comes from a validator’s staked tokens. Consequently, it would be beneficial to explore other methods for incentivizing near-optimal route generation. These new methods should not pay participants that are no longer providing value to the protocol.

One approach could be to have a challenge period during which other participants may validate different channels that address the same channel request. Rather than verify that any given channel is optimal, which requires complete knowledge of the blockchain network’s connectivity, the user can select and reward the participant that supplied the most cost effective option. The benefit of this approach is that a participant will only be paid for the value they provide. Some downsides, however, are that some participants will not be compensated for the work they do, and users may have to wait for the challenge period to end before sending their messages.

Another potential approach is that the user can supply their cost matrix and a maximum allowable cost in the channel request’s memo field. When a channel is validated, the cost of that channel must be less than or equal to the maximum allowable cost with respect to the cost matrix. Unlike the previous approach, this allows the channel to be used immediately. However, this does not incentivize the participant to choose a route that is near-optimal, as the route must just be less costly than the maximum allowable cost.

4.6 Splitting Messages Along Multiple Disjoint Paths

4.6.1 The Security Risk Compromised Chains Pose

In general, it is impossible to verify whether a blockchain performs the correct state transitions unless if you are running a full node, where a full node is a server that stores the entire blockchain state, and that executes all blockchain transactions [53]. Blockchains along a multi-hop channel will only use a light client to verify and store the consensus states of direct neighbours. Consequently, Section 2.5.6 describes that the consensus state of one blockchain can be used to prove the existence of a given consensus state of the previous blockchain along the channel. A blockchain that performs incorrect state transitions can store an arbitrary consensus state for its neighbour, and consequently claim that its neighbour has sent arbitrary messages. Baton helps mitigate these types of exploits where compromised chains fabricate messages. Baton achieves this by establishing many disjoint channels between the source and the destination chain. When a message is sent along disjoint channels, the destination chain can verify that the messages it has received along each path are the same. In this section, we present Baton’s algorithm for reconciling two channels as connecting the same blockchains. We call this the “Multi-Path Handshake Algorithm”. Given that a channel has been validated as described in Section 4.5.1, this algorithm determines other channels that connect the same endpoint blockchains.

4.6.2 The Multi-Path Handshake Algorithm

Let A be an honest source chain that has a channel c_{base} that is connected to another honest chain B . The objective of the multi-path algorithm is to create a channel that is connected to the same endpoints as c_{base} . If blockchains A and B are on either side of the newly created channel, communication along that channel will cause state updates on

both blockchains. This allows both chains to use c_{base} to confirm that they, and not some other blockchains, are on either side of the new channel.

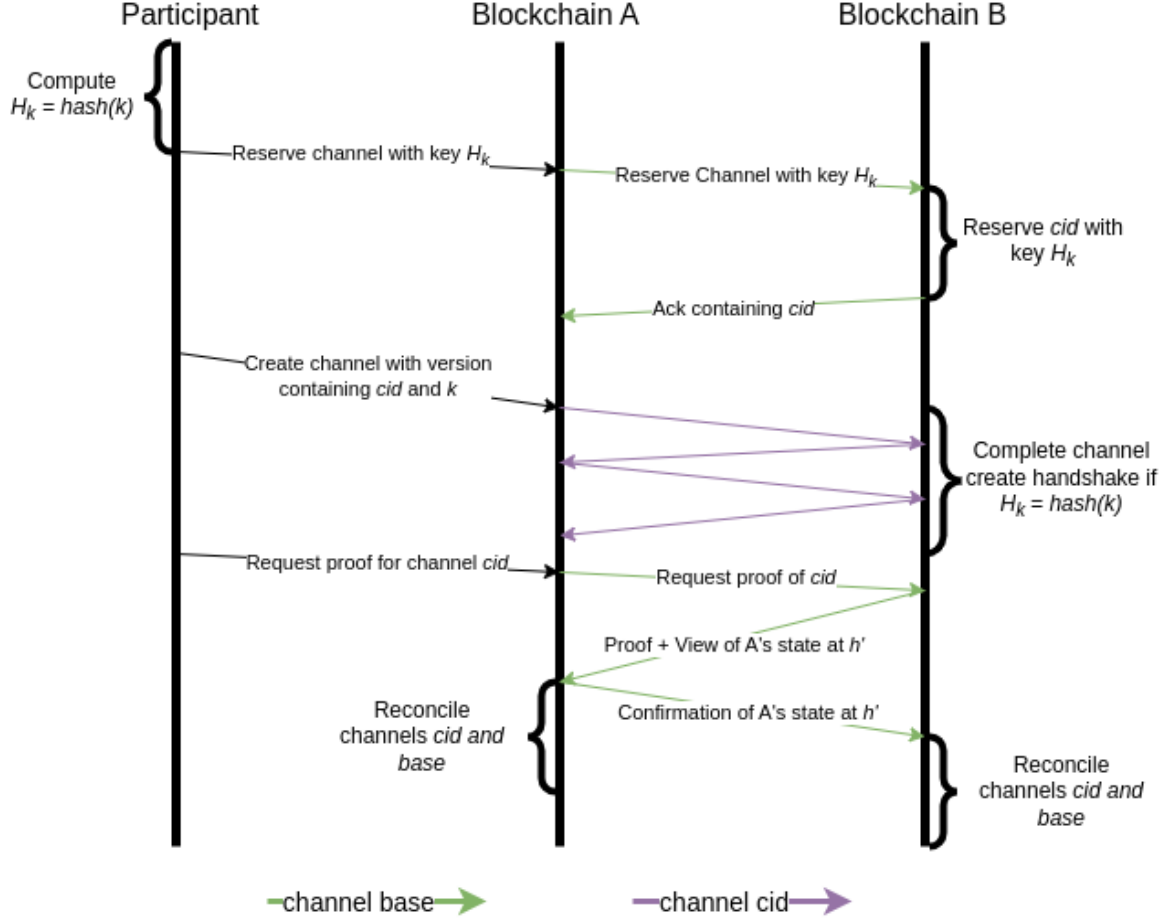


Figure 18: Overview of multi-path handshake algorithm

Figure 18 presents an overview of the multi-path handshake algorithm. Since A and B are honest, their validators are not compromised, and the relayers cannot alter packets by forging signatures as a consequence. First, a channel with the ID cid will be created between A and B . Following that, B will use c_{base} to prove to A that it has created c_{cid} , and that it believes A is on the other end of c_{cid} . Blockchain A will verify that blockchain B has created c_{cid} , and use c_{base} to prove to B that it is on the other end of c_{cid} . The algorithm is as follows:

1. A participant who would like to establish another multi-hop channel between A and B chooses a key k offline. The participant computes $H_k = \text{hash}(k)$.
2. The participant sends a message via c_{base} containing H_k asking B to reserve a channel id.
3. B reserves channel id cid and sends an acknowledgment containing cid back to A .
4. The participant now initiates a multi-hop channel open handshake with $key:cid:k$ as a part of the version.
5. Upon seeing the version, B checks if there is a channel id reserved with the value $\text{hash}(k)$. If so, B will continue the handshake with cid as its channel endpoint ID. If not, B will choose a new channel ID.
6. Both blockchains will complete the channel open handshake.
7. Upon handshake completion, the participant requests for a proof of inclusion of a channel with id cid from B via c_{base} . A sets a timeout.
8. B sends a proof back to A along with its view of A 's state at height h' . B can select one of the consensus states it saw when processing a channel open handshake message. This information is sent via c_{base} , and B will set a timeout.
9. Once A validates the proof of the new channel and that the consensus state is its own, A marks c_{base} and c_{cid} as reconciled. A sends confirmation via c_{base} that B saw the correct consensus state for A . If B saw the incorrect consensus state, A will not reconcile c_{base} and c_{cid} , and will notify B via c_{base} .
10. B will reconcile both channels only if it receives confirmation to do so from A .
11. If a chain's timeout expires at any point before receiving the expected response, that chain will cancel the algorithm and remove all relevant state information.

Liveness is ensured as long as there is a relay submitting messages between A and B , and each chain maintains a timeout.

4.6.3 Catching Misbehaviours

Although the multi-path handshake algorithm will reconcile paths that are not disjoint, we need to use disjoint paths for catching misbehaviours. If multiple paths share a chain, that chain could be compromised and fabricate message along each path. With disjoint paths, compromised chains along each path would need to collude to deceive the destination chain.

One way to catch misbehaviours is to send the same message along multiple disjoint paths. Relayers will generate different multi-hop proofs for the same message. IBC enforces exactly once delivery. Since the endpoint chains are assumed to be honest, it is impossible for the same key to have multiple values. Therefore, if there are differences in the messages sent along the disjoint paths, the destination chain knows that at least one of the channels contains a compromised chain.

Alternatively, messages can be sent along one channel. When an honest observer suspects that a fabricated message was submitted to the destination chain, they can send that message again from the source chain on one of the disjoint paths. Since the message was fabricated, it does not exist on the source chain. Therefore, the source chain will allow the new message to be sent, and either the honest observer or a relayer will deliver it on the destination chain. The destination will validate the proof and see that it has already received a different message for that key. The destination chain now knows that one of the channels contains a compromised chain.

The destination chain will not know which channel is compromised, since it cannot validate the state changes of the other chains. Consequently, the destination chain can invalidate all of the channels involved. At this point, on-chain governance is required to determine what to do moving forward. At the very least, we recommend that the adversary that submitted the fabricated message have a significant amount their tokens burned.

4.6.4 Limitations When Selecting Disjoint Paths

A limitation to this approach is that at least one disjoint channel must not contain any compromised blockchains. Unfortunately, it is very difficult, if not impossible, to verify that a channel is honest without running a full node for each blockchains along that channel [53]. Even though this approach therefore cannot guarantee that fabricated messages will be detected, it makes it easier to detect such messages. Since Baton is permissionless, anyone can establish disjoint channels as needed. The greater the number of disjoint channels between blockchains, the smaller the likelihood that every channel is compromised. It

therefore becomes harder for falsified messages to go unnoticed, as there would likely exist an honest disjoint channel that could be used to detect the misbehaviour.

Chapter 5

Evaluation

We evaluate Baton’s performance in this section. Current approaches for large-scale blockchain interoperability require IBC enabled blockchains to communicate with each other via a hub. Cosmos Hub [18] and Axelar [5] both allow IBC enabled blockchains to communicate with any other IBC enabled blockchain. Both Cosmos Hub and Axelar are hubs that other chains must connect to. Despite the existence of hubs, blockchains still form IBC connections directly with each other. Unlike Cosmos Hub and Axelar, Baton can route messages between chains that are indirectly connected to each other. First, we measure Baton’s decentralization by determining the percentage of chain pairs that can still communicate when the most popular blockchains become unavailable. We then demonstrate how Baton introduces far less congestion compared to how IBC is currently deployed. Following that, we compare multi-hop channels to single-hop channels. This establishes a case for why, all else being equal, the multi-hop channels that Baton creates are better to use than single-hop channels. Next, we evaluate the cost of including Baton’s policy proofs in cross-chain messages. Lastly, we measure the amount of light client updates relative to the amount of messages delivered. By doing so, we argue that the decentralized and secure communication that Baton provides does not come at the cost of being less efficient than competing approaches.

It is infeasible to benchmark IBC on public blockchains, as they are monetarily expensive to use. There are public blockchains that are meant for testing purposes, however these blockchains limit the amount of tokens they distribute each day. Instead, we replicate real-world conditions to the best of our ability using a custom configuration of blockchains.

We use the gas cost and the block time to measure how efficient cross-chain operations are. Cosmos chains compute the gas cost as the amount of data read from and written to a

blockchain’s store. Although there are many ways to compute resource usage [4], we choose this method because it is consistent with how Cosmos Hub and other Cosmos based chains compute gas. Consequently, it is representative of how gas would be computed in many public IBC enabled blockchains. Fischer, Lynch and Paterson have proven that timing assumptions are necessary to guarantee consensus in a system where there is at least one faulty process [27]. Due to the timing assumptions for CometBFT based blockchains, block times are usually fixed with very little variance. We cannot reliably compare the time in seconds for cross-chain operations, as that will change based on a blockchain’s configured consensus parameters. Therefore, we measure time in terms of the number of blocks added to the blockchain.

5.1 Experimental Setup

We use four servers when evaluating Baton. Each server has two Intel E5-2620v2 CPUs, 64 GB of RAM, three 1TB SATA3 hard drives, an Intel S3700 200GB SATA3 SSD, two Intel i350 gigabit ethernet ports and a Mellanox 10GbE SFP port. All of the servers reside on the same internal network.

We use Docker swarm to deploy Baton blockchains across each of these servers. Every blockchain node and relayer runs in its own container. We use Docker swarm’s default configuration which distributes the containers evenly across each server. Consequently, every server runs approximately the same number of relayer containers and blockchain node containers. Three of the servers are running Docker engine major version 27, and the fourth is running major version 24.

Baton was developed using Cosmos SDK. We use a version of the Go relayer that was modified by Polymer Labs to support multi-hop channels. We further enhance the Go relayer to be able to read channel security policies, and dynamically create the necessary policy proofs. A study has shown that using multiple relayers at the same time can hinder performance, as a lack of inter-relayer coordination leads to conflicting IBC transactions [15]. We allow the Go relayer to cache light client updates and to compute, using past IBC client states, whether or not updates are required to generate proofs. We have found that this reduces relayer conflicts, reduces the number of duplicate transactions, reduces gas fees and speeds up cross-chain communication as a whole. In order to provide a fair comparison, we use this improved Go relayer throughout the entire evaluation.

Blockchain nodes can be distributed anywhere across the world. In order to emulate the effects of latency, we have added a propagation delay of 80ms between every pair of

nodes. CometBFT relies on timing assumptions during the voting processes, as excessive propagation delay can result in consensus being delayed. In order to provide realistic assumptions, we use the same consensus parameters as Cosmos Hub’s testnet [19]. As a result, the block time is approximately 6 seconds. The specific parameters used are as follows:

Timeout Parameter	Value
Propose	3 seconds
Propose Delta	500 milliseconds
Prevote	1 second
Prevote Delta	500 milliseconds
Precommit	1 second
Precommit Delta	500 milliseconds
Commit	5 seconds

Table 2: Timeout parameters used for CometBFT consensus protocol.

Going forward, we use the term “network topology” to denote a configuration of Baton enabled blockchains and IBC connections. We used several network topologies depending on the type of evaluation. When evaluating how the number of hops affects the cost of establishing channels and sending messages, we connect eight Baton chains together side-by-side to form a line. When evaluating the n^2 mesh topology, we launched 20 chains and connect each chain to every other chain. When evaluating the hub-and-spoke topology, we connected 20 chains to a single central hub chain. The comparison topology also consisted of 20 chains. Each topology has been illustrated in Figure 25.

There are two critical observations that impact the results from the evaluation. Firstly, given the same input transaction, identically configured validators will compute the same gas costs. While there will be some variance in the gas costs when performing the same type of operation, such as a light client update, this variance is due to slight differences in the amount of bytes being processed. Therefore, there is a narrow upper-bound and lower-bound on how much variability there can be. The second observation is that it is impossible to guarantee consensus in an asynchronous environment where messages can be delayed and where there is at least one adversarial participant [27]. Consequently, there are necessary timing assumptions for the consensus protocol. These timing assumption are provided in Table 2. Baring timeouts and excessive network congestion, the block time it takes to perform a particular operation is generally fixed. Consequently, we have found that running the same experiment multiple times given the same network conditions will yield negligible, if any, differences in gas costs and block time costs. In order to

consistently measure gas costs and block time, we use the results from at least two runs of each experiment where there were little to no timeouts.

5.2 Decentralization

Since the correctness of decentralized protocols relies on the integrity of multiple independent parties, a decentralized protocol will not run correctly only if a significant number of independent parties fail or are unavailable. All else being equal, a decentralized protocol is more resistant to catastrophic failure than a centralized protocol, as a centralized protocol will stop working when smaller number of individuals fail or are unavailable. In this section, we measure how decentralized Baton is by determining the percentage of chain pairs that can still communicate as blockchains become unavailable for any reason.

Since Baton communicates across channels that span multiple IBC connections, two blockchains can communicate as long as there is a sequence of IBC connections connecting them. We first query Cosmos-based blockchain endpoints [17] to obtain the approximate state of IBC enabled blockchains in April 2024. In total, we evaluate 124 blockchains connected using 1115 IBC connections. We determine a blockchain’s popularity by how many IBC connections it has. Finally, we remove the top X blockchains, where $0 \leq X \leq 10$, and compute how many chain pairs are still connected.

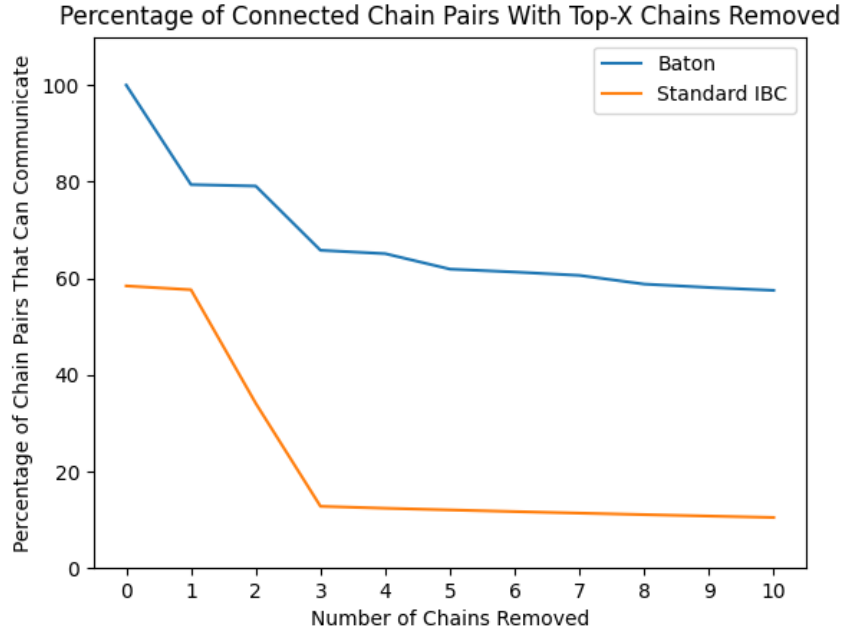


Figure 19: Percentage of blockchain pairs that are still connected even with the top X most popular blockchains removed. X ranges from 0 to 10.

Figure 19 demonstrates how network connectivity is effected as the top X most popular blockchains are made unavailable. Even with the top 10 most connected blockchains unavailable, Baton would allow 57.5% of blockchain pairs to communicate. This includes when both Cosmos Hub and Axelar are removed. Blockchains using standard IBC must communicate using Cosmos Hub, Axelar or direct connections. Although the most connected blockchain in the Cosmos ecosystem is Osmosis, Osmosis does not forward messages between chain pairs like Cosmos Hub or Axelar. Consequently, removing Osmosis has little effect on the overall connectivity of the network. However if Cosmos Hub or Axelar, which are the second and third most connected blockchains, become unavailable, only 32.8% and 48.2% of chain pairs can communicate, respectively. Figure 19 indicates that approximately 58.4% of chain pairs can connect with both Cosmos Hub and Axelar present. However, only 34.1% of chain pairs can communicate when Cosmos Hub and Osmosis are removed, and only 12.8% chain pairs can communicate when also removing Axelar. Therefore, Baton is more decentralized than the hub-and-spoke approach used by both Cosmos Hub and Axelar.

5.3 Congestion

Messages sent in the current Cosmos ecosystem must be manually delivered to a hub chain, such as Cosmos Hub and Axelar, before ultimately being delivered to the destination blockchain. This introduces significant congestion as the send rate of cross-chain messages increases, since each message must be committed to the hub chain. Increased congestion is undesirable, as it forces blockchain users to pay higher fees as their transactions compete for a spot in a block. In contrast, Baton only needs to update the light clients for the intermediate chains as described in Section 2.5.6 when delivering multi-hop message. Consequently, Baton reduces congestion by not needing to submit the message itself on each intermediate chain. Since our custom relayer caches and reuses light client updates for subsequent messages, we are able to further reduce congestion by not submitting unnecessary light client update transactions.

In this section, we compare the congestion caused by Baton to the congestion caused by standard IBC. We implement a discrete event simulation of cross-chain messaging across the topology of IBC-enabled Cosmos chains from April 2024. First, determine every pair of chains that are either directly connected, or connected via Cosmos Hub and Axelar. Then, we send a total of 50000 messages, where the send rate per blockchain pair ranges from one message every 25000 milliseconds, to a message every 1563 milliseconds. Finally, we use the maximum number of transactions found in any one block to indicate congestion.

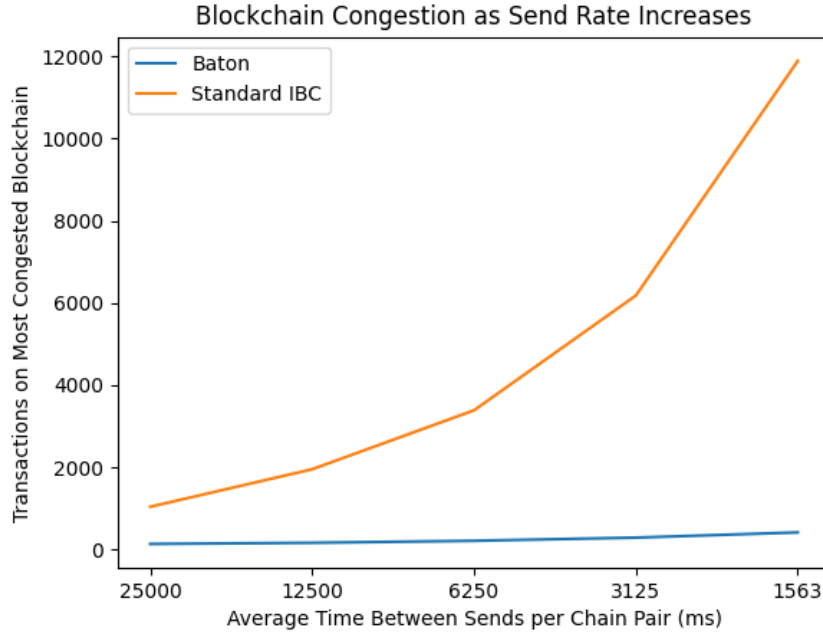


Figure 20: The maximum amount of transaction on a particular block height for the most congested blockchain. We send a total of 50000 messages, where the send rate per blockchain pair ranges from one message every 25000 milliseconds, to a message every 1563 milliseconds. Each chain’s block time is 4 seconds.

Figure 20 demonstrates that Baton introduces far less congestion than standard IBC, and is therefore more scalable as the rate of cross-chain transactions increases. The most congested blockchain ranges from having 1037 to 11890 transactions in a given block when using standard IBC. In contrast, the most congested blockchain ranges from having 130 to 415 transactions in a given block when using Baton. Baton requires a chain to have only one client update per connection per block. Therefore, the number of light client update transactions per block is upper bounded by the number of IBC connections a blockchain has. As the send rate increases, the congestion on a Baton-enabled blockchain grows proportional to the number of messages sent and delivered on that chain per block.

5.4 Single-Hop Channels Vs. Multi-Hop Channels

In this section, we compare the cost of using single-hop channels and multi-hop channels.

5.4.1 Establishing Channels

In this section, we compare the cost of establishing the channels required for pairs of chains to communicate. Where standard IBC requires chains to be connected pair-wise using single-hop channels, Baton will establish multi-hop channels to connect the source and destination chains. In general, establishing channels incurs a one-time cost. Once a channel exists, it can be used indefinitely unless closed. Since light client updates are performed at the connection level, there is no additional cost to keeping idle channels open. Consequently, channels are rarely closed in practice. In fact, certain applications, such as token transfers (ics-020) [21], require that channels remain open indefinitely.

To evaluate the cost Baton incurs by creating these channels, we generate two channels for every amount of hops from 1 to 7, and evaluate the average amount of time and gas to create these channels. Currently, every unit of gas on Cosmos Hub equates to around 0.025 μ atoms on average [17]. Considering that every atom costs around \$5 USD, one million gas equates to around \$0.125 USD spent [22].

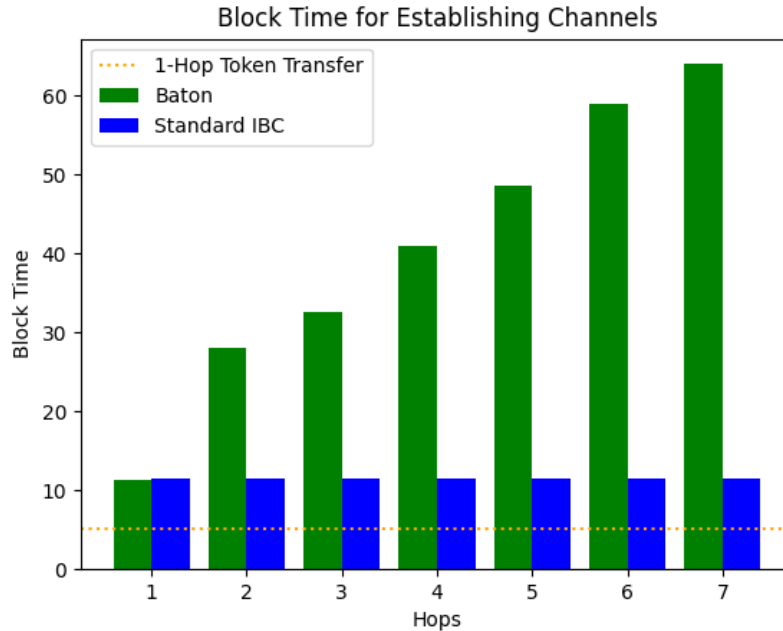


Figure 21: Average block time required to establish channels between pairs of chains. Since each hop is created independently, the best case scenario for standard IBC is to create every single-hop channel concurrently.

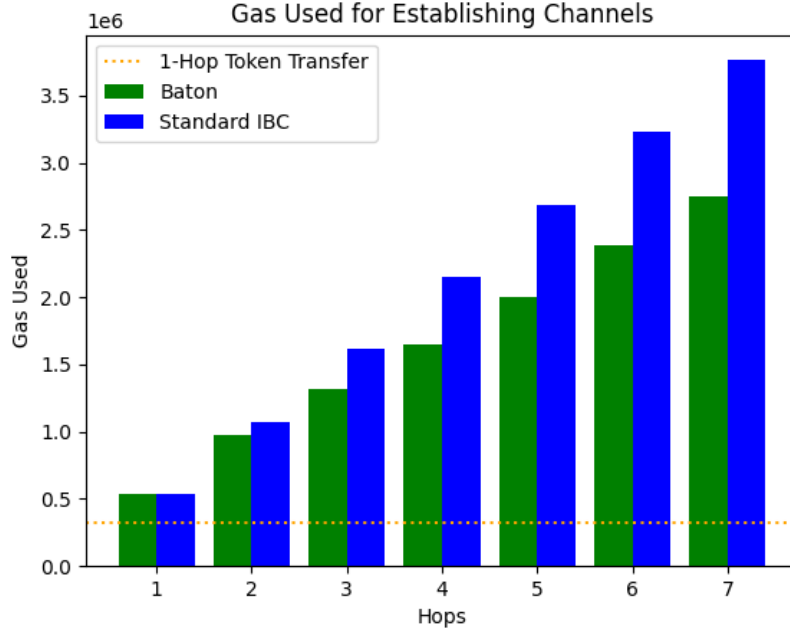


Figure 22: Gas required to establish channels between pairs of chains.

Figure 21 shows the time required to establish single-hop and multi-hop channels that span 2 to 8 blockchains. The best case scenario for standard single-hop IBC is that there is a separate relayer managing each hop along a path. If this is the case, every relayer can construct their respective single-hop channel simultaneously. Each channel works independently and does not require information from any other channel. Consequently, It takes a constant amount of time to establish the required single-hop channels for standard IBC regardless of how many hops there are. However, this is highly atypical as it requires more coordination as the number of hops increases. A more likely scenario is that one individual will create the channels for each hop. It is difficult to determine how long this would take, as that individual can still construct channels in parallel. However, the amount of computation grows linearly in proportion to the number of hops.

The time it takes for Baton to establish the required channel grows linearly as the amount of hops increases. This occurs because Baton must construct each hop of the requisite multi-hop proof one after the other. There is a sharp increase from 1 hop to 2 hops due to how the Go relayer is implemented. For single hop operations, the relayer will include the client update message and the operation itself within the same transaction. For multi-hop operations, the relayer will commit the operation in a separate transaction

after first performing the client updates in earlier transactions.

As shown in Figure 22, Baton requires less gas than single-hop IBC. While both approaches require the same amount of light client updates, Baton’s improvement comes from not having to submit the `MsgChannelOpenInit`, `MsgChannelOpenTry`, `MsgChannelOpenAck` and `MsgChannelOpenConfirm` messages on the intermediate chains as described in Section 2.5.5. Thus, the additional cost for using single-hop channels grows linearly in proportion to the number of hops between the source and destination chain.

5.4.2 Sending and Verifying Messages

We compare the cost of sending messages. We use the same topology as described in the previous section, where 8 chains are connected to form a line.

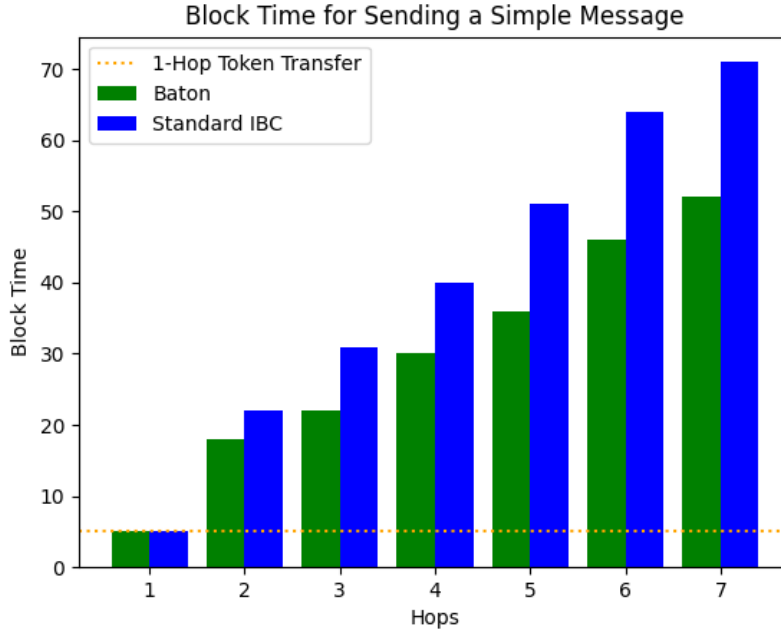


Figure 23: Average block time required send a simple message between two chains.

When using single-hop channels, we send a message pair-wise one hop at a time along the path. we wait for the acknowledgment to be received on the sending chain of that hop before moving to the next hop. This is required since many protocols, such as ics-20 token

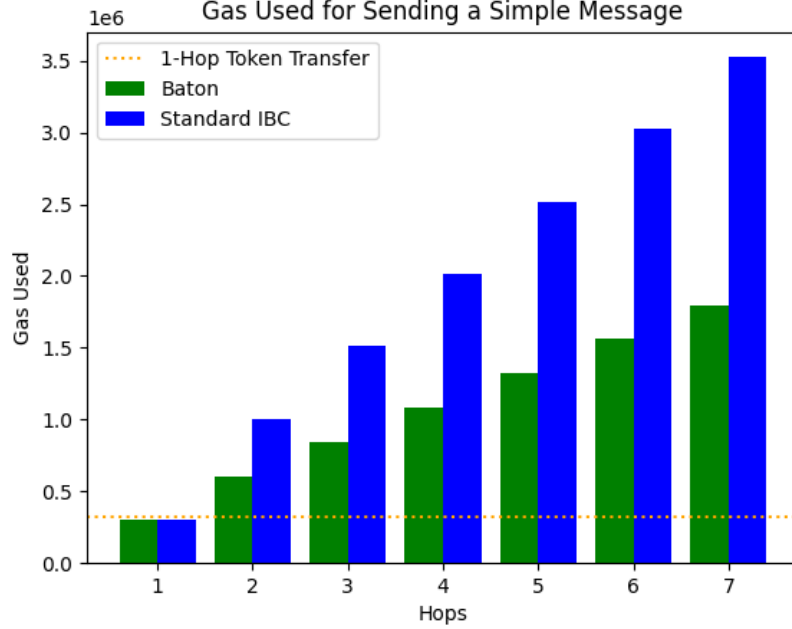


Figure 24: Average gas used by relayers when sending a simple message between two chains.

transfers [21], use acknowledgments to detect and rectify errors. We perform the same single-hop sends in the reverse direction in order to emulate an acknowledgment being sent back to the original source chain. Since Baton uses multi-hop channels, we only need to send a single message on the respective channel on the source chain, and wait for the corresponding acknowledgment. We use the block height of the source chain as a point of references when measuring the block time for sending messages. The resulting block time is the difference in the latest block height of the source chain before and after the send.

In Figure 23, we see that the block time for sending messages in both cases grows linearly in proportion to the number of hops. Just like when establishing channels, there is a spike in block time from 1 hop to 2 hops. Once again, this is due to the relayer sending the client update and the message itself in the same transaction for single-hop channels. Baton is faster because multi-hop channels will only commit a receive message on the destination chain, and an acknowledgement on the source chain. Using single-hop channels requires waiting for these additional receive and acknowledgment messages to be committed on the pair of chains at each hop.

As previously mentioned, we need acknowledgements for each hop in the single-hop case for error handling. Those acknowledgements are application specific and can be any size. From a gas cost perspective, it is best for these acknowledgements to be as small as possible. We therefore omit the cost of these additional acknowledgments in the single-hop case in order to give that approach as much of an advantage as possible. Despite that, Figure 24 shows that Baton still uses less gas than the standard approach. This is due to the additional client updates that the standard approach cannot avoid. When sending the message along the path in the forward direction, there are 2 client updates per hop (once for the receive and once for the acknowledgement). Likewise, there are 2 updates per hop in the reverse direction. Baton requires only 2 updates per hop in total. Consequently, the standard approach requires twice as many client updates as Baton. Although the standard approach has more than twice as many transactions as Baton, Baton’s receive and acknowledgment messages are more expensive than they would be with single-hop channels. This is because the multi-hop proofs included in Baton’s messages are larger than single-hop proofs. All else being equal, using a multi-hop channel is just over half the cost of as using a series of single-hop channels.

5.5 Including Policy Proofs

In this section, we evaluate the cost of including security policy proofs. For our evaluation, we use Cosmos Hub as the example chain. The example policy requires proof that every blockchain has at least 100 validators.

It is possible that security properties for an honest chains will change rapidly. However in practice, security properties change very slowly. Consider the policy where we check that every blockchain has at least a certain amount of validators. Since the message complexity of CometBFT increases quadratically proportional to the number of validators, only the top 180 stakeholders will be validators on Cosmos Hub. Despite this, there are hundreds of other stakeholders that can become a validator if a current validator decides to unstake their token, or if a validator drops out of the top 180. While it is technically possible for the number of validators to drop below 180, we have found that this does not happen in practice. By using the rpc endpoint at “cosmos-rpc.publicnode.com” to check every block added to Cosmos Hub from height 21936557 to height 22037357, a period which spans 7 days, we have found that there has always been 180 validators [17]. In fact, we have not observed that there has ever been less than 180 validators ever since the validator limit was increased to 180 in July of 2023.

As another example, consider the Nakamoto coefficient of Cosmos Hub. It is possible

for one of the top 7 validators to suddenly stake a large amount tokens that would skew the distribution of voting power. In practice, this rarely happens and voting power gradually increases or decreases over time. By using the rpc endpoint at “cosmos-rpc.publicnode.com”, we see that the top 7 validators control approximately 34.4587% of the stake at block height 22037357. The top 6 validators control approximately 31.2088% of the stake. Therefore, the Nakamoto coefficient is 7. By checking the previous 43200 blocks, which spans about 72 hours, we see that the top 7 validators control at least 34.3822% and at most 34.4954% of the stake. In this case, it is excessive to prove the Nakamoto coefficient for every block.

We propose that policy proofs should be cached and use for subsequent messages along the same channel. By doing so, we show that the amortized cost of generating a policy proof is small.

For the experiment, we wanted to determine the average cost of including a validator proof per hop. To do this, we sent 12 messages along a 2-hop channel that contained proofs for 4 validators. We compared the cost of doing so against the cost of sending the same messages along the same channel while omitting the proofs. We repeated this evaluation but for a 3-hop channel. Since the gas cost depends entirely on the amount of data being processed, the following equation can be derived directly from our implementation of policy proofs. Let C be the cost without policy proofs, C' be the cost with policy proofs, H be the number of hops, V be the number of validators and P be the cost of the a single validator policy proof.

$$C' = C + HVP \quad (5.1)$$

Number of Messages	Gas Used	Cost Relative to Single-Hop Token Transfer
20 messages	231885	71.1% ($\sim$ \$0.029 USD)
100 messages	3710.16	14.2% ($\sim$ \$0.0058 USD)
500 messages	742.032	2.84% ($\sim$ \$0.00116 USD)
2500 messages	148.41	0.569% ($\sim$ \$0.000232 USD)

Table 3: Expected additional cost per message for validator count policy.

Solving for C , we found that the average additional gas cost per validator per hop is 15459. Assuming a maximum network diameter of 3, as is the case currently for chains within the Cosmos ecosystem, a reasonable upper-bound on the additional cost of such a proof is $15459 * 3 * 100 = 4637700$ gas units.

IBC allows modules to customize how relayers are compensated (ics-029) [21]. Since relayers submit the policy proofs, they must pay the additional gas cost. It is conceivable

that a module would require users to pay an additional fee for every message. This fee can be used to compensate a relayer when it submits a policy proof in the future. It is useful to look at the amortized cost, because this gives a sense of how much this additional fee could be. The more often a channel is used, the less this additional fee would need to be to cover the cost of policy proofs. Table 3 shows calculations for the expected additional cost amortized over the number of messages sent.

5.6 Implication of Network Topology Gas Cost

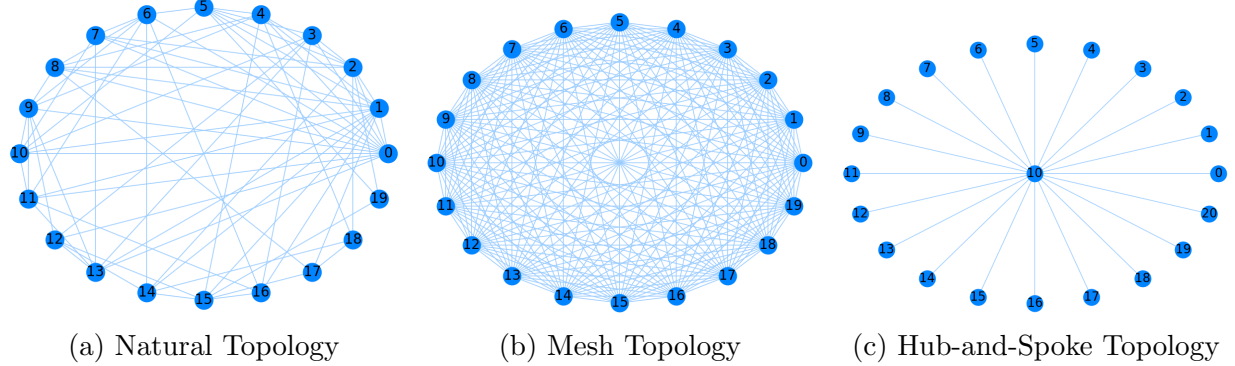


Figure 25: Blockchain network topologies.

Here, we evaluate the implications of network topology on the cost of blockchain interoperability. The three evaluated topologies are given in Figure 25. The natural topology is representative of how the topology of the Cosmos ecosystem arose over time. Baton routes messages across the natural topology. When checking Mintscan.io in June of 2024, only the two chains, Osmosis and Cosmos Hub, were connected to more than 50 chains each [22]. A small proportion of chains such as Juno, Stargaze and ARCHWAY are connected to about 30 other chains. Most chains have no more than 20 connections, with a larger proportion of chains having even fewer connections. In our custom topology, the lower the chain ID, the more connections that chain has. In this section, we demonstrate that, despite being more decentralized, Baton is no less efficient than the hub-and-spoke approach currently being used. We argue that Baton’s efficiency, in terms of the quantity and cost of transactions, is comparable to that of a mesh network where every blockchain is connected to every other blockchain.

For every pair of chains, a message is sent at a time chosen uniformly at random in the interval $[0\text{ms}, 12000\text{ms}]$. There must be at least an 8 second delay separating subsequent

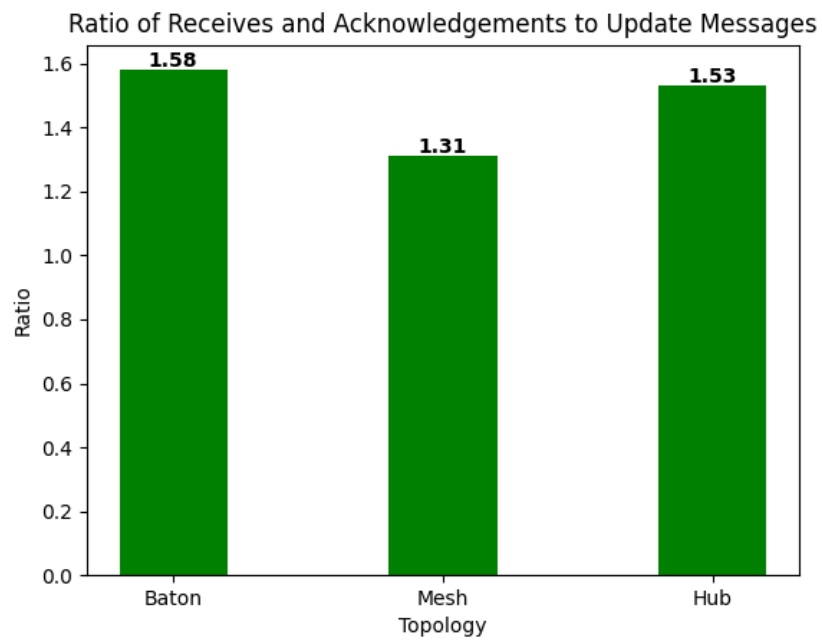


Figure 26: The average amount of receive and acknowledgment messages processed for every update client message. We send messages uniformly at random between pairs of chains. In each scenario, there were 40 relayers relaying packets.

sends between the same pair of chains. This 8 second delay helps avoid the error with Cosmos chains where the same account signs two transactions in the same block. Since channels may share some connection hops, messages sent on different channels can reuse client updates. This is beneficial, as it means relayers will not spend gas tokens performing unnecessary light client updates. We quantify these benefits by computing the ratio of the number of messages delivered to the number of client updates required. A larger ratio indicates better efficiency.

Using our representation of a naturally occurring topology, Figure 26 shows that Baton delivers 20.6% more messages given the same number of client updates than if the network were to be configured as a mesh. Baton delivers 3.26% more messages when compared to the hub-and-spoke topology.

Expected Hops and Gas Usage Per Message For Each Topology					
	1-Hop	2-Hops	3-Hops	Expected Hops	Expected Gas
Natural	144	220	16	1.663	102886
Mesh	380	0	0	1	84757
Hub-and-Spoke	21	420	0	1.952	111253

Table 4: Number of 1-hop, 2-hop and 3-hop channels for each topology. Also includes the expected number of hops and expected gas cost per message if messages are uniformly sent between chain pairs.

Table 4 shows the expected number of hops and the expected gas cost of sending messages between pairs of chains in each topology. The mesh topology gives the best results. Be that as it may, it is unreasonable for these results to be seen in reality. Surveys on interoperability solutions have demonstrated that differences in consensus protocols, cryptographic schemes, smart contract languages and other infrastructure details hinder widespread interoperability [49, 53]. We observe this empirically, as most blockchains connected to the Cosmos Hub use the CometBFT consensus protocol [22]. While blockchains can be updated to be able to interoperate with a heterogeneous chain, it can take months for a blockchain community to agree on and implement software updates [20]. It is therefore more feasible to update blockchains to support Baton, rather than to support every other blockchain’s type of light client. Since Baton requires fewer total transactions, we have observed that the total gas cost is within 10.5% of the ideal mesh case.

Chapter 6

Conclusion

As the number of blockchains has increased, there has likewise been an increase in the importance of blockchain interoperability. Many individualized interoperability protocols have emerged, since blockchains often operate in a different way than one another. IBC abstracts away many of these nuances, such that it can be implemented for most blockchains. Even then, differences in consensus verification algorithms and cryptographic schemes prevent some IBC enabled blockchains from directly connecting with each other. They tend to either directly connect to a handful of other chains, or rely on a central hub for network wide cross-chain communication.

We present Baton, an interoperability protocol for efficient and secure cross-chain message routing and delivery. Baton is built on IBC. By employing multi-hop IBC channels, Baton allows blockchains to communicate with other blockchains to whom they are indirectly connected. This reduces the need for trusted connection setup periods that are vulnerable to exploitation. Furthermore, Baton eliminates the need for hubs that can become a central bottleneck and security risk for cross-chain messaging.

Baton facilitates efficient and near-optimal route discovery. Standard IBC requires users to consult external sources to determine which channel leads to which blockchain. This is tedious and error prone. Instead, Baton enabled blockchains store a mapping of chain ID to channels. This allows users to send cross-chain messages without consulting external sources, simply by specifying the destination chain. Routes are generated off-chain in order to save gas, and verified on-chain. We demonstrate that, given certain cryptoeconomic incentives, rational users will map blockchains along a channel to their correct IDs. Furthermore, since protocol participants are paid whenever others use their channels, participants are incentivized to create channels with near-optimal routes.

Baton also allows users to dynamically enforce security requirements for cross-chain messaging. This is accomplished by augmenting messages with proofs of security properties, such as the number of validators and the distribution of stake, of the chains along a channel. As a result, the security of cross-chain messaging is dependent on the security of the chains involved, rather than the security of a central hub or a trusted third party as is the case with other approaches.

Finally, Baton can detect fabricated cross-chain messages. It is notoriously expensive to detect incorrect state transitions for blockchains, as this usually requires running a full node for that chain. Compromised blockchains that perform incorrect state transitions can fabricate cross-chain messages that were not sent by the source chain. Baton provides a mechanism to detect such malicious behaviour by allowing messages to travel along multiple disjoint paths.

We demonstrate that Baton is more decentralized than standard IBC by showing that it can maintain better connectivity despite when popular chains become unavailable. Baton also introduces far less congestion, as cross-chain messages are not committed to intermediate blockchains along a route. While standard IBC only uses single-hop channels, we show that the gas used and the time taken for sending cross-chain messages decreases linearly in proportion to the length of the path when switching from single-hop channels to multi-hop channels. This establishes a case for why it is more efficient to use multi-hop channels, as Baton does, rather than single-hop channels. We show that the additional gas cost for including relevant security policy proofs, such as a proof of the number of validators being ≥ 100 , is small. The policy proof costs $\sim \$ 0.029$ USD when amortized over 20 messages, and $\sim \$ 0.000232$ USD when amortized over 2500 messages. Baton differs from other approaches, as it can route and deliver messages across the natural topology of the network, instead of relying on direct connectivity or hub chains. Since light client updates are cached, 20.6% and 3.26% more messages can be delivered per client update when using a more natural topology rather than a mesh topology or hub-and-spoke topology respectively. Consequently, Baton is no less efficient than other approaches, despite being more decentralized and secure.

References

- [1] 7 Cross-Chain Bridge Vulnerabilities Explained — Chainlink. <https://chain.link/education-hub/cross-chain-bridge-vulnerabilities>, September 2024.
- [2] Behnaz Abdorrahimi, Atefeh Nekouie, Amir Masoud Rahmani, Jan Lansky, Vladimír Nulíček, Mehdi Hosseinzadeh, and Mohammad Hossein Moattar. Blockchain technology and raft consensus for secure physician prescriptions and improved diagnoses in electronic healthcare systems. *Scientific Reports*, 2024.
- [3] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. <https://app.uniswap.org/whitepaper-v3.pdf>, March 2021.
- [4] Guillermo Angeris, Theo Diamandis, and Ciamac Moallemi. Multidimensional blockchain fees are (essentially) optimal, 2024.
- [5] Axelar. Axelar: Connecting applications with blockchain ecosystems. <https://www.axelar.network/whitepaper> [Accessed 22/08/2024].
- [6] <https://www.mintscan.io/axelar/> [Accessed 31/07/2024].
- [7] Rob Behnke. Explained: The orbit bridge hack (december 2023). <https://www.halborn.com/blog/post/explained-the-orbit-bridge-hack-december-2023> [Accessed 01/11/2024].
- [8] Rainbow Bridge. Rainbow bridge. <https://rainbowbridge.app/transfer> [Accessed 22/08/2024].
- [9] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.
- [10] Vitalik Buterin. Ethereum whitepaper. <https://ethereum.org/en/whitepaper/> [Accessed 22/08/2024].

- [11] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.
- [12] Vitalik Buterin, Diego Hernandez, Thor Kamphofner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X. Zhang. Combining GHOST and casper. *CoRR*, abs/2003.03052, 2020.
- [13] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*, New Orleans, LA, February 1999. USENIX Association.
- [14] 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].
- [15] 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.
- [16] CoinGecko. Cryptocurrency prices by market cap. <https://www.coingecko.com/> [Accessed 13/09/2024].
- [17] Cosmos. Chain registry. <https://github.com/cosmos/chain-registry> [Accessed 13/08/2024].
- [18] Cosmos. Cosmos. <https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md> [Accessed 22/08/2024].
- [19] Cosmos. Cosmos hub. <https://github.com/cosmos/gaia> [Accessed 05/08/2024].
- [20] Cosmos. Cosmos hub forum. <https://forum.cosmos.network/c/hub-proposals/25> [Accessed 04/09/2024].
- [21] Cosmos. Interchain standards (ics) for the cosmos network interchain ecosystem. <https://github.com/cosmos/ibc> [Accessed 05/08/2024].
- [22] <https://www.mintscan.io/cosmos/> [Accessed 31/07/2024].
- [23] DefiLlama. Defillama. <https://defillama.com/> [Accessed 13/09/2024].

- [24] Xiaohong Deng, Kangting Li, Zhiqiang Wang, Juan Li, and Zhiqiong Luo. A survey of blockchain consensus algorithms. In *2022 International Conference on Blockchain Technology and Information Security (ICBCTIS)*, pages 188–192, 2022.
- [25] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
- [26] Shimon Even and Yacov Yacobi. Relations among public key signature systems. Technical report, Computer Science Department, Technion, 1980.
- [27] Michael Fischer, Nancy Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, 04 1985.
- [28] Benoît Garbinato and Ian Rikebusch. Impossibility results on fair exchange. In *10th International Conference on Innovative Internet Community Systems (I2CS) – Jubilee Edition 2010 –*, pages 507–518. Gesellschaft für Informatik e.V., Bonn, 2010.
- [29] Christopher Goes. Relay algorithms. <https://github.com/cosmos/ibc/tree/main/spec/relayer/ics-018-relayer-algorithms> [Accessed 16/09/2024].
- [30] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The L^AT_EX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.
- [31] Sergey Gorbunov. The multichain is now timeless: Introducing future-proof interoperability. <https://www.axelar.network/blog/future-proof-interop-path-adaptability-for-cross-chain-dapps> [Accessed 22/08/2024].
- [32] Dominic Grandjean, Lioba Heimbach, and Roger Wattenhofer. Ethereum proof-of-stake consensus layer: Participation and decentralization, 2023.
- [33] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable and decentralized trust infrastructure, 2019.
- [34] Abhishek Guru, Bhabendu Kumar Mohanta, Hitesh Mohapatra, Fadi Al-Turjman, Chadi Altrjman, and Arvind Yadav. A survey on consensus protocols and attacks on blockchain technology. *Applied Sciences*, 13(4), 2023.
- [35] Charles Hoskinson. Why we are building cardano: A subjective approach. <https://whitepaper.io/document/581/cardano-whitepaper> [Accessed 20/08/2024].
- [36] Hyperlane. Hyperlane docs. <https://docs.hyperlane.xyz> [Accessed 22/08/2024].

- [37] The Interoperability Trilemma: AKA Why Bridging Ethereum Domains is So Damn Difficult. Connex arjun bhuptani. <https://medium.com/connex/the-interoperability-trilemma-657c2cf69f17> [Accessed 03/08/2024].
- [38] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Paper 2016/889, 2016.
- [39] LayerZero Labs. Layerzero. <https://layerzero.network/publications/LayerZeroWhitepaperV2.1.0.pdf>
- [40] Polymer Labs. Polymer: Ethereum’s interoperability hub. <https://www.polymerlabs.org/polymer-whitepaper-pdf> [Accessed 22/08/2024].
- [41] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column)* 32, 4 (Whole Number 121, December 2001), pages 51–58, December 2001.
- [42] Ori Mazor and Ori Rottenstreich. An empirical study of cross-chain arbitrage in decentralized exchanges. 01 2024.
- [43] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf> [Accessed 03/08/2025].
- [44] Octopus Network. Octopus network: Where web3.0 happens. <https://github.com/octopus-network/papers/blob/main/Octopus>
- [45] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, June 2014. USENIX Association.
- [46] Optimism. Optimism collective. <https://www.optimism.io/> [Accessed 31/10/2024].
- [47] Polkadot. Polkadot’s consensus protocols. <https://wiki.polkadot.network/docs/learn-consensus> [Accessed 20/08/2024].
- [48] NEAR Protocol. The near white paper. <https://pages.near.org/papers/the-official-near-white-paper> [Accessed 22/08/2024].
- [49] Kunpeng Ren, Nhut-Minh Ho, Dumitrel Loghin, Thanh-Toan Nguyen, Beng Chin Ooi, Quang-Trung Ta, and Feida Zhu. Interoperability in blockchain: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12750–12769, 2023.

- [50] Kevin Sekniqi, Daniel Laine, Stephen Buttolph, and Emin Gün Sirer. Avalanche platform. https://cdn.prod.website-files.com/5d80307810123f5ffbb34d6e/6008d7bbf8b10d1eb01e7e16_Avalanche
- [51] Balaji S. Srinivasan. Quantifying decentralization. <https://news.earn.com/quantifying-decentralization-e39db233c28e> [Accessed 24/08/2024].
- [52] Chainalysis Team. Multichain protocol experiences mysterious withdrawals, suggesting multi-million dollar hack or rug pull [updated 7/19/23]. <https://www.chainalysis.com/blog/multichain-exploit-july-2023/> [Accessed 01/11/2024].
- [53] Gang Wang, Qin Wang, and Shiping Chen. Exploring blockchains interoperability: A systematic survey. *ACM Comput. Surv.*, 55(13s), jul 2023.
- [54] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. <https://assets.polkadot.network/Polkadot-whitepaper.pdf> [Accessed 22/08/2024].
- [55] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0.8.13. <https://solana.com/solana-whitepaper.pdf> [Accessed 03/08/2024].
- [56] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain, 2019.
- [57] Gengrui Zhang, Fei Pan, Yunhao Mao, Sofia Tijanic, Michael Dang’ana, Shashank Motepalli, Shiquan Zhang, and Hans-Arno Jacobsen. Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms, 2023.