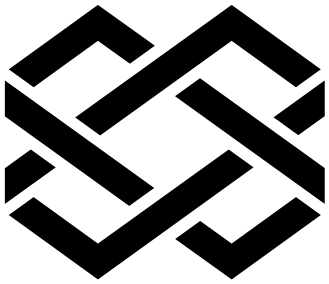
Polymer: Ethereum's Interoperability Hub



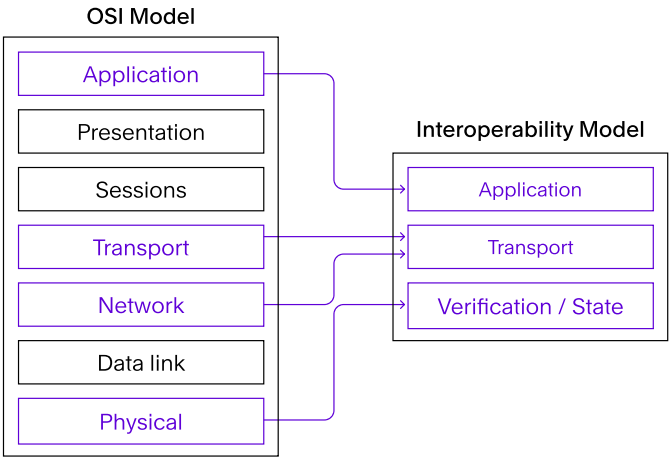
Abstract

Polymer addresses the fragmentation problem caused by the growth of disparate Ethereum rollup ecosystems. Currently, on Ethereum, interoperability (interop) protocols exist solely as smart contracts/applications built atop the EVM. This leads to varying implementations among different interop protocols and across rollup ecosystems, which exacerbates fragmentation, leads to a lack of standards and the adoption of immature specifications as well as posing challenges to scalability. Polymer aims to create a unified interoperability story on Ethereum that extends to the Cosmos and beyond. Built as an Ethereum Layer 2, the Polymer hub combines the best of Cosmos and Ethereum technology to serve as Ethereum's interoperability hub. This is achieved by extending the Inter-Blockchain Communication (IBC) protocol to all chains in the Polymer ecosystem, by decoupling IBC's transport-level execution from the application-level execution occurring on connected rollups through a protocol called virtual IBC. In doing so, rollups are provided with a standardized communication layer, enabling seamless composability between applications on different chains. This enables the permissionless expansion of the IBC network and the creation of an open and vibrant interoperability economy which allows all types of participants to contribute.

1. Current Interoperability Landscape

Introducing the Interoperability Model

The interoperability model consists of three layers — application, transport and verification or state. The application layer consists of application protocols such as token transfers. The transport layer consists of all of the core messaging logic which includes basic features such as acknowledgements and timeouts as well as advanced features such as authentication and upgrades. The verification or state layer handles verifying the state transitions of counterparty chains. We depict the interoperability model below and draw parallels between it and the OSI model.[1]



1.1. VM-based Interoperability Model

VM-based interoperability is currently the most common interoperability model. This includes rollup ecosystem-specific and third party interoperability protocols fully implemented in the form of smart contracts that live within a VM such as the EVM. This means that all three layers of interoperability model are implemented within the EVM.

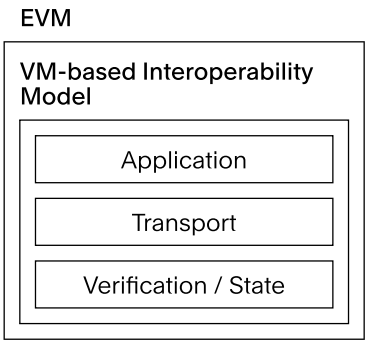
Without discussing the actual specifications of the interoperability protocol, there are a number of reasons why VM-based interoperability is a broken model that doesn't scale:

1.1.1. **Standardization is not possible at the smart contract or VM level.** Every VM is coupled with a set of internal data structures, RPCs and transaction formats specific to each chain. Many VMs also have different language front ends and internal mechanics.

1.1.2. **Maintenance costs grow linearly with chain type diversity.** Every additional chain type creates increasing overhead for the system as a whole. Without modifications to the chain itself, relayers are always required which introduces the overhead of maintaining and implementing chain specific relay logic on a per chain type basis.

1.1.3. **Inefficient execution.** Many VM-based interoperability protocols sacrifice on functionality and compatibility with other interoperability networks for VM level efficiency. Restricted innovation. Interoperability protocols are limited in what they can accomplish by purely operating with a VM. We'll explore the possibilities here and what the future of interoperability could look like in section 7.2. VM-based Interoperability Model

1.1.4. **VM-based interoperability is currently the most common interoperability model.** This includes rollup ecosystem-specific and third party interoperability protocols fully implemented in the form of smart contracts that live within a VM such as the EVM. This means that all three layers of interoperability model are implemented within the EVM.

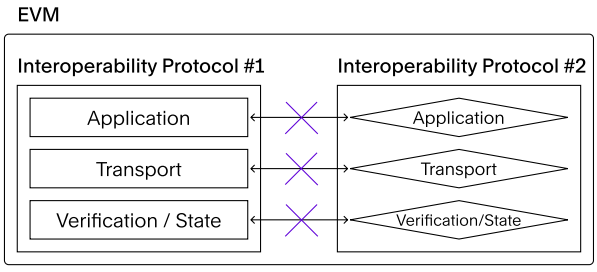


1.2. Diverging Implementations

Interoperability across Ethereum rollups is highly fragmented with a mixture of rollup — specific and third party protocols. A message sent over one protocol cannot be directly handled by another — the way messages are committed and understood are incompatible even among different protocols requiring additional translation layers.

An interoperability protocol defines the API and mechanics used to access the decentralized web either from on-chain to on-chain or in some cases from off-chain to on-chain. At the current rate of divergence, we're going to end up with hundreds of different APIs to access the web all with different underlying mechanics and properties.

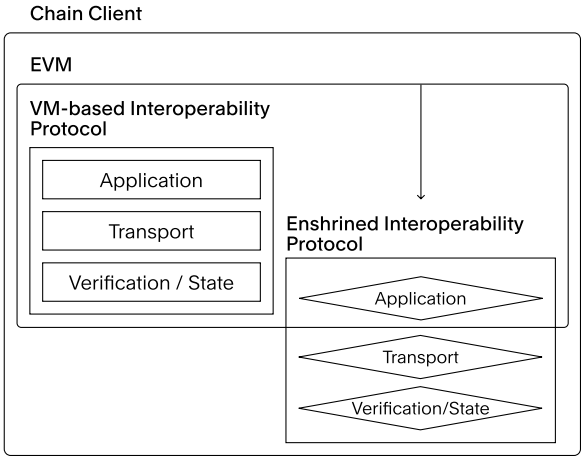
The solution to this neverending divergence is an enshrined interoperability standard such as IBC, which fixes all of the problems created by the VM-based approach commonly taken today. Instead of dealing with hundreds of different APIs, IBC creates a single unified API to access the decentralized web in a way that's analogous to TCP/IP.2



2. Building the Future of Interoperability

2.1. The Vision

We envision an end state for interoperability with IBC as the enshrined or native interoperability standard in every chain and chain development framework. This allows the transport layer of interoperability as a whole to standardize on the same canonical implementation across all types of chains which is impossible to accomplish purely at the VM layer. We expect to see purely VM-based solutions to phase out over time due to their increasing maintenance burden and proliferation of IBC as the enshrined standard.

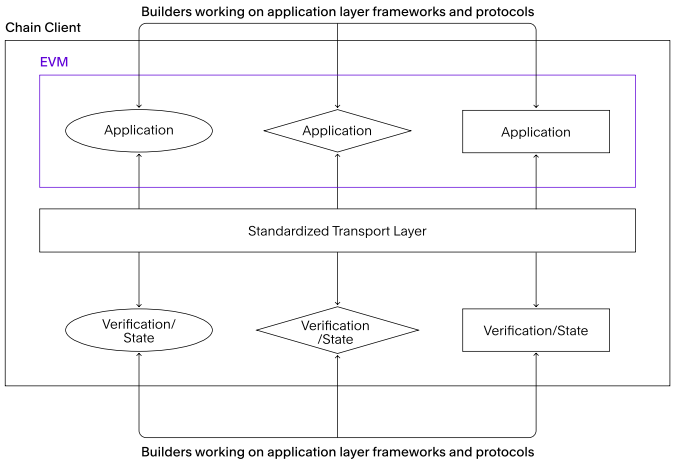


In a world where IBC is enshrined on every chain as the interoperability standard, builders will either migrate up or down the stack. Instead of every interoperability protocol recreating the entire interoperability stack from top to bottom, builders can focus on either the application or verification layers of the stack. It's possible that existing interoperability solutions will either become application side frameworks and developer platforms, shift focus to verification or both.

We also expect to see the rise of domain specific interoperability hubs with Polymer being the first in this category. A domain is defined as a unique combination of infrastructure layers that provide functions such as settlement and data availability (DA) to the rollups being built on top. Each unique combination of infrastructure layers has its own distinct trust model.

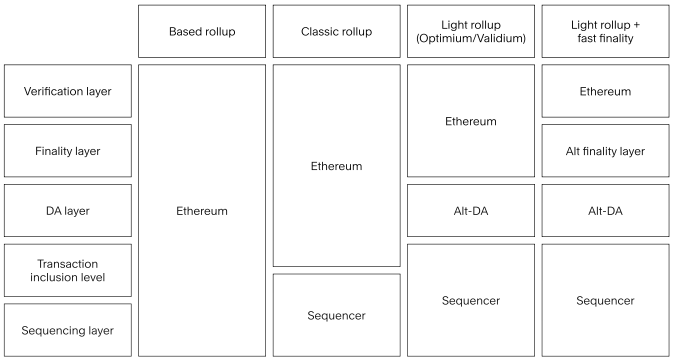
For example, L2Beat categorizes rollups[3] that use off-chain DA as optimiums and validiums for optimistic and ZK rollups respectively. Off-chain DA appears as a data availability committee[4] DAC) to Ethereum where the security of the DAC is non-fungible across alt-DA solutions. Rollups may also opt to adopt a separate finality layer or inherit reorg resistance[5] from outside Ethereum which further complicates the trust model (there isn't a new term for these types of rollups yet).

There are five properties that determine the security of a rollup based on Sreeram's model for determining the security[6] of a chain (or rollup). Ledger growth, censorship resistance[7], DA, reorg resistance and validity. These properties map to the following functional layers of a rollup in the same order - sequencing, transaction inclusion, DA, finality and verification. The first two properties represent liveness while the latter three properties represent safety. All five properties or liveness plus safety equals the security of the rollup. The diagram below highlights the major categories of rollups today and what source they inherit the five security properties from which determines the trust model of the rollup.



The safety properties of data availability, reorg resistance and validity are critical when it comes to interoperability. The layers above providing data availability and validity guarantees affect the correctness of the state transition while the layer providing reorg resistance directly affects the minimum latency of cross rollup transactions. For example, rollups that share Ethereum as a finality layer can have sub finality cross chain transactions between one another as they will all reorg together alongside Ethereum.

Since each of the distinct trust models or domains shown above are not fungible with each other in terms of security and interoperability UX, an instance of Polymer will be deployed to every domain forming a network of hubs. Each instance of Polymer inherits the trust model of the domain and optimizes IBC connectivity for its rollup neighbors. Connected instances of Polymer will then optimize cross domain IBC connectivity creating a cross domain IBC mesh.



2.2. Key Design Goals

There are a few of key design goals behind Polymer's approach and architecture:

2.2.1. Build with the end state in mind. Designing an interoperability solution with only the current state of the world in mind is not likely to succeed long term. A world where an enshrined interoperability standard like IBC is ubiquitous pushes existing interoperability protocols into focusing on other layers of the stack.

2.2.2. Creating an interoperability economy. Currently, it's challenging for both developers and infrastructure operators to navigate how to contribute to the burgeoning interoperability economy. Polymer aims to establish clear economic rails for both participants.

2.2.3. Permissionless IBC network expansion. IBC network expansion is currently limited to the rate of native IBC integrations which is both resource and time consuming. Polymer's virtual IBC protocol enables permissionless expansion to new chains and VMs without requiring a native integration.

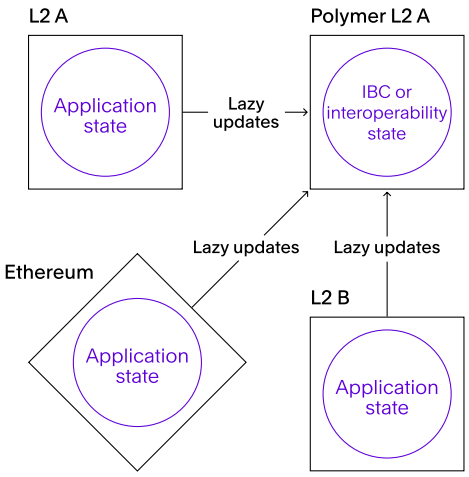
2.2.4. Direct connectivity between Ethereum <> IBC network. Every existing interoperability solution between Ethereum and the broader IBC network exists as a middle chain between the two. Polymer as an L2 means that any IBC enabled chain Cosmos or otherwise can establish direct connectivity with Ethereum and its rollups. No middle chain involved.

3. Polymer Overview

3.1. Polymer is a Port City

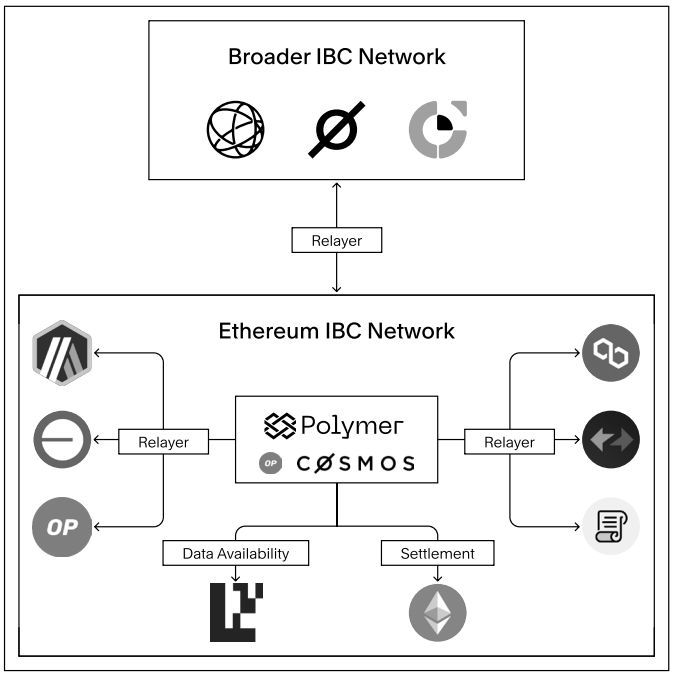
Polymer follows the port city vision of the Cosmos hub[8] but within Ethereum. Polymer establishes trade routes both within Ethereum as well as with external ecosystems such as the Cosmos. Previously, trade routes between Ethereum and the Cosmos have always required a middle man or translation layer in between as Ethereum does not speak IBC. While Cosmos chains could directly verify Ethereum consensus, there is no IBC compatible state or logic that lived on Ethereum.

In the IBC network, every chain is represented in the network topology as an IBC commitment or a state commitment to IBC key/value pairs. Polymer handles IBC state and commitments separately from L2 chain state and commitments, this way the interaction between chain state and IBC state becomes asynchronous. In a way, Polymer produces lazily merkleized interoperability or cross chain state on behalf of connected L2s and Ethereum. Polymer then settles back to Ethereum by posting the IBC state of all L2s and Ethereum itself — thus all states live on Ethereum and are secured by it



This means that the Polymer Hub (L2) on Ethereum acts as an IBC translation layer. Polymer translates all connected L2 and Ethereum state transitions into a fully IBC compatible format that can be understood by any chain with a native IBC implementation. In doing so, any IBC enabled chain can then verify Ethereum consensus and then by unwrapping Polymer's state stored on Ethereum, establish direct IBC connectivity between the Cosmos and Ethereum or a direct trade route between the two ecosystems.

Polymer's network isn't limited to single port city. With the proliferation of L3s on Ethereum as well as other ecosystems, multiple instances of Polymer will form an interconnected mesh of port cities facilitiating both local and foreign trade.



3.2. Credible Neutrality

Credible neutrality is table stakes for an interoperability protocol to be able to achieve industry wide adoption across ecosystems. Many interoperability protocols are siloed to a single ecosystem, framework or company. For a protocol to be credibly neutral, it must satisfy a number of requirements:

3.2.1. **A large contributor base that includes multiple companies and individual collaborators.** IBC currently has 17+ different companies and 120+ individual contributors across its main code repositories working on the protocol.

3.2.2. **Natively adopted across many chains or chain development frameworks (CDFs).** IBC is natively implemented within a growing number of CDFs such as the Cosmos SDK[9] and Sovereign SDK[10] with integrations coming up in Rollkit[11] and the OP Stack.[12] There is a growing number of non-cosmos chains that implement the standard such as Penumbra[13] and Anoma's Namada.[14]

3.2.3. **No token or value capture at the protocol level.** Any entity, chain or CDF can implement IBC and connect to the IBC network with zero opportunity for third party value extraction. Also, IBC itself has no token unlike many other interoperability protocols.

Additionally, many of the companies building interoperability protocols are also building token bridges, DeFi or other applications on top. This would be analogous to the Ethereum foundation also building Uniswap. The foundation or company behind critical infrastructure layers must exercise credible neutrality to ensure the neutrality of the interoperability protocol itself. For this reason, Polymer is not engaged in the building of DeFi protocols or token bridges. These primitives may be built on top of Polymer and IBC by anyone.

3.3. Primer On IBC

The inter-blockchain communication protocol (IBC) is an end-to-end, connection-oriented, stateful protocol for reliable, ordered, and authenticated communication between modules on separate distributed ledgers. The IBC specification defines how each of the following key abstractions is committed to within the state of the chain:

3.3.1. **Clients.** Every client defines arbitrary verification logic that can range from a single private key to a proof of consensus or valid execution. A client is a template of logic that can be instantiated with some security parameters.

3.3.2. **Connections.** A pair of clients form an IBC connection. This represents connectivity between two chains defined over some security or trust model. This is analogous to a highway between two chains.

3.3.3. **Channels.** An application specific communication path over any number of IBC connections. This is analogous to every application getting their own lane on a highway or connection.

3.3.4. IBC is the only interoperability standard that actually works like TCP/IP. Channels allow direct logical connectivity over an infinitely extensive physical topology. Multi-hop channel connectivity (ICS-33)[15] allows channels to span over any number of connections. This is currently unique to IBC and is a Polymer contribution to the spec.

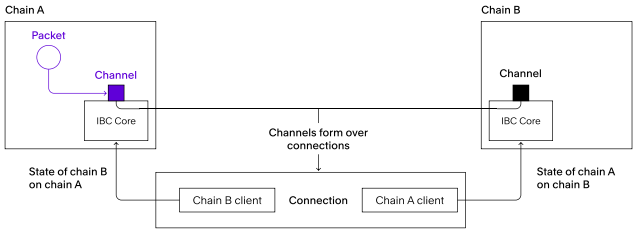
3.3.5. **Packets.** Arbitrary application data that flows over a specific IBC channel.

3.3.6. **Middleware.** Custom logic that can be instantiated on a per application basis. This is useful in cases where there is share-able logic across applications that may not apply to all applications.

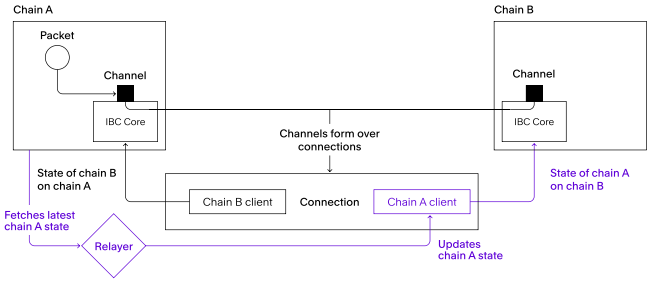
What sets IBC apart from VM-based messaging solutions is that in IBC, blockchains do not directly pass messages to each other over the network. To communicate, blockchains communicate state which encodes a commitment to a precisely defined path reserved for a specific message type and a specific counterparty chain. Relayers monitor for updates on these paths and relay messages by submitting the data stored under the path along with proof of that data to the counterparty chain.

Let's walk through a simplified packet send step by step to better understand how each of the previously defined primitives work in the flow of the protocol below.

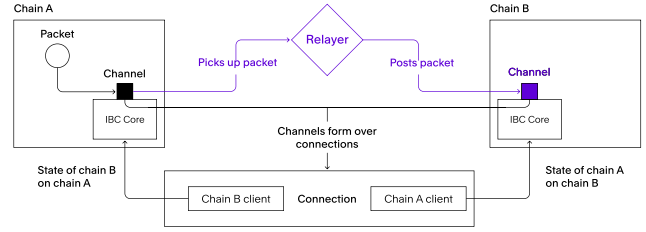
[1] An application on the source chain creates a packet with the data to be transmitted and passes it into a specific channel to be committed to the source chain's state and history.



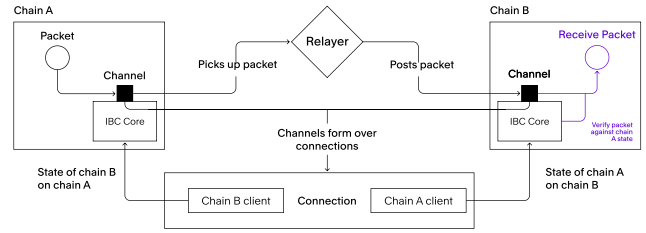
[2] A relayer is an off-chain actor that observes these commitments and relays a client update to ensure chain B has an up to date view of the state of chain A. Clients provide the latest state of the counterparty chain (i.e. the commitment proof that the transaction actually took place) and many packets can be batched into a single client update.



[3] A relayer then relays the packet to the other end of the channel on the destination chain.



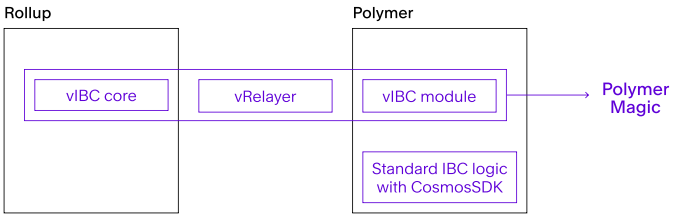
[4] Upon reaching the destination, the inbound packet from the relayer is to be checked against the counterparty chain's state (chain A) to ensure that it was sent over a specific channel and connection(s).



3.4. Virtual IBC: Modular IBC Execution

The motivating idea behind virtual IBC (vIBC) was to make the IBC interoperability stack more modular and permissionless to integrate unlocking fast IBC network expansion. There's three layers to the IBC protocol. The application,[16] transport (or core)[17] and the verification (or client)[18] layers each with its own corresponding specification. Virtual IBC decouples the transport layer of IBC from the connected chain and outsources it to Polymer. The application and verification layers of IBC still live on the virtual chain. The following diagram depicts how Polymer enables IBC on rollups using virtual IBC.

Making a rollup IBC enabled



It allows connected rollups to outsource the overhead of IBC execution/compute to Polymer. A chain that has outsourced its IBC execution to Polymer is called a virtual chain. Virtually connected chains appear as an additional hop in the IBC network which is an accurate mapping of network topology. This also means that if a connected chain wants to natively implement IBC in the future, it can transparently upgrade existing channels to use a native connection when available. Details of each high level vIBC component are documented below.

3.4.1. Smart contract API interface (vIBC core contract) on connected chains. This is a smart contract implementation of the IBC handler[19] interface that exposes IBC channel and packet life cycles to connected applications. It tracks IBC channel and packet information and acts as an IBC post office for the connected chain handling both inbound and outbound mail. The vIBC core contract is bound asynchronously to the actual IBC core or transport implementation that lives on Polymer.

- **Polymer to virtual chain state verification.** The virtual chain or rollup supports modular verification of Polymer execution. This is achieved through encapsulating different verification mechanisms inside of unique IBC clients that live on the virtual chain representing Polymer's latest state.
- **Polymer to virtual chain message verification.** IBC messages such as inbound packet and channel handshakes are verified against the latest state of Polymer verified inside of an IBC client above.

3.4.2. vIBC Relayer. Establishes async communication the smart contract API interface deployed on a connected chain and the virtual IBC module implementation on Polymer. It is a specialized relayer that understands the virtual IBC protocol and picks up vIBC events generated from the vIBC core contract and the proofs required in both directions.

3.4.3. vIBC module on Polymer hub. This component acts as the translator between the vIBC events happening on the connected chain and how they need to be interpreted for the standard IBC module to understand.

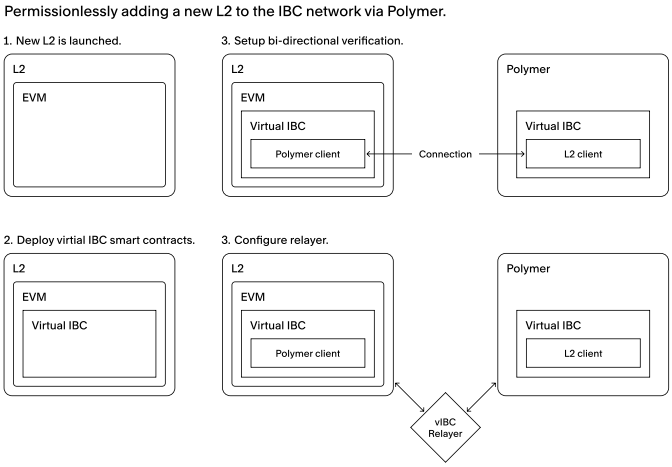
- **Virtual chain to Polymer state verification.** Polymer also supports modular verification of the state transitions of virtual chains or rollups via IBC clients. A bidirectional highway or IBC connection is formed when we pair a client on Polymer with a client on the virtual chain.
- **Virtual chain to Polymer IBC event verification.** Polymer verifies IBC events generated from the vIBC core contract against a valid virtual chain state. The latest state of thevirtual chain lives inside of an IBC client on Polymer. In many chains, events are emitted in the forms of logs encoded in a receipt of execution which can be proved against the state transitions or headers of the chain.

3.5. Permissionless IBC Expansion

IBC expansion has historically been limited by the speed at which teams can integrate ibc-go[20] or ibc-rs[21] natively

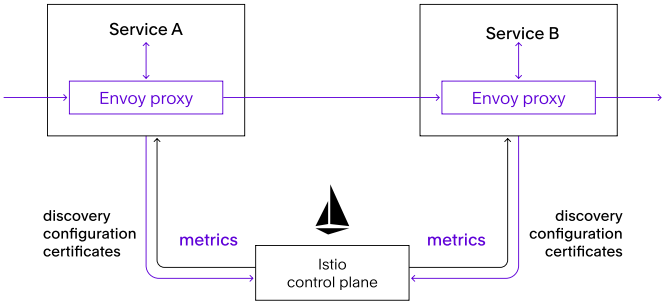
into their chain or CDF. Polymer breaks this limitation with a novel design and protocol which we'll cover in the next section. Permissionless IBC connectivity via Polymer is critical to the growth of the IBC network and expansion to new chains and ecosystems.

For every new or existing rollup, there will be multiple security options available ranging from self verification to relying on the native bridge of the rollup for verification. Relayers in the Polymer network can be run by anyone. A rollup can be permissionlessly added to the IBC network via Polymer by deploying a set of smart contracts, performing some IBC specific setup and configuring a new or existing relayer to relay from the new chain. This setup process can be performed without any permission required from the team or 3rd party.

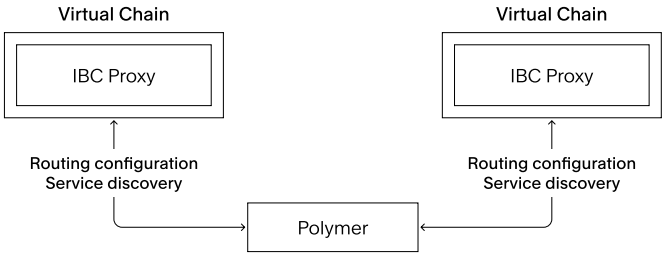


3.6. The IBC Mesh (Cloud Computing Comparison)

To motivate Polymer's long term role in the IBC network, we'll draw a comparison to a well known design pattern in cloud computing. Most modern applications are designed as a collection of microservices in the cloud. A service mesh is a networking layer that you can add to your application that handles various functions such as service discovery, traffic management and security. Istio[22] is a leading service mesh solution and graduated Cloud Native Computing Foundation project shown in the diagram below.



Rollups are analogous to microservices, interoperability is analogous to networking and Polymer is analogous to Istio. Polymer's design was motivated by how the service mesh pattern solves networking at scale in the cloud. Instead of Envoy[23] proxies deployed to services in the Istio diagram above, Polymer deploys IBC proxies to connected virtual chains. Similar to Istio, Polymer also handles service discovery, routing configuration and security.



While a single Istio node can fulfill the requirements of a small deployment, it's common for multiple Istio nodes to be deployed at scale. In a similar fashion, the IBC mesh doesn't end with just a single instance of Polymer. While the initial deployment of Polymer is focused on enabling IBC across L2s on Ethereum, there will be a network of Polymer deployments to L3s as well as other ecosystems or trust domains as well in the future. Each deployment of Polymer facilitates communication between rollups within a domain while instances of Polymer will be connected across domains. This is analogous to intra data center and cross data center communication in the cloud. We'll cover this topic in more detail in section 8.1 of the appendix. Next, let's discuss how the features of Polymer and IBC enable developers to build highly composable cross chain applications.

4. Building Composable Applications

4.1. Functionality

There's an inherent tradeoff when designing lower level protocols between implementing too much or too little. If the protocol implements too many features, not every feature is relevant across all the applications building on top. When a protocol implements too few features, it forces applications to build the same features over and over again.

IBC strikes a good balance between a solid core feature set and opt-in middleware features. IBC was modeled after TCP/IP and includes core features such as:

4.1.1. Acknowledgements:

- Definition: A standard way for a calling contract on

one chain to receive a response from another.
- Why: Most applications operate on an asynchronous cause and effect basiswhere the effect on the source chain is not fully applied or finalized until a successful acknowledgement is returned. Having applications implement acknowledgements becomes complicated when dealing with other features such as ordering guarantees. Each application will need to track packet metadata for enforcement.

4.1.2. Timeouts:
- Definition: A packet can be safely reverted on the source chain if it is not received on the destination chain within some time window.
- Why: Many DeFi applications are time sensitive as the price of underlying assets can change dramatically over time. A timeout can guarantee that a cross chain transaction is executed within the specified time window or be reverted on the source chain. Fluctuations in gas fees across chains and other reasons can cause a packet to not be delivered on time. This is a dangerous feature for an application to implement as you must guarantee no double spends. A timed out packet cannot be executed or an executed packet cannot be timed out.

4.1.3. Authentication:
- Definition: Applications can perform a one time handshake to explicitly authenticate which contracts are allowed to communicate with each other. This is done using channels.
- Why: In most interoperability protocols, connected smart contracts are analogous to web applications bound to a port 80 where they are open to receive packets from any sender. For some applications such as a burn and mint token transfer protocol, a connected token contract may want to restrict minting rights to specific counterparty token contracts deployed on other chains.

4.1.4. Security Model Abstractions:
- Definition: Channels, connections and clients abstract away the underlying security model used for verification of cross chain state.
- Why: In a world where there's modular and configurable security, there needs to be abstractions that can represent the different security options. Otherwise an application would not be able to specify the security model they want to use to send to a chain.

4.1.5. Chain Addressing Format:
- Definition: Channels are used as a chain addressing format specifying exactly how two chains are connected. This is a way to identify a connected chain from the perspective of the source chain.
- Why: A human readable chain ID such as "Optimism" in the context of interoperability is a meaningless addressing format. The actual mapping of this chain ID to a chain requires social consensus around off chain knowledge of the intended destination. A channel ID is a meaningful addressing format as it explicitly specifies the individual connection IDs that link two chains together. Each connection ID represents a verification mechanism or security model mentioned above which is implemented as a pair of clients. Every connection specifies a unique view of the connected chain from the perspective of the source chain.

4.2. Protocol Extensibility

IBC is highly extensible due to it supporting opt-in middleware. Middleware is useful since some features are relevant to a subset of applications but not the entire set making it a poor core feature candidate. With Polymer, middleware can be added to either the VM side or on Polymer. We've produced a design pattern for IBC middleware smart contracts that can be layered on top of Polymers on-chain contracts.

A growing list of opt-in middleware includes:
- **Universal channels.** A universal channel is analogous to the carpool lane of a highway borrowing from the highway analogy we used earlier in the paper when discussing connections and channels.
- **Relayer fee incentivization.** Without incentives, the highways in the IBC network fall into disrepair slowing and ultimately stalling traffic which affects the liveness of connections and channels.
- **Interchain accounts (controller side).** Command and control of cross chain accounts can be established over the IBC network. This can be useful in the case of an app or app chain having core logic in one place while establishing outposts on other chains. Outposts expose a multichain API surface with a central control center.

Universal channels is an example of a Polymer innovation to extend the IBC core feature set and improve application user experience (UX). Some applications may want to avoid performing a channel handshake and instead assume responsibility for authentication, version control and upgrades. Universal channels multiplex pre-established channels between chains allowing an application to send messages between chains without establishing a channel themselves.

4.3. Upgradeability

Applications evolve over time and need a way to manage upgrades and version control especially as the number of chains an application is deployed upon grows. Channels offer a way for applications to perform atomic multi-chain upgrades and version control.

During channel handshakes, an application or middleware performs version negotiation and checks to ensure that all connected parties are all in agreement. Both application and middleware protocols are not static and change over time. To support protocols evolving over time, atomic upgrades can be performed by channels to either use new configuration parameters or migrate to a new version.

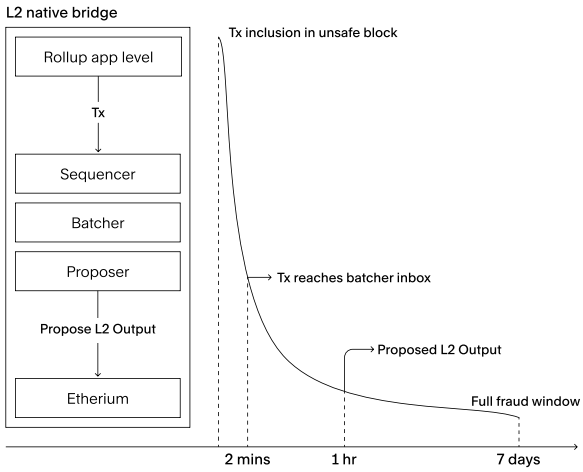
There are a number of edge cases (e.g. out of sync sequences, crossing hellos, packet flushing) that need proper handling to support atomic cross chain upgrades as

multi-chain upgrades cannot be applied in a fully atomic fashion. This would be highly non-trivial for an application to produce a specification on how to handle all of the cases above and implement each correctly. It would also be undesirable for there to be multiple upgrade specifications as it would greatly increase the protocol attack surface.

5. Improving Cross Rollup UX

5.1. Ethereum as a Data Bus

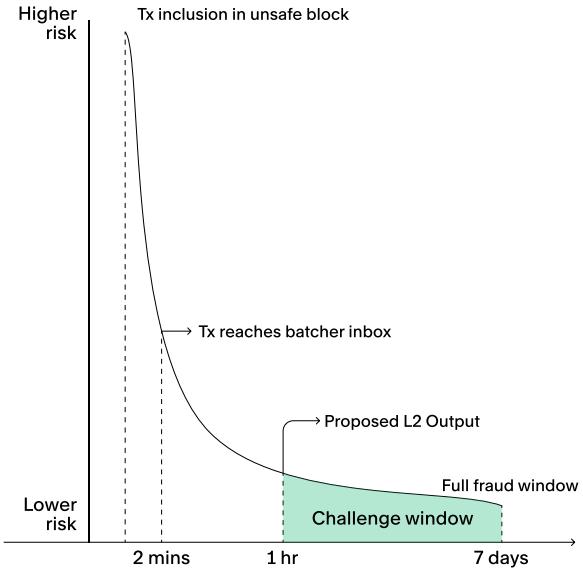
5.1.1. Native Verification Native verification refers to leveraging Ethereum itself for verification. Every rollup has a set of on-chain settlement contracts that attest to its state transition function. In the case of optimistic rollups, validity is inferred through waiting through a long and fixed fraud window. The diagrams below include components from the OP stack and assume the existence of fraud or validity proofs when describing risk over time.



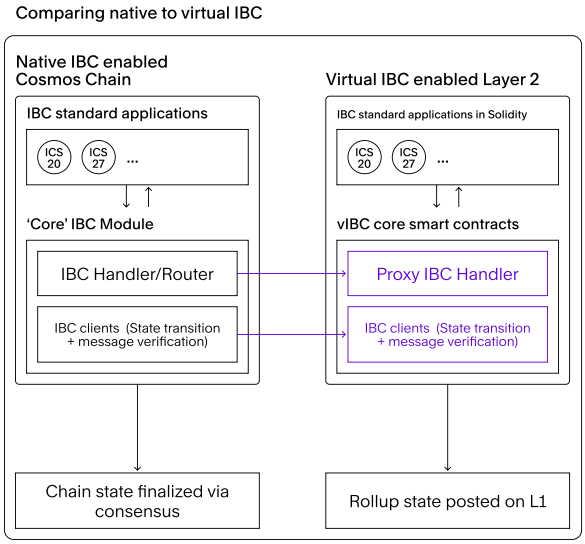
For example, a full packet round trip which includes bi-directional sends and receives using the native bridge of an OP stack rollup could take a little over 14 days. In the diagram below w show the verification process using the native bridge of a rollup which shows interoperability risk level over time. It starts with transaction inclusion in an unsafe block[24] till it's completely settled on Ethereum after the full fraud window has passed.

Polymer leverages the same verification mechanism as the native bridge but introduces a programmable challenge window to tradeoff between latency and censorship risk. Polymer's IBC clients for OP stack rollups check for proposals made to Ethereum via its settlement contracts which happens roughly every hour on mainnet. The clients can be configured with a minimum challenge period that is enforced upon verification.

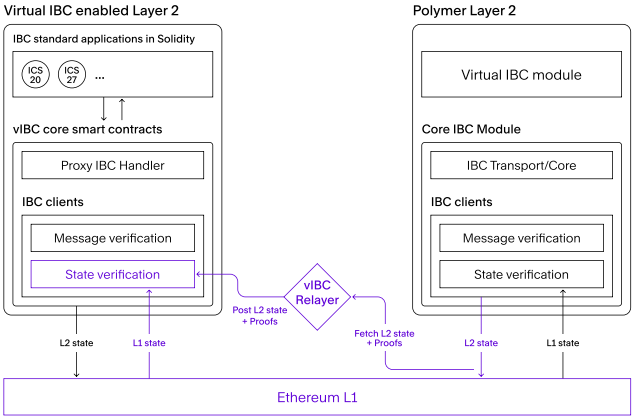
The programmable challenge window shown below allows packets to pass through anywhere between 1 hour up to 7 days depending on the level of security an application is looking for. This is akin to having configurable native L2 bridge security over Polymer.



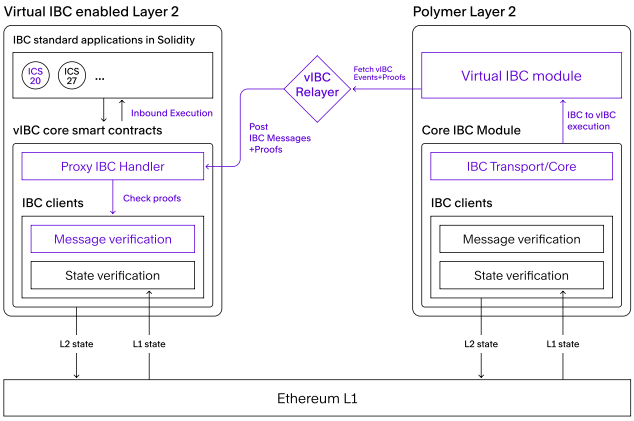
5.1.2. Virtual IBC over a Programmable Native Bridge. In this section, we'll explore how the programmable native bridge presented above is implemented over virtual IBC. Combining the two gets us native bridge like security with IBC features covered in section 4 and security model abstractions which we'll explore in more detail in section 5.3.1. In the diagram below, we compare what a native versus virtual IBC implementation looks like in above. On the virtual chain, the IBC handler is a proxy layer which is bound to the full IBC implementation on Polymer. The relevant vIBC components below and in the following diagrams are highlighted as we step through the execution flow.



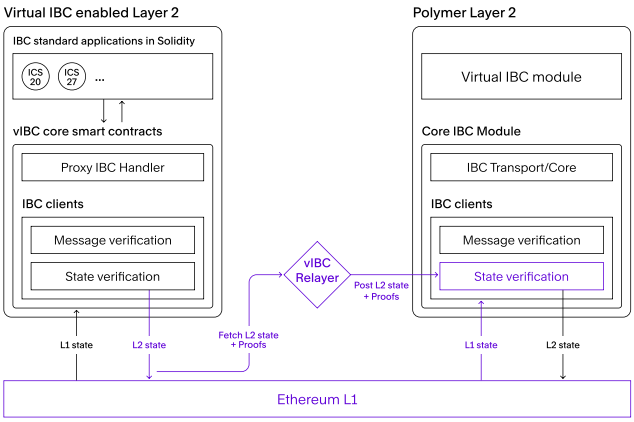
[1] The execution flow begins with the Polymer L2 receiving a packet intended for a virtual IBC enabled L2 which is not visualized. The diagram below depicts the first part of the process of communicating between Polymer and the virtual L2. An off-chain relayer picks the L2 state Polymer proposed on the L1 including proofs and posts them to a client on the L2. The posted state is then verified against the L1 state the L2 is derived from. This updates the virtual L2 with the latest state on Polymer.



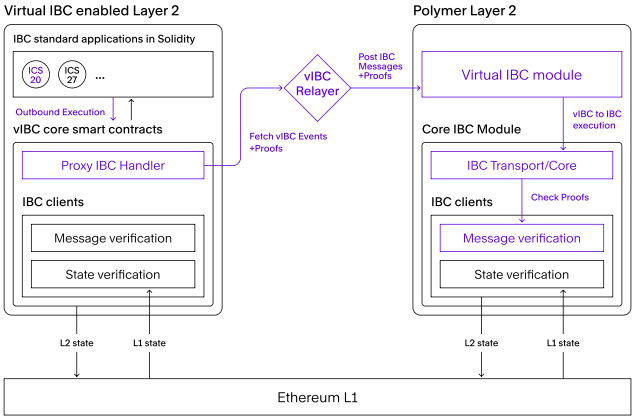
[2] After the counterparty L2 has the latest state on Polymer, an off-chain relayer can pick up the actual IBC message which can be a packet receipt or a channel handshake and its corresponding proofs against Polymer's state. These proofs are posted to the proxy IBC handler on the virtual L2 which verifies these message proofs against its latest verified state of Polymer. The message is then executed on the application layer.



[3] On the return path, we have state updates flowing in the opposite direction with an off-chain relayer updating Polymer with the latest state of the virtual L2. The virtual L2 state and proofs are validated against the L1 state that Polymer is derived from.

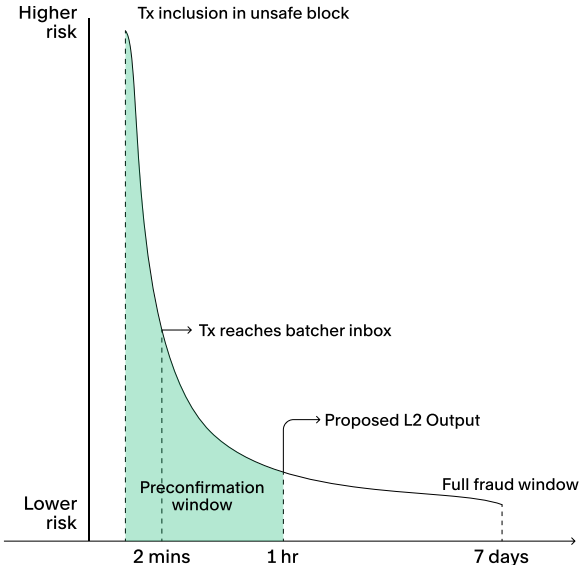


[4] After Polymer has the latest virtual L2 state, an off-chain relayer picks up the IBC events and event proofs and posts them to the virtual IBC module on Polymer. The virtual IBC module verifies the event proofs against its verified virtual L2 state. Afterwards, it converts virtual IBC execution into IBC execution by calling the core IBC module.



5.2. Pre-confirmations

The pre-confirmation window sits on the left side of the programmable challenge window (i.e. before the L2 proposes its state root to L1). It can improve the latency of rollup to rollup communication by changing the verification mechanism. Meaning developers can further tradeoff between risk and latency as visualized below.



There are various forms of pre-confirmations:

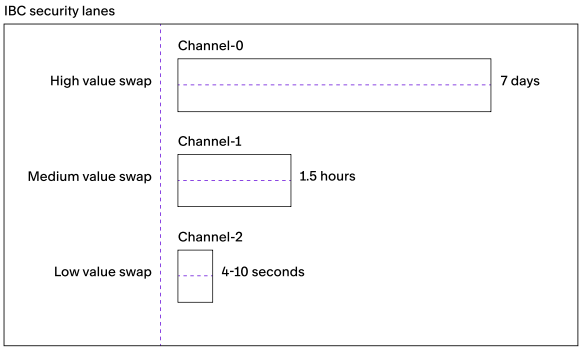
- **Sequencer.** The sequencer of a rollup can attest to its valid execution after attesting to transaction ordering within a block.
- **Oracle.** A third party oracle can attest to the valid execution of the rollup.
- **Eigenlayer AVS.** An AVS can be created to economically secure a third party attestation to the valid execution of a rollup. Both sequencers and oracles can be secured by an AVS.
- **Validity proof.** A zero knowledge (ZK) proof of validity can be used to directly verify the valid execution of a rollup without waiting for settlement to Ethereum.

A sequencer pre-confirmation can happen as soon as a transaction is included in an unsafe block. An AVS, oracle or validity proof pre-confirmation needs to wait for the underlying data to be published to the DA layer as well as having transaction ordering published to the finality layer.

IBC can support all of these pre-confirmation mechanisms as its core is a transport protocol which is not opinionated on the underlying verification mechanism used. Each of the verification modes above come with different levels of security with the most secure being a validity proof. Additionally, Eigenlayer AVS based pre-confirmations are non-fungible in their mechanism and security as well. With so many different options for verification, we need a way to abstract over these security models.

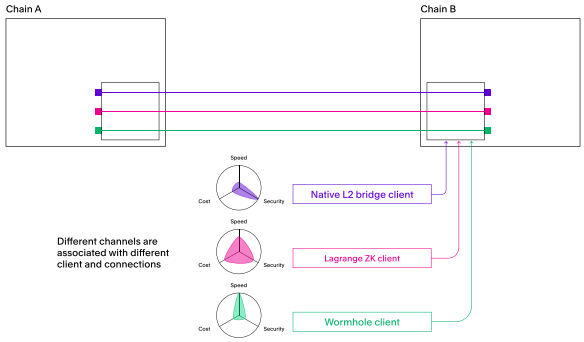
5.3. Modular Security

5.3.1. Security Model Abstraction. IBC abstracts away the underlying verification mechanism with clients and connections. Verification happens in the client which anyone can instantiate with configurable security parameters. A connection is formed over a pair of clients which represents connectivity between two chains defined over a specific security or trust model.



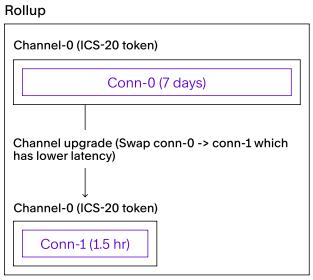
When sending packets between chains, a developer will specify an IBC channel ID. Since channels are established over a specific set of connection IDs, they are also specifying the security model they want to use for the delivery of the packet. In the example below, we show that we have 3 different channels that have different levels of security and developer UX allowing developers the flexibility to make decisions on a per packet basis.

Developers may want to route high value packets through the more secure channel while opting to route low value packets through less secure channels. Note that this calculus can easily change when the low latency channel is secured by a large amount of economic stake. We visualize how different clients can have varying tradeoffs between speed, cost and security.



5.3.2. Changing Security Models. A security model abstraction is no good if there's no way to switch between models. The channel upgrade protocol allows the configured connection IDs to be changed. Since a connection ID maps to a specific security or trust model, this means that it's seamless for an application such as an ICS-20 token to switch between security models.

This is especially relevant in the case of token bridges. Tokens bridged over the ICS-20 standard do not need to be migrated to change the underlying security model. A channel upgrade and connection ID change is all that is required and can be done with no user impact or downtime.



5.3.3. Aggregation Security Model. Both Uniswap[25] and Aave[26] have both conducted independent bridge assessments and both have concluded that a multi-bridge or aggregation based security model is the best solution for secure interoperability. The term bridge in the context of this paper is an IBC client or verification mechanism. Uniswap's bridge assessment[27] recommends multi-bridge verification for L1 and side-chain deployments while continuing to recommend usage of native L2 bridges for cross rollup interoperability.

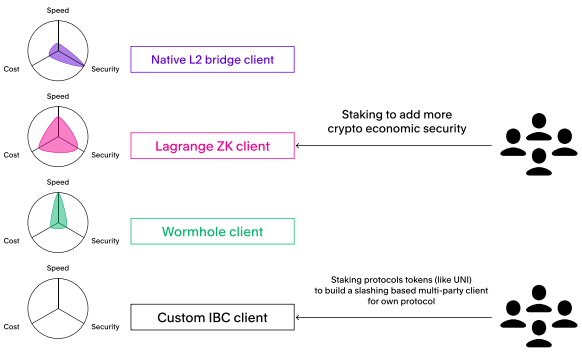
Aave also recognizes the need[28] to avoid vendor lock-in while also boosting security using multi-bridge verification only when there are security advantages. They also recommend usage of native L2 bridges for cross rollup interoperability. For connecting to other L1s, they have built a solution in house called Aave Delivery Infrastructure (a.DI) as a multi-bridge abstraction layer. Polymer not only supports programmable native L2 bridges for cross rollup interoperability but thanks to IBC also has the ability to abstract away and seamlessly switch between security models and aggregate over them.

6. Creating an Interoperability Economy

6.1. The Verification Marketplace

6.1.1. Participating as a Developer. Developers that want to participate in the verification marketplace can create IBC clients that wrap the verification mechanism they have implemented for a connected chain. For example, Lagrange is working on Lagrange State Committees[29] which can serve as a verification mechanism for optimistic rollups on Ethereum. Lagrange can make fees off proof generation for downstream users of their IBC client while securing fast rollup to rollup communication.

Another example is the conversion of the Wormhole guardian set into an IBC client. Wormhole Queries[30] offers an API for guardian attested on chain data. Their queries API serves attested `eth_call` query results back to the caller. Using this API, we can fetch an attested block hash from Ethereum using `blockhash()`. The Wormhole IBC client would then handle the verification of these query results.



6.1.2. Participating as a User. Users can participate in the verification marketplace without needing to learn to code. Users can stake to secure actively validated services[31] (AVS) such as a Lagrange State Committee. Staking adds economic security to the state committee while allowing the user to receive rewards for their help in securing rollup to rollup communication.

6.2. The Relayer Marketplace

6.2.1. Relayer Incentives. As mentioned in section 4.2, relayer incentives play an important role in the interoperability economy. The liveness of channels and connections as well as the timely delivery of packets depends on these incentives. There are currently two different relayer incentive schemes available depending on the ecosystem. ICS-29[32] or IBC relayer fee middleware is a scheme that allows a packet sender to specify a fixed incentive for packet receipt, acknowledgement and timeout. This incentive scheme is available for use within the Cosmos.

The CatalystAMM[33] team has worked on their own EVM based relayer incentive scheme called Generalised Incentives.[34] Their generalised incentive scheme supports gas refunds as well as an opt-in mechanism called strong ack incentives.[35] The idea here is that there is an ideal execution time between the delivery of a packet on the destination and the acknowledgement on the source chain. Packet senders can specify a `targetDelta` where the ack is incentivized more if more time has passed than `targetDelta` and the delivery is incentivized more if less time has passed.

Both features improve end user experience. Gas refunds ensure the user is not overcharged for gas spend while strong ack incentives ensure timely delivery of acknowledgements. Without the latter, a relayer could withhold relaying acknowledgements waiting for low gas prices on the source chain to maximize profit.

6.2.2. Becoming an Operator. Anyone can become a relayer operator by running a pre-built docker image of the Polymer relayer or building the code and running the binary directly. Relayers can be configured to watch for packets to relay between specific connection and channel paths. This means that operators can decide which applications or chains they wish to support.

7. Scaling the Decentralized Web

7.1. Asynchronous versus Atomic Composability

Different types of application composability affect both end user and developer experience as well as the useability of the technologies we're building. There are two major camps when it comes to scaling both Ethereum and the decentralized web broadly speaking with various builders in each camp claiming early victory.

The atomic composability camp is trying to tackle the problem from the angle of treating an arbitrary number of rollups as a single giant state machine. The approach here

generally involves treating each rollup as a state shard, leveraging ZK proofs to prove the execution of each shard and recursively composing the proofs together to produce a mega block.

The asynchronous composability camp is trying to tackle the problem by improving the UX of cross rollup execution. In collaboration with EigenLayer, AltLayer is working on a solution called restaked rollups[36] while Near is working on a super fast finality layer.[37] Both solutions greatly decrease the latency of cross rollup communication with tradeoffs.

While it's difficult to accurately predict the future winner in approach to scalability, we can look at the evolution of the internet so far to make an educated guess. ZK proofs are orders of magnitude more expensive to compute than running the raw computation over CPU. Both types of computation should improve in performance over time due to hardware and software advancements but the relative gap is unlikely to close significantly due to how ZK proof systems enforce their constraints.

A mental model for this is that a single CPU opcode can produce a large number of ZK opcodes that check if certain conditions are met instead. It's like you're building a road and driving at the same time. Some high value computation might justify the overhead of ZK for stronger security guarantees but it's not likely to be the majority of aggregated demand for decentralized compute in the long term.

From a user experience perspective, anything below 100ms is generally perceived as instantaneous, 1 second for seamless thought and 10 seconds to keep the users attention. These metrics are pulled from Nielsen Group publications on Usability Engineering (1993)[38] and a study on website response times.[39] The user doesn't care if the backend of the application they're using is executing things in atomic or asynchronous fashion. This is favorable to low latency asynchronous systems which is how most of the internet works. In the long term, we expect to see most decentralized compute flow through asynchronous systems while a small percentage of high value traffic flows through atomic systems

7.2. Redefining the Interoperability Layer

IBC is an extremely robust protocol for asynchronous communication which becomes increasingly important in a world where a high percentage of decentralized compute runs through asynchronously composed systems.

Polymer accelerates the industry towards the endgame of interoperability by allowing for the permissionless expansion of the IBC network and enabling the creation of a vibrant interoperability economy. Polymer reduces API fragmentation and allows builders to work off a single unified API in IBC.

Polymer can integrate with all the various innovations builders such as EigenLayer, Celestia, AltLayer, Lagrange and many others are working on to improve cross rollup UX and transaction throughput. We envision a future with millions of rollups on Ethereum connected via Polymer with fast asynchronous composability both with each other and external ecosystems such as the Cosmos.

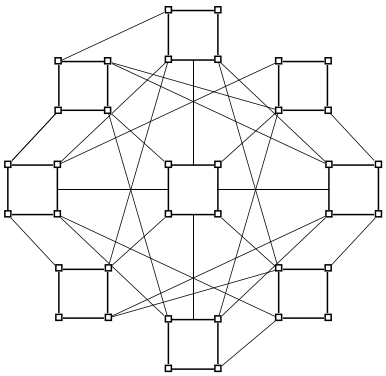
As IBC becomes ubiquitous across different ecosystems and frameworks, the Polymer network itself will expand, forming an interconnected IBC mesh of Polymer hubs belonging to different security domains. Polymer hubs will provide application discovery, routing configuration and modular security to connected rollups within each domain. The Polymer network as a whole will optimize the network topology and connectivity both inter and intra-domain.

Additionally, Polymer will be exploring some novel innovations at the interoperability layer such as establishing a p2p network instead of relayers between IBC connected chains in a network. This further accelerates IBC network expansion as new chain integrations will no longer require relayer modifications and operational overhead. Connecting to the IBC network will be as simple as adding an IBC integration and joining the p2p network.

8. Appendix

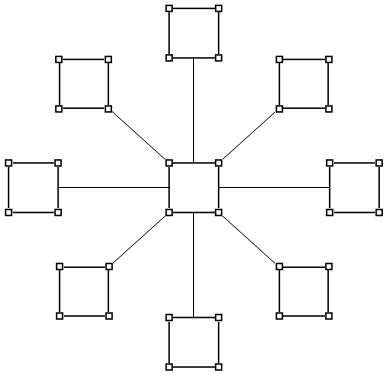
8.1. Evolving Network Topologies

8.1.1. **P2P**. Peer to peer connectivity is a common topology in interoperability protocols where each node or chain in the network needs to connect directly to another node to communicate. This is fine while the number of nodes in the network is small but does not scale as the size of the network grows. It would be impossible for all of the servers and devices connected to the internet to form direct connections with each other. N nodes in the network require N^2 connections for full connectivity.

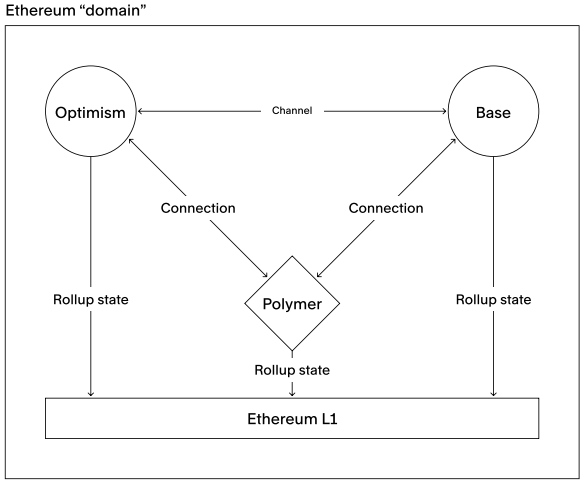


8.1.2. Hub and Spoke. A hub and spoke topology improves on the scalability of p2p by allowing traffic to flow through a middle hub. N nodes in the network now only require N connections for full connectivity. A new node joining the network only needs to form a single connection with the hub to access the rest of the network.

Although a hub and spoke topology is more scalable, introducing a trusted middleman (the hub) in between rollups breaks the trust model for rollup to rollup communication.



8.1.3. Domain Specific Hubs. Polymer is the first domain specific hub. A domain specific hub is an interoperability hub that retains the same trust model as the rollups it connects. Instead of being a sovereign chain with its own validator set, Polymer is architected as a L2 on Ethereum. This allows Polymer to offer scalable connectivity while also maintaining the trust model for rollup to rollup communication.



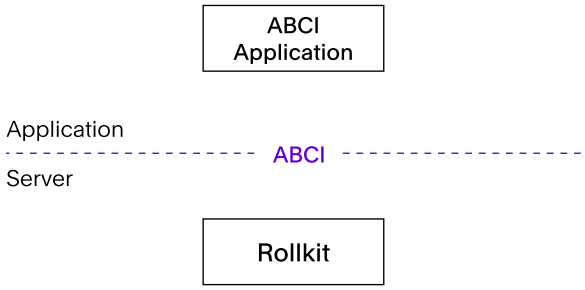
8.1.4. Mesh. Even hub and spoke approaches have scalability limitations when it comes to connectivity. A single hub can only handle so many connections. Polymer's work on ICS-33 enables the IBC network to form a mesh topology where nodes can be connected indirectly as a connection path exists between it and a counterpart. This network topology is similar to that of the internet today where nodes are neither connected directly nor is there a single central hub. Rather, a spanning network of routers or hubs connects a massive number of individual nodes.

This mesh topology motivates the long term vision for the Polymer network. Instead of just a single domain specific hub, the Polymer network will encompass a number of Polymer instances deployed to different domains. Each instance of Polymer connects rollups within its domain while instances of Polymer are connected across domains. This is the most scalable and secure way to connect a world of millions of rollups deployed across various settlement, finality and DA layers.

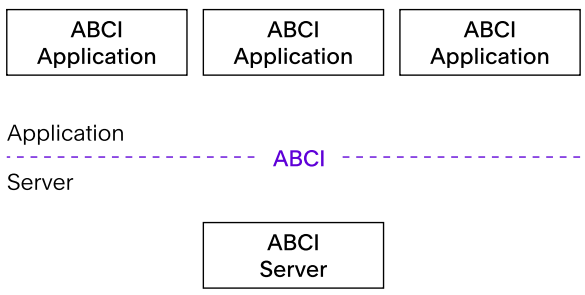
8.2. Evolving Rollup Design

8.2.1. Application / Server Model. The application / server model is common in traditional software development. We'll use this mental model to describe how blockchains and in particular rollups are designed. The Application BlockChain Interface (ABCI) in the Cosmos separates application logic from consensus or settlement. The most common example that most are familiar with is a sovereign app chain where there's a single application written using Cosmos SDK deployed over CometBFT consensus.

For rollups, the ABCI server is no longer CometBFT since consensus is not required. Rollkit is an ABCI server implementation that allows developers to deploy a chain as a sovereign rollup on top of Celestia or another DA layer.

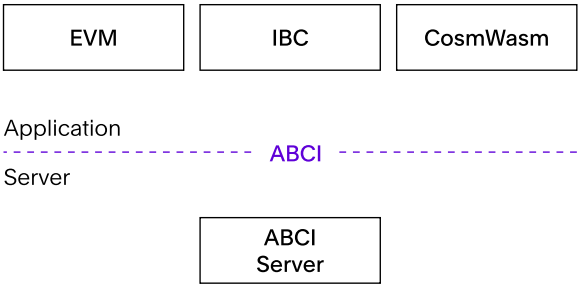


8.2.2. Multi-Application Rollups. A single chain does not need to be limited to a single application. Multiple ABCI applications can run simultaneously in the context of a single logical chain. This is analogous to multiple running docker containers over a single docker daemon.



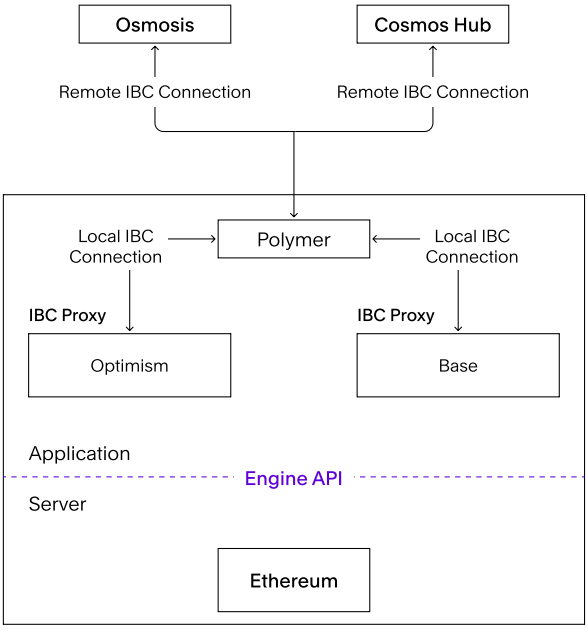
Each ABCI application can be responsible for a different functional component of the chain. In the example below we have a single chain running two

VMs and IBC for interoperability. This design improves on the modularity of frameworks like the Cosmos SDK by allowing each component to be implemented in different languages and/or runtimes and be run as separate processes within the chain.



8.2.3. Function Specific Rollups. Rollups running on top of a settlement layer are analogous to a number of ABCI or Engine API[40] applications running on top of a single server such as Ethereum. Currently, most rollups are EVM rollups running a generalized workload or any set of smart contracts that are deployed to the chain.

Polymer introduces the idea of a function specific rollup. A function specific rollup provides a specialized function to other rollups deployed on the same settlement layer. It does not support generalized compute or arbitrary smart contracts. Polymer specializes in providing native IBC interoperability as a function to other rollups on Ethereum. As other rollups on Ethereum specialize like Polymer, a single logical rollup can comprise of many function specific rollups. In the example below, we show Polymer providing IBC interoperability to other Ethereum rollups.



References

1. What is the OSI Model?
   https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/
2. What are IP & TCP?
   https://www.cloudflare.com/learning/ddos/glossary/tcp-ip/
3. The state of the layer two ecosystem.
   https://l2beat.com/scaling/summary
4. Data availability and light nodes.
   https://ethereum.org/developers/docs/data-availability#data-availability-committees
5. What is a reorg?
   https://www.alchemy.com/overviews/what-is-a-reorg
6. Kannan. What are the five properties that secure a confirmation rule?
   https://twitter.com/sreeramkannan/status/1683735133835370499?s=20
7. What is censorship resistance.
   https://www.bitcoin.com/get-started/what-is-censorship-resistance/
8. The Cosmos Hub is a Port City.
   https://medium.com/m/global-identity-2?redirectUrl=https%3A%2F%2Fblog.cosmos.network%2Fthe-cosmos-hub-is-a-port-city-5b7f2d28debf
9. A Framework for Building High Value Public Blockchains.
   https://github.com/cosmos/cosmos-sdk
10. A framework for building seamlessly scalable and interoperable rollups that can run on any blockchain.
    https://github.com/Sovereign-Labs/sovereign-sdk
11. A modular framework for rollups, with an ABCI-compatible client interface.
    https://github.com/rollkit/rollkit
12. Optimism is Ethereum.
    https://github.com/ethereum-optimism/optimism
13. Penumbra is a fully private proof-of-stake network and decentralized exchange for the Cosmos ecosystem.
    https://github.com/penumbra-zone/penumbra
14. Rust implementation of Namada, a Proof-of-Stake L1 for interchain asset-agnostic privacy.
    https://github.com/anoma/namada
15. Multi-hop IBC channels.
    https://github.com/cosmos/ibc/tree/main/spec/core/ics-033-multi-hop
16. 16 IBC specs.
    https://github.com/cosmos/ibc/tree/main/spec/app
17. ICS Documentation: Grammar and Command Fixes.
    https://github.com/cosmos/ibc/tree/main/spec/core
18. Ibc/spec/client/
    https://github.com/cosmos/ibc/tree/main/spec/client
19. Christopher Goes. Handler Interface.
    https://github.com/cosmos/ibc/blob/main/spec/core/ics-025-handler-interface/README.md
20. Inter-Blockchain Communication Protocol (IBC) implementation in Golang.
    https://github.com/cosmos/ibc-go
21. Rust implementation of the Inter-Blockchain Communication (IBC) protocol.
    https://github.com/cosmos/ibc-rs
22. Istio.
    https://istio.io/
23. Cloud-native high-performance edge/middle/service proxy.
    https://github.com/envoyproxy/envoy
24. Bedrock Differences.
    https://community.optimism.io/docs/developers/bedrock/how-is-bedrock-different/#chain-reorganizations
25. Uniswap Protocol.
    https://uniswap.org/
26. Open Source Liquidity Protocol.
    https://aave.com/
27. The all-in-one workspace for your notes, tasks, wikis, and databases.
    https://uniswap.notion.site/Bridge-Assessment-Report-0c8477afadce425abac9c0bd175ca382
28. BGD. a.DI - Aave Delivery Infrastructure.
    https://governance.aave.com/t/bgd-a-di-aave-delivery-infrastructure/13951#limitations-of-the-current-aave-cross-chain-governance-4
29. State Committees on EigenLayer via Lagrange.
    https://medium.com/@lagrangelabs/state-committees-on-eigenlayer-via-lagrange-7752f1916db4
30. The Flow of a Query.
    https://docs.wormhole.com/wormhole/queries/overview
31. AVS Developer Guide.
    https://docs.eigenlayer.xyz/avs-guides/avs-developer-guide
32. Aditya Sripal. Ethan Frey. General Fee Payment.
    https://github.com/cosmos/ibc/tree/main/spec/app/ics-029-fee-payment
33. Permissionless liquidity for modular chains.
    https://catalyst.exchange/
34. Standardization of incentives for AMBs.
    https://github.com/catalystdao/GeneralisedIncentives
35. Strong Ack Incentives (opt-in).
    https://github.com/catalystdao/GeneralisedIncentives?tab=readme-ov-file#strong-ack-incentives-opt-in
36. Restaked Rollups.
    https://altlayer.io/restaked-rollup
37. NEAR Foundation and Eigen Labs Partner to Enable Faster, Cheaper Web3 Transactions for Ethereum Rollups via EigenLayer.
    https://pages.near.org/blog/near-foundation-and-eigen-labs-partner-to-enable-faster-cheaper-web3-transactions-for-ethereum-rollups-via-eigenlayer/
38. Usability Engineering : Book by Jakob Nielsen.
    https://www.nngroup.com/books/usability-engineering/
39. Jakob Nielsen. Website Response Times.
    https://www.nngroup.com/articles/website-response-times/
40. Engine API — Common Definitions.
    https://github.com/ethereum/execution-apis/blob/main/src/engine/common.md