PolyNetwork: An Interoperability Protocol for Heterogeneous Blockchains

Poly Team

2020-05-15

Introduction

Blockchain technology can make a significant impact on our daily life. Finance[1][2] , supply chain[3], identity management[4], digital assets[5][6] , and distributed storage[7][8] are some areas that allow us to see infinite potential of a new generation of decentralized internet protocols. Blockchain can be a reshaping mechanism for social-economic operations. However, due to the massive development of blockchain, the phenomenon of information isolated island is re-emerging. Information exchange and asset replacement between blockchain infrastructures are limited. If blockchains are regarded as isolated computers, then we might as well go back to the nascent era of the internet in the 1990s.

At present, the two most representative solutions on the market are Cosmos[9] and Polkadot[10]. Cosmos is mainly a cross-chain protocol for homogeneous chains, and the IBC protocol has not been fully developed yet.

In order to build a better next-generation internet infrastructure, we have launched a new cross-chain technology, the Poly Network. Poly Network is based on the side-chain/relay mode and adopts a two-layer architecture. It employs the Poly chain as a cross-chain coordinator, multiple homogeneous chains as cross-chain transaction executors, and Relayer as a cross-chain information porter. By resolving issues such as trust, security and transaction issues of chain data, we have realized a safe, easy-to-use and efficient cross-chain system.

Our protocol includes the following two parts:

1. a solution to implement interoperation between blockchains
2. a two-phase commit protocol for atomic cross chain transactions

This groundbreaking interoperability protocol mainly has the following advantages:

- Wide range of support and strong versatility

  Support for heterogenous and homogeneous chains, including BTC, ETH, NEO, Ontology and Cosmos. Our protocol can quickly support other heterogeneous chains and don't need to make a lot of changes to the underlying architecture. For the heterogeneous chain that are already supported, the homogeneous chain fork from the chain can join in seamlessly.

- Easy to join in

  If the underlying architecture of the chain supports smart contract, only two contracts need to be deployed to support cross-chain. If not, only two additional functional modules need to be added.

- Support atomic transaction

  Our protocol aims to achieve the ultimate and atomic nature of cross chain transactions, with a focus on cross chain smart contract interactions to extend the scope of application scenarios for decentralized applications.

- Support cross-chain of arbitrary information

  Our protocol is not limited to cross-chain of assets, it will support cross-chain of arbitrary information.

- Security Enhancement

  Our protocol is based on cryptography, and adds a complex set of mechanisms at the technical and operational levels to enhance the security of cross chain transactions and interactions.

- Eco Friendly

  Our protocol is designed for cross chain, and neither issuing tokens nor a dedicated smart contract system, it is more able to deal with compliance issues, the alliance chain and private chain can also join in.

Need and Application

Blockchains are facing trade-offs among decentralization, security and scalability. Different preferences of these trade-offs make different blockchains have their own specific strengths and limitations. Then different blockchains may focus on different areas. For example, NEO[6] may focus on digital assets while Ontology[11] may focus on digital identity. We believe that effective interoperation between blockchains will allow users, accounts, features, applications and value to flow freely throughout the ecosystem.

Thus several potential use cases can be achieved via interoperation[12]:

- Portable assets

  For instance, being able to take one unit of Fedcoin (a hypothetical government-issued digital asset) from its "home ledger" that is ultimately authoritative on its ownership, securely move it to another chain, trade it, use it as collateral or otherwise take advantage of it on that chain, and be confident that the option to move the Fedcoin back to its home ledger is always available if desired.

- Payment-versus-payment or payment-versus-delivery

  Essentially, the goal is to allow user X to transfer digital asset bundle A to user Y in exchange for Y transferring digital asset bundle B to user X (where A and B are on different chains, and X and Y have accounts on each chain), in a way that is guaranteed to be atomic and secure.

- Cross-chain oracles

  For instance, one might imagine a smart contract on one chain that performs some action with some address only when it receives proof that an identity oracle on another chain specifies that the address is a particular unique identity.

- Asset encumbrance

  Lock up asset bundle A on chain X and have locking conditions be dependent on activity on chain Y. Use cases include liens, collateral in financial derivatives, bankruptcy clawbacks, court orders and various use cases involving security deposits.

- General cross-chain contracts

  For example, paying dividends on chain X to holders of an asset whose ownership is registered on chain Y.

  We can summarize the above use cases into the following three aspects:

- Value Transfer

  Digital assets can flow freely via blockchain interoperation which can boost cryptocurrency exchanges, cryptocurrency loans, multi-cryptocurrency payments and settlements, digital asset transactions and digital asset investing and financing. Beside this, all the values on blockchain can be transferred among heterogeneous blockchains.

- Function Calls

  Different applications and services can be developed on suitable blockchains. Another application scenario for blockchain interoperation is cross chain smart contract call which can make computing, storage, network resources and ecology distributed among blockchains.

- Generalized Locking

  “Atomic” is a widely used word in distributed scenarios, which means “uncuttable”, i.e. there is a guarantee that either all executions happen or no execution does. Actually a lock is a synchronization mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution. Based on the previous two aspects, generalized locking among blockchains is a way to control the business workflow and expand the ablility of distributed applications.

Previous work

The concept of blockchain transactions that are somehow connected across blockchains has been proposed several times under a wide variety of systems, with the first high-impact publication being Pegged Sidechains[13], which proposed a protocol under which a coin or token could be transferred from a main chain to its child chains, allowing for heterogeneous protocols to be integrated into a chain without requiring alterations in the main chain's protocol. The inter-chain interactions proposed by this system are limited to the use-case mentioned and therefore the system may be unsuitable if there are other requirements.

Interledger protocol[14], proposed by ripple LABS, aims to connect different ledgers and achieve synergy between them. Interledger protocol is applicable to all accounting systems and can accommodate the differences of all accounting systems. The goal of this protocol is to create a unified global payment standard and create a unified network financial transmission protocol. It allows two different billing systems to freely transfer money to each other through a third party "connector". The billing system does not need to trust the "connector" because the protocol uses a cryptographic algorithm to create a fund escrow for the two billing systems so that only when all parties agree on the transaction, they can trade with each other.

BTC Relay enables ethereum smart contracts to securely validate BTC transactions without requiring any third party. It is a cross-chain payment solution. BTC Relay is a one-way solution that only introduces BTC into Ethereum but does not transfer BTC from ethereum back to the BTC system. The core of BTC Relay is to keep a lightweight BTC transaction data in Ethereum, which is the BTC block head Hash data. When it is necessary to verify the validity of a BTC transaction, user just need to submit the transaction information and the merkle path, and the smart contract can verify the validity of the transaction through the Hash saved in the contract.

Hashed Timelock Contract (HTLC)[15] is a cross-chain atomic exchange protocol. It completes the point-to-point cross-chain interchange among users in a decentralized way. HTLC decomposes the exchange process of cross-chain assets into several sub-processes to ensure that all the exchange process between the participants will either succeed or fail.

WanChain[16] also supports cross-chain transactions between mainstream public chains, but first needs to complete registration on WanChain to ensure that WanChain uniquely identifies the chain. It uses multi-party computation and threshold secret-sharing tachnology to implement cross chain.

A somewhat more advanced concept, Cosmos[9], it is a network of many independent blockchains, called zones. The zones are powered by Tendermint BFT.

The first zone on Cosmos is called the Cosmos Hub. The Cosmos Hub is a multi-asset proof-of-stake cryptocurrency with a simple governance mechanism which enables the network to adapt and upgrade. The hub and zones of the Cosmos network communicate with each other via an inter-blockchain communication (IBC) protocol, a kind of virtual UDP or TCP for blockchains. All inter-zone token transfers go through the Cosmos Hub, which keeps track of the total amount of tokens held by each zone. The hub isolates each zone from the failure of other zones.

Finally, Polkadot[10] is a scalable heterogeneous multi-chain. It is designed to provide no inherent application functionality at all, and enables cross-blockchain transfers of any type of data or asset, not just tokens. Polkadot provides the bedrock "relay-chain" upon which a large number of validatable, globally-coherent dynamic data-structures may be hosted side-by-side, these data-structures are called "parallelised" chains or parachains, and there is no specific need for them to be blockchain in nature.

Definition

In the following sections we formalize the concept of blockchain and what it means for a blockchain to have crosschain capabilities.

Blockchain

A blockchain is essentially a distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participant parties. Each transaction in the public ledger is verified by consensus of a majority of the participants in the system. Once entered, information can never be erased. The blockchain contains a fixed and verifiable record of every single transaction ever made[5].

According to formal definitions of blockchain as a state machine in the literature[17], a blockchain should be formalized as a probabilistic state machine due to the possible existence of conflicting forks. In this paper we will simplify this concept by adding the assumption that blockchains will not fork (it would be possible to add requirements such as a having a minimum of 6 blocks confirmed on top of another block before considering such a block as part of the state machine in order to achieve this property on some blockchains without finality, like Bitcoin[18] in practice). The assumption could be realized by governance solutions in practice, such as the requirement of pledge and compensation for nodes. This assumption allows us to model the blockchain as a deterministic state machine, concretely as a Moore machine[19] M:

$$M = (\mathbb{S}, \mathbf{S}^{(0)}, \mathbb{I}, \mathbb{O}, f_{\texttt{transition}}, f_{\texttt{output}})$$

The definition of each parameter will be introduced in the following sections.

A solution based on probabilistic state machines may be proposed in further research.

State Domain

State domain $\mathbb{S}$ is the domain of states the blockchain can be in, where a state is defined as a sequence of all the blocks that have been included in the blockchain (the inclusion process is defined by the transition function) up until a point in time and a block is a sequence of transactions with a header. Therefore, $\forall \mathbf{S} \in \mathbb{S}$, $\mathbf{S}$ is an ordered subset of $\mathbb{I}$.

$$\mathbb{S} = \mathbb{I}^m$$

Initial State

Initial state $\mathbf{S}^{(0)} \in \mathbb{S}$ is the starting state of the state machine. which is an empty sequence.

$$\mathbf{S}^{(0)} = []$$

Blockchains that include a genesis block[20] fit into this definition if we regard the genesis block as the first block to be added.

$$\mathbf{S}^{(0)} = [I_{\texttt{genesis}}]$$

Input Alphabet

Input alphabet $\mathbb{I}$ is the finite set of correct blocks that can be generated and broadcasted. The exact definition of this concept may vary from blockchain to blockchain but in all these cases, it is the basic element that enables the modification of blockchain state. For this reason we will use the block as the basic unit upon which we will base the construction of the other concepts.

We formally define block the following way:

$$\mathbb{I} = \mathbb{H} \times \mathbb{X}^m$$

Where $\mathbb{H}$ is the set of possible block headers, which are defined by a tuple of several values such as miner signature, transaction merkle root, state root and several others that may vary from blockchain to blockchain (see Bitcoin's Block Header[21] and Ethereum's Block Header[22] for detailed definitions of two particular cases), $\mathbb{X}$ is the set of valid transactions[17] and $m \in \mathbb{N}$.

Therefore, $\forall I \in \mathbb{I}$, I consists of the block header H and a sequence of transactions $\mathbf{X}$. The header H contains the proof of the sequence of transactions $\mathbf{X}$ and the output $O^{(t)} = f_{\texttt{output}}(\mathbf{S}^{(t)})$ at block height t. In other words, one can say that a transaction X is confirmed at block height t if one can provide the proof that matches the block header $H^{(t)}$ at block height t. This is usually called merkle root in many blockchains. Similarly, one can also prove the status at a block height.(The definition of status and output is defined in the Output Alphabet section) This is usually called state root in many blockchains. In this paper we won't go too deep in detail regarding block structures because different blockchains may have different implements, but probably their block headers contain such essentials.

$$I = (H, \mathbf{X})$$

For the rest of this paper, the following notations will be used:

We use $X \in \mathbf{X}$ to represent that there is an object X in the sequence $\mathbf{X}$.

$$X \in \mathbf{X} \to \exists i \in \mathbb{N}, X_i = X$$

We use $X \in I$ to represent that the transaction X is included in block I

$$X \in I \to \exists \mathbf{X}, X \in \mathbf{X}, I = (H, \mathbf{X})$$

We use $X \in \mathbf{S}^{(t)}$ to represent that the transaction X has been confirmed by the blockchain at block height t

$$X \in \mathbf{S}^{(t)} \to \exists I \in \mathbf{S}^{(t)}, X \in I$$

In UTXO model, a transaction is a transfer of value between two users[1723] . While in account model like ethereum or NEO, a transaction is identical to any distributed or OLTP transaction (TPP Council 2010) that acts on some data[622] . However, in our model, a transaction is a set of data passed to the output function signed by the sender.

Output Alphabet

The output alphabet $\mathbb{O}$ is the finite set of the results that can be obtained by evaluating the output function $f_{\texttt{output}}$ on a state, resulting in a view that represents the circumstances under which new transactions will be executed. For example, in Bitcoin, it would be represented by $\mathbb{N}^{2^{160}}$ which would be the set of all possible balances for all possible addresses. Each blockchain has their own output alphabet (for example, UTXO model and account model have different output alphabets).

We use output to represent a item in the output alphabet and we use status to denote a specific view of the output. For example, in Bitcoin, all the balances of all accounts are output, while the balance of one account is a status.

Transition Function

Transition function $f_{\texttt{transition}} : \mathbb{S} \times \mathbb{I} \to \mathbb{S}$ maps a state and an input to the next state.

Blockchains organize data in blocks, and update the entries using an append-only structure. More precisely, data can only be added to the blockchain in *time-ordered sequential order*, and it cannot be removed or changed.

For a block I, applying $f_{\texttt{transition}}$ will cause changes to state $\mathbf{S}$. As it is shown below, the current state (block sequence) $\mathbf{S}$ and the input (a new block) I are the inputs of transition function $f_{\texttt{transition}}$, while $n_{\mathbf{S}}$ is the length of $\mathbf{S}$.

$$f = \lambda \mathbf{S}.\lambda I.\mathbf{S} + I$$

$$\mathbf{S} + I = [S_0, S_1, \ldots, S_k, \ldots, S_{n_{\mathbf{S}}}, I]$$

Output Function

Output function $f_{\texttt{output}} : \mathbb{S} \to \mathbb{O}$.

Unlike the transition function, the application of the output function represents the process of executing all the transactions contained in the blocks that a state is composed of. This execution will result in an $\mathbf{O} \in \mathbb{O}$ which represents the circumstances in which any transactions in the immediately next block would be executed. The specifics of what this function does vary from blockchain to blockchain and depend on the capabilities of the blockchain's VM[24], it's transaction model and many more details, so, in order to remain general, the details of this function will stay unspecified and would have to be adapted for each blockchain.

Cross Chain

While in the initial few years of the blockchain industry one may have been forgiven for thinking that there would be only "one blockchain to rule them all", in recent times such a possibility has been receding further and further from reality. Within the public blockchain space, different projects have been staking out different regions of the trade-off space between security, privacy, efficiency, flexibility, platform complexity, developer ease of use and even what could only be described as political values. In the private and consortium chain space, the notion that different chains exist for different industries - and even different chains within the same industry - is even less controversial and arguably universally understood as obvious. In such a world, one natural question that emerges is: how do these chains interoperate[12]?

To solve this problem, we start by giving a definition of chain interoperability, also known as cross chain, which we divide into two types: write and read operations. All high level operations (such as asset transfer, asset exchange, cross chain contract call, etc.) can be constructed using these two basic building blocks.

Here is an example of a cross chain assets transfer (solid arrows are used to represent write operations while dotted arrow are for read operations):

1. Some assets are locked on chain A.
2. A cross chain call is performed from chain A to chain B to release assets there. This is a write operation.
3. Before releasing the assets on chain B, a check is performed to ensure that the assets on chain A have been locked correctly. This is a read operation.
4. If the check passes, the assets are released on chain B.
5. If it does not, the locked assets on chain A can be unlocked after checking that the releasing progress failed on chain B. This step requires a read operation.

The following terminology will be used in the definitions of the read and write operations:

source chain The chain initiating the cross chain operation

destination chain Target chain of the cross chain operation

participant chain Source chain or destination chain; in the rest of this paper, we use $\mathcal{M}$ to denote all participant chains

cross chain user The user who wants to do a cross chain operation and owns an address on the source chain. It can only send transactions on the source chain but it wants to interoperate with the destination chain.

Write Operation

A write operation is an operation that will affect the state of the destination chain. For example, calling a contract on the destination chain from the source chain is a write operation because it needs to append a transaction to the state of the destination chain.

For the rest of this paper, $Q_{\texttt{write}}$ will denote a write operation.

A write operation is finished if the transaction $X_{\texttt{dst}}$ which contains the write operation $Q_{\texttt{write}}$ has been confirmed by the destination chain at block height $t_{\texttt{dst}}$.

$$\exists t_{\texttt{dst}} \in \mathbb{N}, X_{\texttt{dst}} \in \mathbf{S}_{\texttt{dst}}^{(t_{\texttt{dst}})}$$

Note that a finished write operation may not be a successful write operation because the transaction $X_{\texttt{dst}}$ may have failed. If a transaction has failed, it is still included in a block and confirmed, which means the state has been changed by the transaction but the output has not.

Read Operation

A read operation is any operation that will not affect the state of the destination chain but it will return back the status (a specific view of the output) of the destination chain.

For the rest of this paper, $Q_{\mathtt{read}}$ will denote a read operation.

A cross chain user can get the status of the destination chain at a specific block height $t_{\mathtt{dst}}$ from the source chain by using a read operation, receiving a specific view of the output $O_{\mathtt{dst}}^{(t_{\mathtt{dst}})}$.

For example, if a cross chain user wants to know the balance of their accounts on the destination chain a read operation would be used.

Solution

We propose the following solution to solve the cross chain problem. The solution consists of four parts.

The first two parts of the solution are Write Operation Solution and Read Operation Solution, in which we introduce relayer entity. For write operation solution, relayer would help transfer the write operation from the source chain to the destination chain. For read operation solution, relayer would help transfer the status on the destination chain to the source chain. Whether the relay was performed correctly could be validated by both the source chain and the destination chain, which establishes trust on the relayer.

Actually, the scenario of write operation and read operation could be extended to one source chain interoperating with multiple destination chains. For the sake of simplicity, only the scenario of one source chain and one destination chain would be described below, as the extension is easy to realize.

Moreover, The write operation solution and read operation solution are based on a mechanism of synchronizing block header, which is introduced in the Synchronize Block Header section, and we propose a two-phase protocol for atomic cross chain transaction based on the write operation to ensure the final consistency of status of chains involved in the cross chain operations.

Write Operation Solution

Currently, transactions in most blockchains require the existence of a sender address, so, in order to minimize the changes to the underlying protocol needed for cross chain, cross chain transactions will also contain a sender address, which will be the cross chain user itself on the source chain and an entity called relayer on the destination chain.

The main idea behind our solution for write operations is that a relayer entity will help the cross chain user transfer the write operation from the source chain to the destination chain.

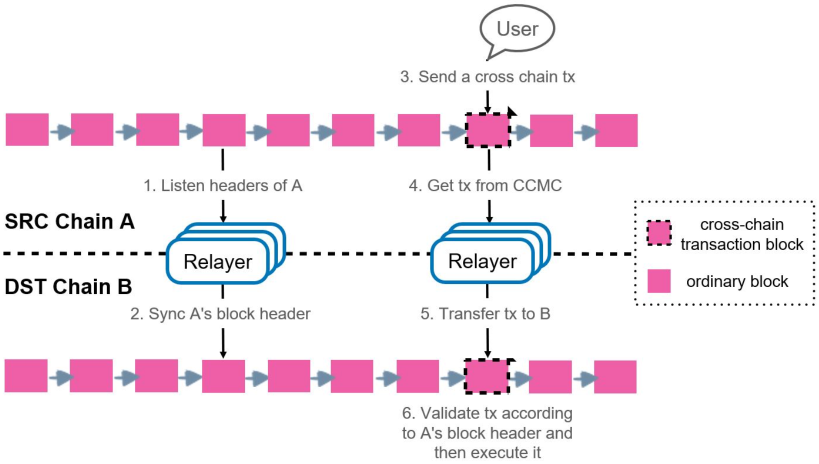

Figure 1: solution overview for write operayion

This relayer mentioned previously, which could be any node (including the cross chain user itself) that is connected to the destination chain as well as the source chain, will simply relay some transactions from one chain into the other, without requiring any trust due to the validation checks performed at several stages of the protocol.

The details of each step will be discussed in the following sections.

How is a Cross Chain Write Operation Initiated on the Source Chain?

To initiate a cross chain write operation on the source chain, a cross chain user must send a transaction $X_{\texttt{src}}$ on the source chain, calling the cross chain management contract and providing the write operation $Q_{\texttt{write}}$ to perform on the destination chain $M_{\texttt{dst}}$, which will be logged.

After the cross chain management contract is called, the transaction $X_{\texttt{src}}$ is confirmed on the source chain $M_{\texttt{src}}$ at the block height $t_{\texttt{src}}$, which means,

$$X_{\texttt{src}} \in \mathbf{S}_{\texttt{src}}^{(t_{\texttt{src}})}$$

The destination chain $M_{\texttt{dst}}$ and the write operation $Q_{\texttt{write}}$ can be inferred from the transaction $X_{\texttt{src}}$ after its successful execution.

Example:

1. A cross chain user calls a contract on the source chain.

2. The contract needs to do a cross chain write operation, so it calls the cross chain management contract and tells it the destination blockchain $M_{\texttt{dst}}$ and the write operation $Q_{\texttt{write}}$.

How does the relayer get the Cross Chain Write Operation from the Source Chain?

Only if the transaction $X_{\texttt{src}}$ is executed successfully, the cross chain management contract will log the destination blockchain $M_{\texttt{dst}}$ and the write operation $Q_{\texttt{write}}$, which will be obtained by a relayer when it retrieves the log of the cross chain management contract. In other words, the relayer gets such information from the output $O_{\texttt{src}}^{(t_{\texttt{src}})}$ of the source chain at block height $t_{\texttt{src}}$.

How does the relayer send the Cross Chain Write Operation to the Destination Chain?

When a relayer gets a cross chain write operation from the source chain, the destination blockchain $M_{\texttt{dst}}$ and the write operation $Q_{\texttt{write}}$ will be extracted.

Next, such a relayer would send a transaction $X_{\texttt{dst}}$ on the destination chain. This transaction, $X_{\texttt{dst}}$, must call the cross chain management contract and provide it the write operation $Q_{\texttt{write}}$, which will be executed by it after the validation process.

So after the cross chain management contract is called, the transaction $X_{\texttt{dst}}$ is confirmed on the destination chain $M_{\texttt{dst}}$ at the block height $t_{\texttt{dst}}$, which means,

$$X_{\texttt{dst}} \in \mathbf{S}_{\texttt{dst}}^{(t_{\texttt{dst}})}$$

The following diagram illustrates the steps that will be taken if $Q_{\texttt{write}}$ is a cross chain contract call:

1. A relayer calls the cross chain management contract and provides the write operation $Q_{\texttt{write}}$ to the cross chain management contract.
2. The cross chain contract executes the write operation $Q_{\texttt{write}}$, which leads to a contract call.

How to Validate the Cross Chain Write Operation on Destination Chain?

When the cross chain management contract is called by the relayer, who must provide both the write operation $Q_{\texttt{write}}$ and a proof of its inclusion on the source chain, the contract will first validate the write operation $Q_{\texttt{write}}$ using the block header $H_{\texttt{src}}^{(t_{\texttt{src}})}$ on the source chain at the block height $t_{\texttt{src}}$, which should be straightforward if it is able to retrieve the valid block header of the source chain.

Therefore, the relayer must provide both the write operation $Q_{\texttt{write}}$ and a proof of its inclusion on the source chain to the cross chain management contract. And the cross chain management contract can validate it by the block header $H_{\texttt{src}}^{(t_{\texttt{src}})}$ on the source chain at the block height $t_{\texttt{src}}$.

But how can a contract on the destination chain retrieve the block header $H_{\texttt{src}}^{(t_{\texttt{src}})}$ on the source chain? The system used to obtain such capability will be discussed in section Synchronize Block Header, for now it will just be assumed it's possible.

Who will pay for the Cross Chain Write Operation?

A transaction fee should be paid when executing transactions in any blockchain, and given that the solution discussed involves many transactions on different chains, there must be entities responsible for paying the transaction costs. On the source chain, the cross chain user is the one sending the transaction $X_{\texttt{src}}$, so the burden of paying for it falls on him, while, on the destination chain, the relayer will be the one sending the transaction $X_{\texttt{dst}}$ and covering it's cost.

Given that the sending $X_{\texttt{dst}}$ on the destination chain is not going to benefit the relayer directly, there should be some incentives to make relayers be willing to do their work, incentives which may cover the transaction fee.

How to know whether a Cross Chain Write Operation has been executed successfully?

This can be achieved by validating the state root in the block that contains the transaction $X_{\texttt{dst}}$.

- If the validating party is a user or a relayer in the destination blockchain $M_{\texttt{dst}}$, they can get the results directly without requiring any cross chain operation.
- If it's a user in the source blockchain $M_{\texttt{src}}$ instead, they can get the results by means of a cross chain read operation, which will be discussed in the next section.

Write Operation Summary

This section will summarize our write operation solution and provide an overview of our cross chain interoperability process.

`USER` in the diagram is the cross chain user. `SRC_MANAGER` is the cross chain management contract on the source chain. `RELAYER` is the relayer. `DST_MANAGER` is the cross chain management contract on the destination chain.

1. Initiate

   To initiate a cross chain write operation on the source chain, `USER` sends a transaction $X_{\texttt{src}}$ on the source chain. The transaction $X_{\texttt{src}}$ must call `SRC_MANAGER` and provide the following information:

   - destination blockchain $M_{\texttt{dst}}$
   - write operation $Q_{\texttt{write}}$

2. Hunt

   After transaction $X_{\texttt{src}}$ is executed successfully, `SRC_MANAGER` will log the following information:

   - destination blockchain $M_{\texttt{dst}}$
   - write operation $Q_{\texttt{write}}$

   `RELAYER` will then get the information by retrieving the log of `SRC_MANAGER` and proceed to the next stage.

3. Transfer

   `RELAYER` sends a transaction $X_{\texttt{dst}}$ on the destination chain. The transaction $X_{\texttt{dst}}$ must call the `DST_MANAGER` and include the following information:

   - write operation $Q_{\texttt{write}}$
   - inclusion proof

   The main function of the proof is to allow the validation of the existence of the log containing the write operation on the source chain. For example, the proof can be a merkle path to be used in SPV[25].

   The address a can help `RELAYER` prove his work successfully.

4. Execute

   After receiving the proof, `DST_MANAGER` validates the write operation using the block header $H_{\texttt{src}}^{(t_{\texttt{src}})}$ on the source chain at block height $t_{\texttt{src}}$.

   If the validation is successful, `DST_MANAGER` will execute the write operation $Q_{\texttt{write}}$.

5. validate

   After $Q_{\texttt{write}}$ was executed successfully, the relayer can make a proof and send to the source chain, source chain will validate proof and update the status of the transaction on the log.

Read Operation Solution

Similar to the write operation solution, the main idea for the read operation solution is that a relayer will help the cross chain user transfer the status on the destination chain to the source chain, where the status returned will be validated, allowing for the role of relayer to not require any trust.

The details of each step will be discussed in the following sections.

How is a Cross Chain Read Operation initiated on the Source Chain?

To initiate a cross chain read operation on the source chain, the cross chain user should send a transaction $X_{\texttt{src}}$ there, calling the cross chain management

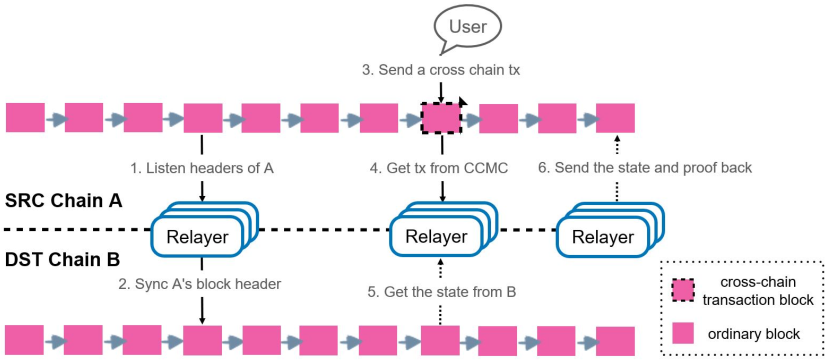

Figure 2: solution overview for read operayion

contract and providing the destination blockchain $M_{\texttt{dst}}$ and the read operation $Q_{\texttt{read}}$, which will be logged by the contract.

After the cross chain management contract is called, the transaction $X_{\texttt{src}}$ is confirmed on the source chain $M_{\texttt{src}}$ at the block height $t_{\texttt{src}}$, which means,

$$X_{\texttt{src}} \in \mathbf{S}_{\texttt{src}}^{(t_{\text{src}})}$$

The destination chain $M_{\texttt{dst}}$ and the read operation $Q_{\texttt{read}}$ can be inferred from the transaction $X_{\texttt{src}}$ after which the execution is deemed successful.

Example:

1. A cross chain user calls a contract on the source chain.
2. The contract needs to know the cross chain user's balance on the destination chain, so it calls the cross chain management contract and tells it the destination blockchain $M_{\texttt{dst}}$ and the read operation $Q_{\texttt{read}}$.

How does the relayer get the Cross Chain Read Operation from the Source Chain?

Only if the transaction $X_{\texttt{src}}$ is executed successfully, the cross chain management contract will log the destination blockchain $M_{\texttt{dst}}$ and the read operation $Q_{\texttt{read}}$, which will be obtained by a relayer when it retrieves the log of the cross chain management contract. In other words, the relayer gets such information from the output $O_{\texttt{src}}^{(t_{\text{src}})}$ of the source chain at block height $t_{\texttt{src}}$.

How does the relayer get the Status of the Destination Chain?

When a relayer gets a cross chain read operation from the source chain, the destination blockchain $M_{\texttt{dst}}$ and the read operation $Q_{\texttt{read}}$ will be extracted.

Afterwards, the relayer will infer the block height $t_{\texttt{dst}}$ from the read operation $Q_{\texttt{read}}$, then proceed the retrieve the blockchain state at block height $t_{\texttt{dst}}$ on the destination chain $M_{\texttt{dst}}$. Finally, it will use the output function $f_{\texttt{output}_{\texttt{dst}}}$ to obtain the output of the destination chain $M_{\texttt{dst}}$ and get the specific view of the output requested by $Q_{\texttt{read}}$.

How does the relayer send the Status to the Source Chain?

The relayer sends the status via a call to the cross chain management contract on the source chain, which will validate it and, if real, log it, making it available to the cross chain user.

How is the Status sent by the relayer validated?

When the cross chain management contract is called by the relayer, who must provide both the status and a proof of its inclusion on the destination chain, the contract will first check the validity of the status provided using the block header $H_{\texttt{dst}}^{(t_{\texttt{dst}})}$ on the destination chain at the block height $t_{\texttt{dst}}$.

But how can a contract on the source chain retrieve the block header $H_{\texttt{dst}}^{(t_{\texttt{dst}})}$ on the destination chain? The system used to obtain such capability will be discussed in section Synchronize Block Header, for now it will just be assumed it's possible.

Cost and Incentives

A transaction fee should be paid when executing transactions in any blockchain, and given that the solution discussed involves two transactions on the source chain, one sent by the crosschain user and the other by the relayer, these must be paid for.

To prevent losses on the relayer side, the user can set a reward for relayer, which will be awarded to the first relayer that returns a valid status back. The relayer gets paid only after the correct status is sent to the source chain. Note that only the first relayer to transfer a valid status for $Q_{\texttt{read}}$ will get paid.

Read Operation Summary

This section will summarize our write operation solution and provide an overview of our cross chain interoperability process.

The `USER` in the diagram is the cross chain user. The `SRC_MANAGER` is the cross chain management contract on the source chain. The `RELAYER` is the relayer.

1. Initiate

   To initiate a cross chain read operation on source chain, `USER` sends a transaction $X_{\texttt{src}}$ on the source chain. The transaction $X_{\texttt{src}}$ must call the `SRC_MANAGER` and contain the following information:

- destination blockchain $M_{\texttt{dst}}$
- read operation $Q_{\texttt{read}}$

2. Hunt

   Only after transaction $X_{\texttt{src}}$ is executed successfully, `SRC_MANAGER` will log the following information:

   - destination blockchain $M_{\texttt{dst}}$
   - read operation $Q_{\texttt{read}}$

   `RELAYER` will then get the information by retrieving the log of `SRC_MANAGER` and proceed to the next stage.

3. Execute

   The relayer will infer the block height $t_{\texttt{dst}}$ from the read operation $Q_{\texttt{read}}$, then proceed with the retrieval of the blockchain state at block height $t_{\texttt{dst}}$ on the destination chain $M_{\texttt{dst}}$. Finally, it will use the output function $g_{\texttt{dst}}$ to obtain the output of the destination chain $M_{\texttt{dst}}$ and get the specific view of the output requested by $Q_{\texttt{read}}$.

4. Return

   The relayer sends the status via a call to the cross chain management contract on the source chain, which will validate it and, if real, log it, making it available to the cross chain user.

5. Validate

   After receiving the proof and status, `SRC_MANAGER` validates the read operation by checking the block header $H_{\texttt{src}}^{(t_{\texttt{src}})}$ on the source chain at the block height $t_{\texttt{src}}$.

Synchronize Block Header

We assumed that the block headers of both destination and source chains will be available to each other, as the verification process of the write operation involves retrieving a block header of the source chain from the destination chain, and the inverse process is performed when verifying a read operation. This section will explain how this is achieved.

A baseline solution for header synchronization is to have the source chain and the destination chain synchronize their block headers with each other. If this is done, there should be initial trust between the source chain and the destination chain at the start of the synchronization, due to the fact that the first block headers will have to be hardcoded.

But, after this initial synchronization, this solution would be mostly trustless, having only two main problems:

- As an SPV-based solution it would be vulnerable to the attack in which the validators/miners create illegal block headers, which the other chain would accept as valid. With no effect on their own chain, the attack could be launched by current validators or previous validators if the sychronization from one participant chain to another chain is not timely. The assumption of validators' kindness are not supposed to be expanded, as they only need to be responsible for their own chain instead of all the chains. Actually, the attack is a kind of fork in some sense, which may cause more serious problems in cross chain circumstances.
- If there are $n_{\mathcal{M}}$ blockchains that want to interoperate with each other, there will be $C^2_{n_{\mathcal{M}}}$ sequences of block headers in total to be synchronized in every chain, therefore adding storage, transaction and processing costs that burden the source chain and grow with O(n), where n is the number of chains that can communicate via crosschain with the source chain. The cost would be very high considering the size and number of block header of current blokcchains.

To solve these issues, we propose a solution based on a relay chain which called "Poly Chain" that will synchronize all the participant chain's block headers so that the participant chains only need to stay synchronized with the Poly Chain's header, from which the participant chain's block headers could be obtained, as they would have been collected and put in blocks by Poly Chain in advance. Poly Chain can prevent every participant chain from forking through its governance model. In addition, each node of Poly Chain is supposed to run full nodes of all participant chains to guarantee timeliness of synchronization between Poly Chain and participant chains. We proceed to fix the security issue through forcing the consensus nodes of Poly Chain to run full nodes in every blockchain supported by the crosschain protocol. That way the nodes will perform full verification on every block so that if an illegal block was to appear they would reject it. This solution allows us to lower the costs on each of the chains to just O(1) (as we only synchronize a single block header independently of the amount of chains in the system).

We formally define the output $\mathbb{O}_0$ of the Poly Chain M_0 the following way:

$$\mathbb{O}_0 = \bigcup_{i \in \mathcal{M}} \mathbb{H}_i^m$$

We use $\mathbb{H}_i$ to represent the set of block headers of each blockchain. Therefore, the output of Poly Chain is a sequence of the headers of each blockchain.

Then, given that it's possible to use the block header H to prove some parts of a blockchain's status, validation can be performed using Poly Chain's header H_0 to prove the existence of participant chain's header $H_i, i \in \mathcal{M}$. Then, we can use H_i to prove the status of the blockchain i, enabling the participant chain to do cross chain validation after synchronizing Poly Chain's header.

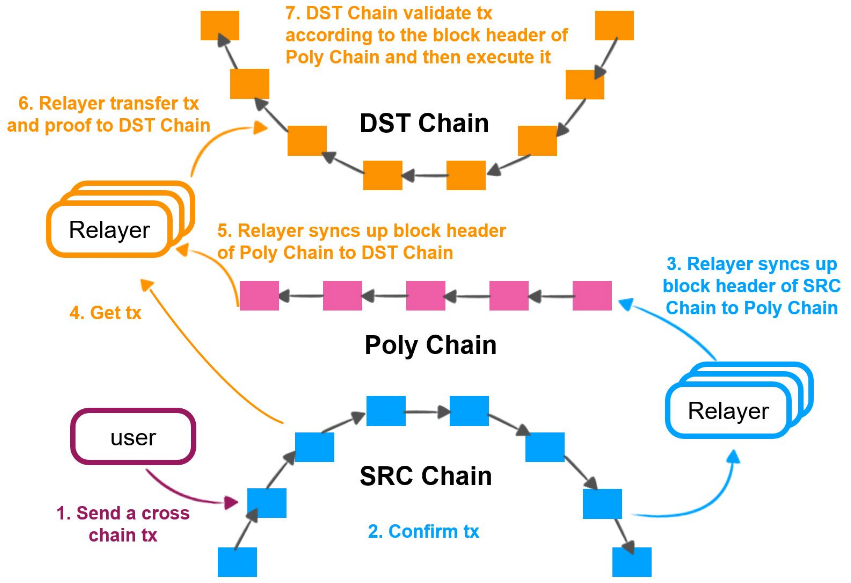
7. DST Chain validate tx according to the block header of Poly Chain and then execute it
6. Relayer transfer tx and proof to DST Chain
DST Chain
Relayer
5. Relayer syncs up block header of Poly Chain to DST Chain
3. Relayer syncs up block header of SRC Chain to Poly Chain
4. Get tx
Poly Chain
Relayer
user
SRC Chain
1. Send a cross chain tx
2. Confirm tx

Figure 3: cross chain transaction process with Poly Chain

Furthermore, Poly Chain that we proposed will have following features:

- Unlike other existing cross chain solutions like Cosmos[9] and Polkadot[10], we define Poly Chain as a tokenless consortium blockchain.
- The consensus algorithm of Poly Chain will be BFT-like consensus, therefore forks can theoretically be avoided.
- Poly Chain can prevent every participant chain from forking through its governance model. For example, it can set a challenge period and detect the any forking blockchain, in order to punish it.

Wrapping everything up, Poly Chain is the coordinator of the entire multi-chain architecture, managing the registration, change, and cancellation of participant chains. Any participant chain can submit a registration request to Poly Chain and, if the audit passes, become a member of the entire chain network system.

Further details on Poly Chain workings will be published in a future governance whitepaper.

Finally, it's possible to optimize this solution even further, which could be divided into the following two parts.

Synchronize Block Header from Participant Chain to Poly Chain

The blocks of Poly Chain could only store the hash of block headers of participant chains, and the respective block headers could be provided when a crosschain call needs them, as each node of Poly Chain extra run full nodes of participant chains. In this system, validation would be performed by getting H_0, obtaining the hash of the participant chain's header through an SPV proof, retrieving the real participant chain header H_i, confirming its authenticity by checking its hash and then applying the SPV proofs that have been discussed earlier.

Synchronize Block Header from Poly Chain to Participant Chain

Participant chains could only synchronize the key block header of Poly Chain, which records the change of validators of Poly Chain. Thus, validation could be performed by check the signature of validators of Poly Chain.

However, in case that previous validators of Poly Chain launch attack described previously in baseline solution, there should be a merkle tree composed of all the current merkle roots in the block header of Poly Chain, and the root of this merkle tree is also stored in block header of Poly Chain. Thus, the block history of Poly Chain could be one and only for all the participant chains.

Atomic Cross Chain Transactions

As the scale of business expands, the scenario of interchain information interaction becomes more complex and requires the support of atomic cross chain transactions, that is, multiple operations of a single transaction are performed between different chains with the condition that, if the execution on one of these chains fails, the transaction as a whole fails to execute. Making it so the transaction can be considered successful only if the execution of all operations on all the different chains is successful. But, how can we ensure that all chain operations succeed or fail at the same time, given that this involves transactional issues of cross chain operations?

We propose a two-phase commit protocol for atomic cross chain transactions based on cross chain write operations. This protocol is not a core part of the system presented in this paper but an extension of it, as its implementation is not required for everything else to work. Nevertheless, it provides a solution to smart contracts that need atomic cross chain transactions.

Two-Phase Contract

To implement atomic cross chain transactions, we use a two-phase commit protocol which will be implemented by the cross chain contract as two phases:

- `prepare` phase

  `prepare` phase executes atomic cross chain transactions and locks related resources in participant chains. It will return a success or failure flag to Poly Chain indicating the result of the execution. In addition, to `prepare` for indicating a success it needs to ensure that the `commit` and `rollback` phases would succeed if they were to be executed later on.

- `commit`/`rollback` phase

`commit` phase unlocks related resources and finalizes all states of an atomic cross chain transaction after receiving the commit command from Poly Chain. `rollback` phase rolls back all states of an atomic cross chain transaction after receiving the roll back command from Poly Chain.

The life cycle of an atomic crosschain transaction is as follows:

1. An atomic cross chain transaction is initiated and sent to some chains.
2. The `prepare` function in each one of these chains executes the transactions, locks related resources and sends a success/failure message to Poly Chain.
3. Poly Chain determines whether the execution is successful in every participant chain. If so, the commit command will be sent to participant chains and the `commit` function on each participant chain will submit all states of the transaction after unlocking the related resources. If not, the rollback command will be sent to participant chains and the `rollback` function of each participant chain will roll back all states of the atomic cross chain transaction in that participant chain.

An Example of a Successful Atomic Cross Chain Transaction While the proposed protocol can support more than two chains, for the sake of simplicity there will only be 2 destination chains in the following example.

- `RELAYER` is the relayer.
- `PLOY_CHAIN` is Poly Chain
- `DST_CHAIN_1` is the first destination chain
- `DST_CHAIN_2` is the second destination chain

1. An atomic cross chain transaction $X_{\mathtt{ato}}$ is initiated on source chain $M_{\mathtt{src}}$, which calls the cross chain management contract on source chain, and provides the following information. Only if $X_{\mathtt{ato}}$ is executed successfully the following data will be logged. Then such information will be retrieved by relayer.
    - destination blockchains `DST_CHAIN_1` and `DST_CHAIN_2`
    - write operations Q_1 and Q_2
    - execution costs $C_{\mathtt{execution1}}$ and $C_{\mathtt{execution2}}$
    - rewards $C_{\mathtt{reward1}}$ and $C_{\mathtt{reward2}}$
2. `RELAYER` sends a transaction on `DST_CHAIN_1` to execute the `prepare` phase of Q_1 through write operation. The execution cost and the reward are included in $C_{\mathtt{execution1}}$ and $C_{\mathtt{reward1}}$. Then the cross chain management contract on `DST_CHAIN_1` will validate the atomic cross transaction with the block header $H_{\mathtt{src}}^{(t_{\mathtt{src}})}$ on the source chain at the block height $t_{\mathtt{src}}$ and execute it.
3. The execution in `DST_CHAIN_1` is successful, and the two-phase contract's `prepare` function locks all related resources and returns success. The success of `prepare` phase of Q_1 is synchronized to Poly Chain, which

means all related resources are locked and `commit/rollback` phase will never fail.

4. `RELAYER` sends a transaction on `DST_CHAIN_2` to execute the `prepare` phase of Q_2 through write operation. The execution cost and the reward are included in $C_{\texttt{execution2}}$ and $C_{\texttt{reward2}}$. Then the cross chain management contract on `DST_CHAIN_2` will validate the atomic cross transaction with the block header $H_{\texttt{src}}^{(t_{\texttt{src}})}$ on the source chain at the block height $t_{\texttt{src}}$ and execute it.
5. The execution in `DST_CHAIN_2` is successful, and the two-phase contract's `prepare` function locks all related resources and returns success. The success of `prepare` phase of Q_2 is synchronized to Poly Chain, which means all related resources are locked and `commit/rollback` phase will never fail.
6. On Poly Chain, `RELAYER` can get the proof of that the `prepare` phases on all destination chains are succeed.
7. `RELAYER` sends a transaction with the previous proof on `DST_CHAIN_1` to execute the `commit` phase of Q_1.
8. `RELAYER` sends a transaction with the previous proof on `DST_CHAIN_2` to execute the `commit` phase of Q_2.

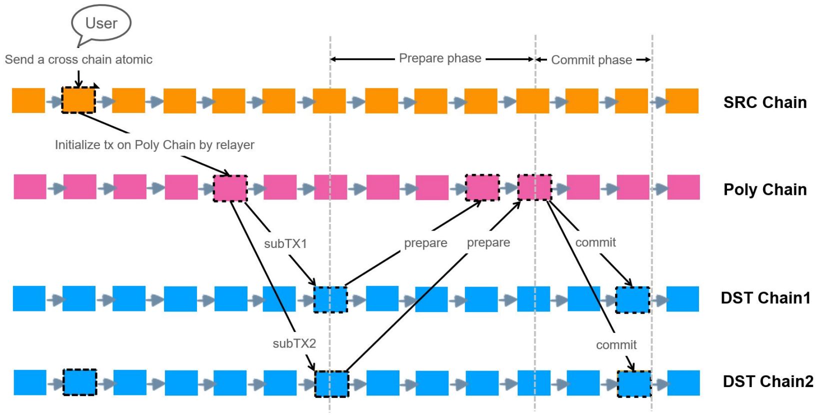

Figure 4: Example of a Successful Atomic Cross Chain Transaction Process

An Example of a Failed Atomic Cross Chain Transaction

1. Steps 1-4 are the same as in the previous example.
2. The `prepare` phase of Q_2 on `DST_CHAIN_2` fails and the fail of `prepare` phase of Q_2 is synchronized to Poly Chain.
3. On Poly Chain, `RELAYER` can get the proof of that not all `prepare` phases on destination chains are succeed.

4. RELAYER sends a transaction with the previous proof on DST_CHAIN_1 to execute the rollback phase of Q_1.
5. The execution of rollback phase of Q_1 is synchronized to Poly Chain at end.

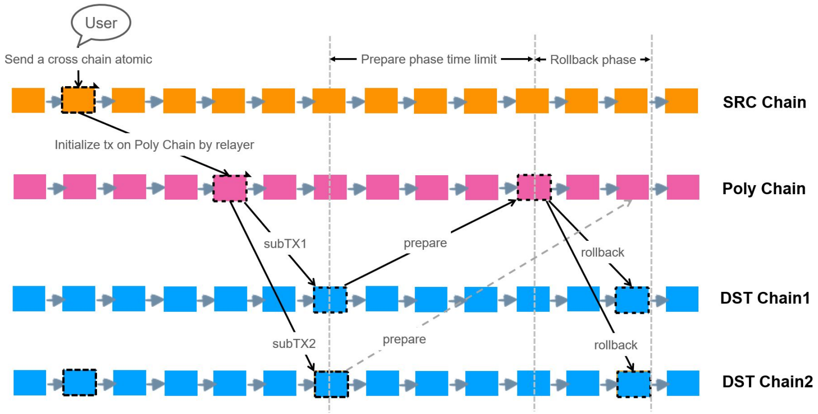

Figure 5: Example of a Failed Atomic Cross Chain Transaction Process

Poly Chain as a Coordinator

To help participant chains know whether the prepare phase is executed successfully, Poly Chain should maintain a table of prepare-phase statuses for each atomic cross chain transaction and provide two functions to notify whether the prepare phase is executed successfully on a participant chain: function success $f_{\texttt{success}}$ and function fail $f_{\texttt{success}}$. We use $\mathbf{\Lambda}$ to represent the data stored in Poly Chain for an atomic cross chain transaction, data that will be modeled as a n-length vector of -1, 0, and 1 where n is the number of destination chains and -1, 0, 1 represent successful, unknown, failed respectively.

$$\mathbf{\Lambda} \in \{-1, 0, 1\}^n$$

We use $\mathbf{0}^n$ to denote a n-length vector populated by 0, $\mathbf{1}^{(k)}$ to denote a vector in which the k-th item is 1 and all the other items are 0 and $-\mathbf{1}^{(k)}$ to denote a vector in which the k-th item is -1 and all other items are 0.

The two functions mentioned beforehand obey the following rules:

- success

  When the prepare phase runs successfully on blockchain M_k, it can tell Poly Chain that the prepare phase is executed successfully on blockchain M_k using this function $f_{\texttt{success}}$.

$$f_{\texttt{success}}(\mathbf{\Lambda}, k) = \begin{cases} \texttt{map}_{\texttt{max}}(\mathbf{\Lambda}, \mathbf{1}^{(k)}) & \texttt{min}(\mathbf{\Lambda}) > -1 \\ \mathbf{\Lambda} & \texttt{else} \end{cases}$$

- `fail`

  When the `prepare` phase fails on blockchain M_k or the `rollback` phase is executed, the function $f_{\texttt{fail}}$ will be called.

$$f_{\texttt{fail}}(\mathbf{\Lambda}, k) = \begin{cases} \texttt{map}_{\texttt{min}}(\mathbf{\Lambda}, -\mathbf{1}^{(k)}) & \texttt{min}(\mathbf{\Lambda}) < 1 \\ \mathbf{\Lambda} & \texttt{else} \end{cases}$$

$\mathbf{\Lambda}$ is $\mathbf{0}^n$ initially. The function `success` $f_{\texttt{success}}$ and function `fail` $f_{\texttt{success}}$ can update the data $\mathbf{\Lambda}$, which means the next state of $\mathbf{\Lambda}$ can be the result of $f_{\texttt{success}}$ or $f_{\texttt{fail}}$:

$$\mathbf{\Lambda}^{(t+1)} = \begin{cases} f_{\texttt{success}}(\mathbf{\Lambda}^{(t)}, k) \\ f_{\texttt{fail}}(\mathbf{\Lambda}^{(t)}, k) \end{cases}$$

When the atomic cross chain transaction is initially launched on the source chain, the relayer helps transfer this transaction to multiple destination chains and Poly Chain. Then, Poly Chain creates a new $\mathbf{\Lambda} = \mathbf{0}^n$, which will be modified after the `prepare` phase is executed in the multiple destination chains, whether the `prepare` phase has been executed successfully in each destination chain can be synchronized to Poly Chain. Afterwards, if all the `prepare` phases have been executed successfully, the changes will be committed gradually in the multiple destination chains by the `commit` phase. If one of the `prepare` phases ends up in failure, then the relayers will help all the destination chains execute the `rollback` phase.

The same incentives for relayers explained in previous sections are applied directly when performing atomic crosschain transactions.

Optimization

Some optimization strategies are introduced below, which could be alternatives for the final solution or could be combined with further optimization.

intermediate contract

Who will act as the Smart Contract Caller on the Destination Chain when performing cross chain calls?

A naive solution, illustrated in the following diagram, would be to make the relayer be the caller of the cross chain management contract, with the caller of the real contract being the cross chain management contract.

The problem with this solution is that all the cross chain users will share the same caller, which is insecure. There are at least 3 solutions to this problem:

1. Have the destination chain support proxy calls natively, which would allow for the caller, sender and even payer of a transaction to be different. While this may be a perfect solution, there are almost no chains that support this currently.
2. Publish a standard that prohibits the use of the contract caller as an authentication mechanism in smart contracts. This solution is not friendly nor compatible with most smart contracts.
3. Have the cross chain management contract create a intermediate contract for each cross chain user and then use the intermediate contract to call the real contract.

We recommand the third solution, the solution involves having the cross chain management contract create an intermediate contract that acts as an account for each cross chain user. Then, each write operation will use the cross chain user's own intermediate contract to perform the call, essentially turning the contract into a proxy of the cross chain user.

Time Out

To improve the operation efficiency of cross chain transactions and ensure token liquidity, the time out feature, which could be viewed as execution deadline, on cross chain transactions set by cross chain users is proposed. This feature could especially benefit cross chain users, for example, in the previous solution without time out feature, after initiating cross chain transactions, cross chain users need to wait until relayers execute them, which may cause long lead time for cross chain execution and higher uncertainty for locked rewards in the cross chain contract. The details of time out feature in different cross chain operations are illustrated below:

- Write Operation

  Cross chain users could set a time out feature when issuing a new cross chain write operation. This time out feature could be monitored and determined by the specific block height on the destination chain: if the current block height is lower than the time out, this cross chain write operation could still be executed; while the block height is higher than the time out, the cross chain management contract on the destination chain will raise an error and this cross chain write operation will be expired and no further execution would be permitted.

- Read Operation

  Cross chain user could set a time out when initiating the cross chain read operation, similar to the write operation, this time out feature could be monitored and determined by the specific block height on the source chain: if the current block height is lower than the time out, this cross chain read operation could still be executed; if higher than the time out, the cross chain management contract on the source chain will raise an error.

- Atomic Cross Chain Transaction

  Cross chain users could set a time out when initiating atomic cross chain transactions, too. This time out feature could be monitored and determined by the specific block height on Poly Chain. If the current block height is lower than the time out, only function `fail` can be called on Poly Chain.

Relayer Protection

There may exist some risks for relayers of the current solution.

Firstly, based on "the first relayer is the winner" rule, block proposer[6], which is one of consensus nodes to generate and send new blocks, could cheat on relayers by copying cross chain transactions submitted by relayers with a new address and rank their transactions ahead of relayers. In this situation, it is hard to prove who is the first executor and whether consensus nodes change the order of blocks or not, so relayers may lose transaction fees and get no reward. To reduce this risk, the optional solution is proposed here: when a relayer identifies a cross chain transaction opportunity on the source chain, he could deposit some tokens, the amount and the type of which will be specified by cross chain users.

The deposit information will be logged and the cross chain management contract can only allow the relayer who is bound to transfer the cross chain operation. Other relayers will stop executing this cross chain transaction. This method could largely reduce relayers' transaction fee losses of not being the winner. If someone proves that the cross chain operation is not confirmed before time out, all of the locked token can be used to compensate the cross chain user.

Secondly, even if adding the time out feature, the block proposer could still cheat on relayers by refusing to proceed cross chain transactions until the time out expires. Since it is hard to prove whether relayers should be blamed or the consensus nodes should be blamed, relayers may finally need to suffer from transaction fee losses. When the relayer promises to execute this cross chain operation, he should carefully evaluate and be responsible for all potential risks. Actually it's not a issue only for cross chain but also blockchain transactions because common transactions. (e.g. assets transfer and contract call maybe delayed by the block proposers, too.) However, block proposers are replaced frequently in most blockchains (even in each block). If one of the block proposers is upright, the cross chain operation will be transferred correctly.

Analysis

The analysis of our solution is divided in several parts as following including safety, latency, TPS and cost.

Safety

Safety properties informally require that "something bad will never happen"[26][27][28].

The premise of our discussion of safety is that the participant chains are functioning properly. Otherwise, even if the cross chain solution provides the correct information, the information may still be tampered with or discarded due to abnormal operation of the participant chain.

We mentioned in the previous sections that having a trusted Poly Chain and the governance mechanism of Poly Chain provides us non-fork participant chains. In our solution, the status in the output of Poly Chain can be hash of the header of participant chain $\mathtt{hash}(H_{\mathtt{participant}})$ and the key header $K_{\mathtt{0}}$ of Poly Chain is stored in the participant chain which can be obtained by the cross chain management contract.

The following proof can ensure that a status of a participant chain on another participant chain is valid.

$$O_{\mathtt{status}} \leftarrow H_{\mathtt{participant}} \leftarrow \mathtt{hash}(H_{\mathtt{participant}}) \leftarrow H_{\mathtt{0}} \leftarrow K_{\mathtt{0}}$$

- $O_{\mathtt{status}} \leftarrow H_{\mathtt{participant}}$: header can prove the status (SPV Proof)[18].
- $H_{\mathtt{participant}} \leftarrow \mathtt{hash}(H_{\mathtt{participant}})$: A cryptographic hash function allows one to easily verify whether some input data map onto a given hash value, but if the input data is unknown it is extremely difficult to reconstruct it (or any equivalent alternatives) by knowing the stored hash value[29][30] .
- $\mathtt{hash}(H_{\mathtt{participant}}) \leftarrow H_{\mathtt{0}}$: header can prove the status and the status on Poly Chain is the header on each participant chain (SPV Proof)[18].
- $H_{\mathtt{0}} \leftarrow K_{\mathtt{0}}$: common headers can be verified by the signature of the majority* of validators mentioned in the key header[31].

Safety in Synchronizing Block Header

Safety in synchronizing block chain means that respective block header could be synchronized to participant chains and Poly Chain correctly.

For synchronizing block header from participant chain to Poly Chain, as each node of Poly Chain runs extra full node of all participant chains, they could synchronize block header of participant chain and store hash of block header in Poly Chain correctly and in time.

For synchronizing block header from Poly Chain to participant chain, nodes of Poly Chain are supposed to synchronize key block header of Poly Chain to participant chain in time, which records the change of validators of Poly Chain. Actually the merkle root of previous block headers on Poly Chain is written in key block header which avoid long range attacks[32].

Safety of Write Operation

Safety of write operation means that forged cross chain write operation would never be executed on the destination chain by relayers and fake relayers can never get the reward of write operation.

Actually all the cross chain write operations will be validated before execution. The cross chain management contract will ensure that the write operation $Q_{\texttt{write}}$ has been confirmed successfully on the source chain because the write operation $Q_{\texttt{write}}$ is a status in output of the source chain and it can be verified by the destination chain. Then the forged write operation will be denied by the cross chain management contract.

Similarly, the relayer will be paid only if their transaction $X_{\texttt{dst}}$ is confirmed on the destination chain. Actually the block $H_{\texttt{dst}}$ header can also help to validate the state of a blockchain and the $X_{\texttt{dst}}$ can be validated.

Safety of Read Operation

Safety of read operation means that forged status cannot be returned back to the cross chain user and only if the relayer returned the correct status, the relayer can get the reward of read operation. Actually all the status can be verified by the source chain by the block header that we discussed before and the cross chain management contract ensures that the relayer gets paid only after the status is verified.

Safety of Unrelated Participant Chain

Safety of unrelated participant chain means that result of a transaction between source chain and destination chain wouldn't have effect on other participant chains, a.k.a. unrelated participant chain.

In our solution, Poly Chain is supposed to help participant chains synchronize block header, and transactions between source chain and destination chain are handled just by these two chain. Thus, unrelated participant chains wouldn't be affected.

Latency

We use latency to evaluate the delay from a cross chain user to initiate a cross chain operation to the cross chain operation being executed.

We can not ensure that any cross chain operation would eventually be executed in a static finite time because whether the operation is finished depends on the relayer. Incentives can increase the enthusiasm of the relayer to speed up trading. The more rewards for the relayer, the lower latency of the cross chain transactions. It is a kind of cross chain transaction fee (just like the transaction fee in many blockchains, which can promote the transaction to be packaged into the block by the consensus node).

Actually there may be some optimized solution to improve the latency of the cross chain solution in the future work.

Write Operation Latency

To discuss the latency of write operation $d_{\mathtt{write}}$, we first use the following notations:

- $d_{\mathtt{write},\mathtt{src}}$

  delay from the transaction $X_{\mathtt{src}}$ being sent on the source chain by user to the transaction $X_{\mathtt{src}}$ being confirmed at the block $I_{\mathtt{src}}^{(t_{\mathtt{src}})}$ on the source chain

- $d_{\mathtt{sync},\mathtt{write}}$

  delay from the transaction $X_{\mathtt{src}}$ being confirmed at the block $I_{\mathtt{src}}^{(t_{\mathtt{src}})}$ on the source chain to enough information for the proof being synchronized.

- $d_{\mathtt{write},\mathtt{relayer}}$

  delay from the transaction $X_{\mathtt{src}}$ being confirmed on the source chain to the transaction $X_{\mathtt{dst}}$ that including the write operation $Q_{\mathtt{write}}$ being sent on the destination chain by relayer

- $d_{\mathtt{write},\mathtt{dst}}$

  delay from transaction $X_{\mathtt{dst}}$ being sent on the destination chain by relayer to the transaction $X_{\mathtt{dst}}$ being confirmed at the block $I_{\mathtt{dst}}^{(t_{\mathtt{dst}})}$ on Poly Chain

The formula of latency of write operation being executed $d_{\mathtt{write}}$ could be described as

$$d_{\mathtt{write}} = d_{\mathtt{write},\mathtt{src}} + \mathtt{max}\begin{cases} d_{\mathtt{sync},\mathtt{write}} \\ d_{\mathtt{write},\mathtt{relayer}} + d_{\mathtt{write},\mathtt{dst}} \end{cases}$$

Actually we have the following constraint, otherwise the validation must be regarded as a failure.

$$d_{\mathtt{write},\mathtt{relayer}} + d_{\mathtt{write},\mathtt{dst}} \geqslant d_{\mathtt{sync},\mathtt{write}}$$

Then

$$d_{\mathtt{write}} = d_{\mathtt{write},\mathtt{src}} + d_{\mathtt{write},\mathtt{relayer}} + d_{\mathtt{write},\mathtt{dst}}$$

- The transaction $X_{\mathtt{src}}$ may be executed and confirmed within a very short period if the time of it being sent is close to generation of next block ($d_{\mathtt{write},\mathtt{src}}$ could be close to zero). However, the transaction may also wait for a lone time if source chain is in congestion.

$$d_{\texttt{write,src}} > 0$$

- If validators of Poly Chain don't change and they synchronize hash of block header of source chain once the transaction $X_{\texttt{src}}$ is confirmed, $d_{\texttt{sync,write}}$ could be close to zero. However, if validators of Poly Chain change and sychronization of key block header is delayed because of some reasons such as congestion of destination chain, $d_{\texttt{sync,write}}$ could be very high. Thus, the range of $d_{\texttt{sync,write}}$ is greater than zero:

$$d_{\texttt{sync,write}} > 0$$

- If relayer detects the transaction $X_{\texttt{src}}$ once it is confirmed on the source chain and sends the transaction $X_{\texttt{dst}}$ immediately, $d_{\texttt{write,relayer}}$ could be close to zero. If the transaction $X_{\texttt{src}}$ is not detected timely or its reward is not high enough, respective $d_{\texttt{write,relayer}}$ could be very high. Thus, the range of $d_{\texttt{write,relayer}}$ is greater than zero:

$$d_{\texttt{write,relayer}} > 0$$

- Similar to $d_{\texttt{write,src}}$, the range of $d_{\texttt{write,dst}} > 0$ is also greater than zero:

$$d_{\texttt{write,dst}} > 0$$

Read Operation Latency

To discuss the latency of read operation $d_{\texttt{read}}$, we first use the following notations:

- $d_{\texttt{read,status}}$

  delay from the transaction $X_{\texttt{src}}$ being sent on the source chain by user to the block $I_{\texttt{dst}}^{(t_{\texttt{dst}})}$ on the destination chain specified by the read operation $Q_{\texttt{read}}$ being confirmed.

- $d_{\texttt{sync,read}}$

  delay from the transaction on the block $I_{\texttt{dst}}^{(t_{\texttt{dst}})}$ being confirmed on the destination chain to fetch enough information for the proof being synchronized.

- $d_{\texttt{read,src}}$

  delay from the transaction $X_{\texttt{src}}$ being sent on the source chain by user to the transaction $X_{\texttt{src}}$ being confirmed at the block $I_{\texttt{src}}^{(t_{\texttt{src}})}$ on the source chain

- $d_{\texttt{read,relayer}}$

  delay from the transaction $X_{\texttt{src}}$ being confirmed on the destination chain to the transaction $X_{\texttt{return}}$ that includes the result of read operation $Q_{\texttt{read}}$ being sent on the source chain by relayer

- $d_{\texttt{read,return}}$

  delay from transaction $X_{\texttt{return}}$ being sent on the source chain by relayer to the transaction $X_{\texttt{return}}$ being confirmed at the block $I_{\texttt{src}}^{(t_{\texttt{return}})}$ on Poly Chain

The formula of latency of read operation being executed $d_{\texttt{read}}$ could be described as

$$d_{\texttt{read}} = \texttt{max}\begin{cases} d_{\texttt{read,status}} + d_{\texttt{sync,read}} \\ d_{\texttt{read,src}} + d_{\texttt{read,relayer}} + d_{\texttt{read,return}} \end{cases}$$

Actually we have the following constraint, otherwise the validation must be considered a failure.

$$d_{\texttt{read,src}} + d_{\texttt{read,relayer}} + d_{\texttt{read,return}} \geqslant d_{\texttt{read,status}} + d_{\texttt{sync,read}}$$

Then

$$d_{\texttt{write}} = d_{\texttt{read,src}} + d_{\texttt{read,relayer}} + d_{\texttt{read,return}}$$

- Similar with write operation, the $d_{\texttt{read,src}}$ is larger than 0.

$$d_{\texttt{read,src}} > 0$$

- If the cross chain user wants to get an exist status of the destination chain, $d_{\texttt{read,status}}$ could be 0. If the cross chain user wants to get a status in the future, the delay depends on how many blocks need to wait $t_Q - t_{\texttt{dst}}$ and the block generation rate $r_{\texttt{block/time,dst}}$

$$d_{\texttt{read,status}} = \texttt{max}\begin{cases} \frac{(t_Q - t_{\texttt{dst}})}{r_{\texttt{block/time,dst}}} \\ 0 \end{cases} \geqslant 0$$

- If the cross chain user wants to get an exist status of the destination chain, $d_{\texttt{sync,read}}$ could be 0. If the cross chain user want to get a status in the future, it needs some time to synchronize the block headers

$$d_{\texttt{sync,read}} \geqslant 0$$

- Similar to write operation. $d_{\texttt{read,relayer}}$ is greater than 0

$$d_{\texttt{read,relayer}} > 0$$

- Similar to write operation. $d_{\texttt{read,return}}$ is greater than 0

$$d_{\texttt{read,return}} > 0$$

TPS

The cross chain TPS will be discussed in this section.

Poly Chain's TPS could be expressed by the following notations and equation. (Actually the TPS of Poly Chain would be very high). Firstly, the TPS of Poly Chain should be larger than $\sum_i r_{\texttt{block/time}_i}$ to maintain smooth operation since $\sum_i r_{\texttt{block/time}_i}$ represents all blocks created per second on participant chain that should be synchronized on Poly Chain. Secondly, there is a positive correlation between Poly Chain's TPS $r_{\texttt{tx/time}_0}$ and participant chain's TPS $r_{\texttt{tx/time}_i}$ for $\forall i \in \mathcal{M}$, which means higher TPS on participant chain tends to require higher TPS on Poly Chain. Thirdly, there is a negative correlation between Poly Chain's TPS $r_{\texttt{tx/time}_0}$ and participant chain's transactions per block $r_{\texttt{tx/block}_i}$ for $\forall i \in \mathcal{M}$, which means for the certain amount of transactions waiting to be synchronized on participant chain, if there are more transactions per block ψ_i, the number of block headers to be synchronized will be less, so the requirement for Poly Chain's TPS will be lower.

$$r_{\texttt{tx/time}_0} > \sum_i r_{\texttt{block/time}_i} = \sum_i \frac{r_{\texttt{tx/time}_i}}{r_{\texttt{tx/block}_i}}$$

The participant chain's TPS could be expressed by the following notations and equation. Firstly, the TPS of participant chain should be larger than $r_{\texttt{key/time}_0}$, which represents all key blocks created per second on Poly Chain that should be synchronized on participant chain. Actually $r_{\texttt{key/time}_0}$ is low because the validators on Poly Chain don't change frequently. So our cross chain solution has almost no requirements for TPS of participant chains.

for $\forall i \in \mathcal{M}$:

$$r_{\texttt{tx/time}_i} > r_{\texttt{key/time}_0}$$

Cost

Finally, we discuss about the cost of cross chain operations including the cost on Poly Chain, the cost on participant chain, the cost of relayer and the cost of cross chain user.

Cost on Poly Chain

This section will discuss the cost of Poly Chain.

Let $C_{\texttt{0}}$ be the total cost of Poly Chain. Assume there are $n_{\texttt{0}}$ validators on Poly Chain and each validator spends $C_{\texttt{0,validator}}$ on operation and maintenance on average. Therefore, the equation of Poly Chain's total cost will be:

$$C_{\texttt{0}} = n_{\texttt{0}} C_{\texttt{0,validator}}$$

Narrowing down the average cost for each validator, as every validator needs to run both consensus nodes and run all participant chains' sync nodes, $C_{\texttt{0,validator}}$ could be calculated by the average cost of running a consensus node plus the sum of the cost of running sync nodes of all participant chains:

$$C_{\texttt{0,validator}} = C_{\texttt{node,consensus}_{\texttt{0}}} + \sum_{i \in \mathcal{M}} C_{\texttt{node,sync}_i}$$

To simplify the model, it is assumed the average cost of running a node $C_{\texttt{node}}$, no matter for a consensus node or a sync node, only take the computing ($C_{\texttt{compute}}$), storage ($C_{\texttt{storage}}$), and network ($C_{\texttt{network}}$) fees into consideration.

$$C_{\texttt{node}} = C_{\texttt{compute}} + C_{\texttt{storage}} + C_{\texttt{network}}$$

The details of the three types of cost will be discussed in the following:

1. For the computational capability, let $C_{\texttt{compute,consensus}}$ be the total computational cost for a consensus node and $C_{\texttt{compute,sync}}$ be the total computational cost for a sync node. The requirement for running a consensus node on Poly Chain will usually be higher for running a sync node.

$$C_{\texttt{compute,consensus}} \gg C_{\texttt{compute,sync}}$$

2. For the storage cost, $C_{\texttt{storage}}$ equals to unit storage fee $C_{\texttt{storage,price}}$ times the incremental storage size in unit period $l_{\texttt{block}} r_{\texttt{block/time}}$, where $l_{\texttt{block}}$ represents the block size on average and $r_{\texttt{block/time}}$ represents the block generation rate on average. The complete equation will be:

$$C_{\texttt{storage}} = C_{\texttt{storage,price}} l_{\texttt{block}} r_{\texttt{block/time}}$$

3. For the network cost, $C_{\texttt{network}}$ is assumed to be proportional to $C_{\texttt{storage}}$ with the ratio of $r_{\texttt{network/storage}}$. The ratio for consensus node is $r_{\texttt{network/storage,consensus}}$ and ratio for sync node is $r_{\texttt{network/storage,sync}}$. The $r_{\texttt{network/storage,sync}}$ could be approximately seen as 1 since sync nodes only do synchronization. Besides, based on BFT consensus algorithm,

the $r_{\texttt{network/storage,consensus}}$ will be generally larger than $r_{\texttt{network/storage,sync}}$ since validators need to do synchronization, consensus, and broadcasting. Then the estimated equation will be:

$$C_{\texttt{network}} \approx r_{\texttt{network/storage}} C_{\texttt{network,price}} l_{\texttt{block}} r_{\texttt{block/time}}$$

$$r_{\texttt{network/storage,consensus}} \gg r_{\texttt{network/storage,sync}} \approx 1$$

Narrowing down the calculation of storage size on Poly Chain, there are three main components: the storage of Poly Chain's block header, the storage of all participant chains' block header hash, and the storage of atomic cross chain transactions. On one hand, blocks, as well as block headers, are generated at a constant rate $r_{\texttt{block/time}_{\texttt{0}}}$ on Poly Chain, $l_{\texttt{header0}} r_{\texttt{block/time}_{\texttt{0}}}$ will represent the storage of Poly Chain's block header. On the other hand, the body sizes of blocks on Poly Chain are positively correlated to the block generation rate of participant chains $r_{\texttt{block/time}_i}$, so $\sum_{i\in\mathcal{M}} l_{\texttt{hash}} r_{\texttt{block/time}_i}$ will represent the storage of participant chains' block header hash on Poly Chain. Therefore, the equation of the incremental storage size on Poly Chain will be:

$$l_{\texttt{block0}} r_{\texttt{block/time}_{\texttt{0}}} = l_{\texttt{header0}} r_{\texttt{block/time}_{\texttt{0}}} + \sum_{i\in\mathcal{M}} l_{\texttt{hash}} r_{\texttt{block/time}_i} + l_{\texttt{atomic}} r_{\texttt{atomic}}$$

Therefore, by combining all computational, storage and network cost calculated above, the summarized equation of the average cost for each validator $C_{\texttt{0,validator}}$ will be:

$$C_{\texttt{0,validator}} = \texttt{sum}\begin{cases} C_{\texttt{compute,consensus}_{\texttt{0}}} \\ C_{\texttt{storage,price}}(\texttt{sum}\begin{cases} l_{\texttt{header0}} r_{\texttt{block/time}_{\texttt{0}}} \\ \sum_{i\in\mathcal{M}} l_{\texttt{hash}} r_{\texttt{block/time}_i} \\ l_{\texttt{atomic}} r_{\texttt{atomic}} \end{cases}) \\ r_{\texttt{network/storage}_{\texttt{0}}} C_{\texttt{network,price}}(\texttt{sum}\begin{cases} l_{\texttt{header0}} r_{\texttt{block/time}_{\texttt{0}}} \\ \sum_{i\in\mathcal{M}} l_{\texttt{hash}} r_{\texttt{block/time}_i} \\ l_{\texttt{atomic}} r_{\texttt{atomic}} \end{cases}) \\ \sum_{i\in\mathcal{M}} C_{\texttt{compute,sync}_i} \\ \sum_{i\in\mathcal{M}} C_{\texttt{storage,price}} l_{\texttt{block}i} r_{\texttt{block/time}_i} \\ \sum_{i\in\mathcal{M}} r_{\texttt{network/storage}_i} C_{\texttt{network,price}} l_{\texttt{block}i} r_{\texttt{block/time}_i} \end{cases}$$

Currently we are going to use specific values to estimate the cost on Poly Chain. We suppose there are 7 nodes ($|n_{\texttt{0}}| = 7$) and 10 participant chains ($|\mathcal{M}| = 10$). We estimate the market price and get the unit price of the computing, storage and network ($C_{\texttt{compute,consensus}_0} = 10^{-4}\texttt{\$/s}$, $C_{\texttt{storage,price}} = 5 \times 10^{-12}\texttt{\$/b}$, $C_{\texttt{network,price}} = 5 \times 10^{-12}\texttt{\$/b}$ and for $\forall i \in \mathcal{M}$, $C_{\texttt{compute,sync}_i} = 10^{-5}\texttt{\$/s}$). According to the average block generation rate and the size of block header by BFT algorithm, we estimate some following properties for Poly Chain ($l_{\texttt{header0}} = 4\times 10^3\texttt{b/block}$, $r_{\texttt{block/time}_0} = 1\texttt{block/s}$). We also estimate correlative properties for each participant chain(for $\forall i \in \mathcal{M}$, $r_{\texttt{block/time}_i} = 1\texttt{block/s}$ and $l_{\texttt{block}i} = 10^5\texttt{b/block}$). We use SHA-256 to get the hash of each block header ($l_{\texttt{hash}} = 256\texttt{b/block}$). We assume the size for each atomic transaction is 16 bits ($l_{\texttt{atomic}} = 16\texttt{b/tx}$ and $r_{\texttt{atomic}} = 1\texttt{tx/s}$). And we assume the ratio between the consumption of network and the consumption of storage on Poly Chain is 100 because of BFT consensus cost and the ratio between the consumption of network and the consumption of storage on participant chain is 1 ($r_{\texttt{network/storage}_0} = 100$ and $r_{\texttt{network/storage}_i} = 1$).

As a result, we estimate the following cost for validators in Poly Chain

$$C_{\texttt{0,validator}} = 2.1 \times 10^{-4}\texttt{\$/s} = 6.6 \times 10^3\texttt{\$/year}$$

Cost on Participant Chain

Participant chain would synchronize key block header of Poly Chain, in which the change of validators of Poly Chain is recorded, whose frequency of generation is low. Thus, the cost of participant chain is very small.

Cost of Relayer

Relayers are like miners under PoW consensus algorithm. Firstly, relayers get rewards by doing cross chain write and read operations. Secondly, relayers need to compete with each other since only the relayer whose operation is firstly completed and verified would be rewarded. Thirdly, relayers need to bear some costs to conduct cross chain write and read operations.

This section will discuss about the cost of relayer.

The cost of relayers comes from two aspects: the first one is running nodes. From our mechanism, relayers need to run nodes of participant chains, destination chains and Poly Chain obviously.

Another part is cross chain transaction fee. If relayer want to get the reward set by cross chain user, they need pay for the cross chain transaction in participant chains. We will discuss in detail below

Running Nodes Relayer needs to run the sync node of Poly Chain and participant chains that relayer would like to help handle cross chain transactions(i.e. respective participant chains of relayer).

Let $C_{\texttt{relayer,nodes}}$ be relayer's cost of running nodes, $\mathcal{M}_{\texttt{relayer}}$ be respective participant chains of relayer. Thus, relayer' cost of running nodes could be sum of the cost of running sync nodes of Poly Chain and respective participant chains:

$$C_{\texttt{relayer,nodes}} = \sum_{i \in \mathcal{M}_{\texttt{relayer}} \cup \{\texttt{0}\}} C_{\texttt{node,sync}_i}$$

Transaction Fee Relayers need to send transactions which gives rise to transaction fee. In this section, we will talk about the transaction fee of relayers.

1. Write Operation

The transaction fee of write operation for relayers is generated in calling cross chain management contract which is also called execution cost in sections above, and getting reward from cross chain management contract in source chain or destination chain, which can be called demand fee.

The charge unit of execution cost is the same as destination chain's transaction fee. In addition, if the cross chain management contract is a native contract, the execution cost will be determined according to the rules of the native contract, which is cheap for the relief mechanism of transaction fee. Otherwise, we can estimate the execution cost $C_{\texttt{execution}}$ as follows:

- $C_{Q_{\texttt{write}}}$ is the cost that should be spent executing the write operations in destination chains
- $C_{\texttt{validation,write}}$ is the cost of cross chain management contracts to verify the legitimacy of cross chain status

$$C_{\texttt{execution}} = C_{Q_{\texttt{write}}} + C_{\texttt{validation,write}}$$

The charge unit of demand fee depends on the participant chain where relayers get a reward. In addition, if the cross chain management contract is a native contract, the execution cost will be determined according to the rules of the native contract, which is cheap for the relief mechanism of transaction fee. Otherwise, we can estimate the demand fee $C_{\texttt{getreward}}$ as follows:

- $C_{\texttt{payreward}}$ is a normal asset transfer fee and usually very low
- $C_{\texttt{validation,getreward}}$ is the validation cost of cross chain operations. The validation cost can be less if relayers get their reward on the destination chain instead of source chain, because to validate the success execution for source chain is another cross chain read operation.

$$C_{\texttt{demand}} = C_{\texttt{demand}} + C_{\texttt{validation,demand}}$$

2. Read Operation

Unlike the transaction fee of write operation, relayers only need to get the status in destination for almost free, then execute a transaction in management contract of source chain with the status and proof. Relayers can only get the reward if the transaction is success. Therefore, relayer need to evaluate the transaction fee and provide the right status.

The charge unit of read fee should be same as source chain's transaction fee. In addition, if the cross chain management contract is a native contract, the execution cost will be determined according to the rules of the native contract, which is cheap for the relief mechanism of transaction fee. Otherwise, we can estimate the read fee $C_{\texttt{read}}$ as follows:

- $C_{Q_{\texttt{read}}}$ is the cost of returning status to source chain and usually depends on the size of the status
- $C_{\texttt{validation,read}}$ is the validation cost of read operation. To be specific, relayers call the cross chain management contract to validate status and get reward which will generate transaction fee

$$C_{\texttt{read}} = C_{Q_{\texttt{read}}} + C_{\texttt{validation,read}}$$

The cross chain validation cost $C_{\texttt{validation}}$ can be estimated by 2 SPV proof cost and 1 hash cost and 1 signature check cost, which is about `0.1 GAS` in `NEO`.

Cost of Cross Chain User

The cost of cross chain user consists of two parts: one is the transaction fee of $X_{\texttt{src}}$, the other is the reward $C_{\texttt{reward}}$.

In order to get their cross chain transactions be executed in time and smoothly, except for some special situations that some relayers are willing to serve some users without benefits, cross chain users need to set reasonable and even attractive rewards, covering both transaction costs and premium incentives, for relayers.

In write operation, cross chain users need to at least cover relayer's transaction fees of calling cross chain management contract plus fees of getting rewards, that is:

$$C_{\texttt{reward,write}} > C_{\texttt{execution}} + C_{\texttt{demand}}$$

In read operation, cross chain users need to at least pay for relayer's transaction fees of doing read operation, that is:

$$C_{\texttt{reward,read}} > C_{\texttt{read}}$$

In addition, cross chain users also need to consider the physical cost of the relayer including computing cost, storage cost, network cost and so on.

Dynamic token exchange rates should be taken into consideration when evaluating cross chain transactions especially in write operation, because relayers need to spend destination chain tokens and be rewarded by other tokens.

Application

Decentralized Exchange

The current situation of the encrypted digital currency market is that centralized exchange has become the gateway to the largest amount of traffic, and thus commands the power of discourse in the world of blockchain. These centralized trading platforms have proven unlikely to be able to deal with certain special events such as hard forks, thereby giving rise to high policy risks. The goal of Decentralized Trading Platform (also known as DEX) is to solve the problem of centralized architecture by establishing a peer-to-peer market directly on the blockchain. It allows users to continue to monitor their funds with a unique transaction technology called "atomic swap". Users can implement cryptocurrency transactions without exposing their cryptographic assets from the security of their private wallets. Most popular decentralized exchange applications currently focus on token transactions only on one chain, which greatly reduces the potential application scenarios and the richness of user experience.

Our two-phase protocol enables cross-chain atomic transactions by introducing two phases. In the two-phase protocol, the system consists of three types of roles: one is Poly Chain, of which there is only one in the system. The second is participant chain. There are a lot of participant chains. In the DEX scenario, one transaction generally involves two participant chains. The third class is relayer, which is responsible for passing messages between the chains. Each participant chain has multiple relayers. The execution process is as follows:

The first phase: prepare stage. Poly Chain prompts the participant chain to prepare to commit or rollback the transaction and then enters the decision process. During the decision process, the participant chain will keep Poly Chain updated of its status, success or failure.

The second phase: commit/rollback phase. Poly Chain will make a decision, commit or rollback based on the results of the first phase. If and only if all the participate chains prepare successfully, then Poly Chain prompts all participant chains to commit the transaction, otherwise Poly Chain prompts all participate chains to rollback the transaction. The participant chains will perform the corresponding operation after receiving the message from Poly Chain.

The first phase is used to ensure that the transaction can be executed, and the second phase guarantees the consistency of the transaction.

Cross-chain Contract Call

Sometimes we will encounter a situation such that different modules of a DAPP may be deployed on different chains. How does a user from a chain call modules on other chains? And how do different modules interact if chain A needs to know the result of chain B to proceed to the next step?

Our solution distinguishes the cross-chain read operations and cross-chain write operations. Users can get the status of the destination chain at a specific block height from the source chain by using a read operation. Users can send a transaction to the destination chain (transfer assets, call a contract or some other operations) by initialing a write operation. The combination of read and write operations enables interaction between cross-chain contracts.

Different from some current popular cross-chain solutions, our solution does not require the participant chain to follow the unified cross-chain protocol standard at the initial design level of the underlying platform. Of course, the unified cross-chain protocol standard can be used to constrain the future chain. But for many existing public chains, it is impractical to make changes in the underlying layer to conform to the cross-chain standard. Our two-phase protocol is suitable for most of the existing blockchains and it is a universal solution in the true sense.

Conclusion

In this cross chain project, we outlined our core structure based on Read and Write operation to implement the interoperation between blockchains. This project can remove the barrier of information silos and improve the user experience when using and developing DApps. We have provided a precise proof for the safety of this cross chain project. We briefly talked about some new strategies to optimize our cross chain solution for further development. We have drafted an economic model for incentivizing relayers to make our network better. We are devoting ourselves for a better blockchain network, and we hope our project will be a big step in the development process of Web 3.0.

Reference

[1] J. Leon Zhao, Shaokun Fan, and Jiaqi Yan. “Overview of business innovations and research opportunities in blockchain and introduction to the special issue”, Financ Innov (2016) 2: 28.

[2] Tapscott, Alex, and Don Tapscott. “How blockchain is changing finance”, Harvard Business Review 1.9 (2017): 2-5.

[3] Kshetri, Nir. “1 Blockchain’s roles in meeting key supply chain management objectives”, International Journal of Information Management 39 (2018): 80-89.

[4] G. Zyskind, O. Nathan, and A. Pentland, “Decentralizing Privacy: Using Blockchain to Protect Personal Data”, IEEE Symposium on Security and Privacy Workshops, pp (2015): 180-184.

[5] Crosby, Michael, et al. “Blockchain technology: Beyond bitcoin” Applied Innovation, 2.6-10 (2016): 71.

[6] Igor M. Coelho, Vitor N. Coelho, Peter Lin, and Erik Zhang. “Community Yellow Paper: A Technical Specification for NEO Blockchain”, NeoResearch, March 2019, https://neoresearch.io/assets/yellowpaper/yellow_paper.pdf.

[7] Benet, Juan. “Ipfs-content addressed, versioned, p2p file system”, arXiv preprint arXiv:1407.3561 (2014).

[8] “Public Version of the Research Plan for NeoFS: Distributed Decentralized Blockchain-based Storage Platform”, NEO Saint Petersburg Competence Center, February 2019, https://github.com/nspcc-dev/research-plan/blob/master/research_plan.pdf.

[9] Jae Kwon, and Ethan Buchman. “A network of Distributed Ledgers”, Cosmos, https://cosmos.network/resources/whitepaper.

[10] Gavin Wood. “POLKADOT: Vision For A Heterogeneous Multi-Chain Framework”, PolkaDot, https://polkadot.network/PolkaDotPaper.pdf

[11] “A New High Performance Public Multi-Chain Project & A Distributed Trust Collaboration Platform”, Ontology, 2017, https://ont.io/wp/Ontology-Introductory-White-Paper-EN.pdf.

[12] Vitalik Buterin. “Chain Interoperability”, September 9, 2016, https://www.r3.com/reports/chain-interoperability/

[13] Adam Back, et al. “Enabling Blockchain Innovations with Pegged Sidechains”, Blockstream, October 22, 2014, https://blockstream.com/sidechains. pdf

[14] Stefan Thomas & Evan Schwartz. A Protocol for Interledger Payments. https://interledger.org/interledger.pdf

[15] Joseph Poon, Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf

[16] WanChain. Building Super Financial Markets for the New Digital Economy. https://github.com/wanchain/explorewanchain/raw/master/_media/Wanchain-Whitepaper-EN-version.pdf

[17] Jamsheed Shorish. “Blockchain State Machine Representation”, SocArXiv, January 2017.

[18] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”, Bitcoin, 2008, https://bitcoin.org/bitcoin.pdf.

[19] Moore, Edward F. “Gedanken-experiments on Sequential Machines”, Automata Studies, Annals of Mathematical Studies. Princeton, N.J.: Princeton University Press 34 (1956): 129–153.

[20] “Genesis Block”, Bitcoin, https://en.bitcoin.it/wiki/Genesis_block.

[21] “Block Headers”, Bitcoin, https://bitcoin.org/en/developer-reference#block-headers.

[22] Gavin Wood. “ETHEREUM: A Secure Decentralized Generalized Transaction Ledger Byzantium Version”, Ethereum, August 2019, https://ethereum.github.io/yellowpaper/paper.pdf.

[23] “Bitcoin Developer Glossary”, Bitcoin, https://bitcoin.org/en/developer-glossary.

[24] Elad Elrom. “NEO Blockchain and Smart Contracts”, Apress, Berkeley, CA, 2019. 257-298.

[25] “Simplified Payment Verification”, Bitcoin, https://en.bitcoinwiki.org/wiki/Simplified_Payment_Verification.

[26] C. Cachin, R. Guerraoui, L. Rodrigues. “Introduction to Reliable and Secure Distributed Programming”, New York, NY, USA:Springer, 2011.

[27] L. Lamport, “Proving the correctness of multiprocess programs”, IEEE Trans. Software Eng., vol. SE-3, pp. 125-143, Mar. 1977.

[28] Alpern, B. and Schneider, F. B. “Recognizing safety and liveness”, Distributed Computing. 2 (3): 117, 1987.

[29] Bruce Schneier. “Cryptanalysis of MD5 and SHA: Time for a New Standard”. Computerworld, April 20, 2016.

[30] S. Al-Kuwan, J.H. Davenport, R.J. Bradford. “Cryptographic Hash Functions: Recent Design Trends and Security Notions”, IACR Cryptology ePrmt Archive, 2011.

[31] Eliza Paul. “What is Digital Signature- How it works, Benefits, Objectives, Concept”, September 12, 2017, https://www.emptrust.com/blog/benefits-of-using-digital-signatures.

[32] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis. “A survey on long-range attacks for proof of stake protocols”, IEEE Access, vol. 7, pp. 28712–28 725, 2019.