Blockchain Transmission Protocol: A Brief Introduction

Scott Smiley, Cyrus Vorwald, Eric Solomon, Elise Shin
ICON Foundation

MoonKyu Song
ICONLOOP

June 2, 2022

Abstract

An increasing number of teams are creating and launching their own blockchains, drawn both by the merits of building customized platforms and the increasing ease of doing so. As a result of the growing number of blockchains, there is a need for solutions that connect siloed systems to leverage the various offerings spread across blockchains.

In response to this trend, various cross-chain messaging protocols have thus far been introduced, characterized by a tradeoff between cost, security, extensibility, and speed. The Blockchain Transmission Protocol ("BTP") is a chain-agnostic and trustless generic messaging protocol that is designed to minimize this tradeoff with the use of gas-efficient, on-chain light clients.

1 Introduction

The purpose of BTP is to enable complex cross-chain applications with smart contracts residing on two or more blockchain networks with as few security sacrifices as possible. On-chain light clients ("Verifiers") deployed on all BTP-enabled blockchains are key components of the BTP architecture designed to achieve this goal.

While on-chain light clients pose challenges of gas cost and maintenance, the ICON blockchain has been optimized to support this implementation, mitigating the cost of maintenance and providing a secure, cost-efficient, and trustless cross-chain messaging protocol.

This paper provides a summary of the existing solutions in comparison to BTP, the components of BTP, a walk through of the process for integrating those components with a blockchain,[1] and an explanation of the specific modifications

[1]For more information about the data structures in BTP, visit `https://github.com/icon-project/btp/blob/iconloop/doc/icon.md`

made to the ICON blockchain to support this protocol.

2 Existing Solutions

There are several existing models for cross-chain messaging. The following are the most popular solutions as of today:

Proof-Of-Authority Consensus - Rely on signatures by permissioned validators to confirm to the destination chain that a transaction occurred on the source chain. Because validators are assumed to be trustworthy, verification is unnecessary. Example Implementation: Wormhole [1]

SMPC (Secure-Multiparty-Computation) - Securely and secretly break a private key into many pieces, then distribute the pieces amongst a validator set. The validator set can be permissioned (i.e. Proof of Authority) or permissionless (i.e. Proof of Stake). Validators form consensus on transactions that occur on the source chain and relay transactions deemed valid to the destination chain. Example Implementation: MultiChain [2]

Relay + Oracle - Rely on two-party consensus between a Relay with message proofs and an Oracle to provide block data. The transaction proof is checked against the block data to validate its authenticity. Example Implementation: LayerZero [3]

Optimistic Interchain Communication - Assume all transactions are honest, usually after a certain time frame, and rely on off-chain participants to report malicious activity. Malicious activity is easily detectable by design. It can be accompanied with Proof of Stake to punish malicious participants. Example Implementation: Optics [4], Nomad [5]

Light Client - Rely on a light-client to validate data from a source blockchain on the destination blockchain. A Relay forwards message proofs from the source to the destination, where the destination has a light client of the source to validate the message proof. Example Implementation: IBC [6]

3 BTP Security Model

BTP leverages light clients for on-chain verification of messages passed between blockchains. There are two common challenges with light client solutions:

1. Extensibility - Integrating new networks can be difficult and time consuming
2. Maintenance costs - Updating on-chain light clients with live block data can be prohibitively expensive

BTP addresses both of the aforementioned challenges. Extensibility, in the context of integrating new networks to the BTP ecosystem, is achieved through the use of on-chain light clients and a hub architecture. The hub architecture refers to a blockchain that is used as an intermediary to route messages from the source blockchain to the destination blockchain. More details on the ICON

blockchain architecture that enables it to efficiently operate as a hub can be found in the section titled *ICON Blockchain Modifications*.

The hub model includes light clients of all connected networks deployed on one network. It also makes new integrations easier. The hub maintains light clients of all connected networks to verify and route transactions as needed. New integrations must only deploy the light client of the hub, which can be easily copied or translated from existing implementations. See 1 below for a graphical depiction of the hub architecture.

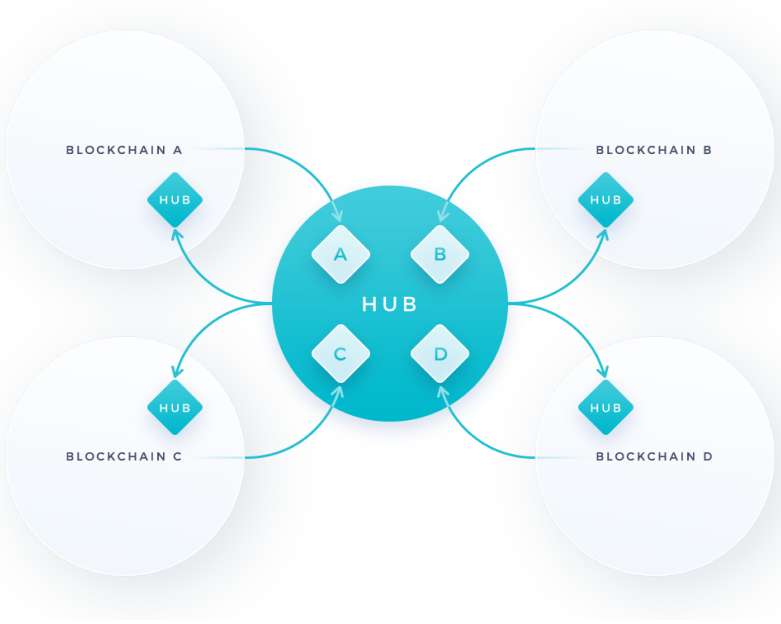

Figure 1: Hub architecture

Existing light client solutions may require the target blockchain's validators to upgrade their software, which calls for a network hard-fork. On-chain light clients remove this requirement because the light client state is deployed and maintained at the smart contract layer. This makes it easier to deploy BTP to new networks, because neither permission nor action is required from the target network.

Maintenance costs are addressed by two features on the core ICON blockchain: BTP Blocks and the Verifier Whitelist. See the section titled *ICON Blockchain Modifications* for more information on how maintenance costs are addressed using these features.

4 Components

BTP includes one off-chain component as well as a set of smart contracts to process and verify messages (the reason for the off-chain component is explained in the component subsection titled *Message Relay.*):

1. Service Handler
2. Message Broker
3. Message Relay (off-chain)
4. Message Verifier

For more information about the components, visit `https://github.com/icon-project/IIPs/blob/master/IIPS/iip-25.md`

4.1 Service Handler

The service handler handles requests to send any instructions from one blockchain to another. Only permissioned Service Handlers may call a Broker (described in the next section) in order to prevent malicious contracts from attacking the BTP network.

The primary service handler is the Arbitrary Call Service, which allows for third-party developers to build their own bespoke services as extensions of BTP. End users will call the Service Handler on the source blockchain, resulting in a BTP message being sent to the destination blockchain. *Depending on an application developer's design decisions, users may need to call functions to execute the transaction on the destination network.*

4.2 Message Broker

Message brokers, which may be shortened to brokers, act as central hubs of all BTP messages for their respective networks. Each BTP-enabled network has one message broker smart contract at minimum – however, it can have more to scale with the number of messages if needed.

The message broker links the permissioned service handler and message verifier. On the sending blockchain, the message broker translates service messages from the service handler to BTP messages and sequentially routes BTP messages to the message relay. On the receiving blockchain, the message broker sequentially routes all relay messages to the message verifier and sequentially routes verified BTP messages to the service handler.

4.3 Message Relay

The relay, short for message relay, is a necessary off-chain component because blockchain networks are inherently restricted from initiating outward calls to external networks. In the context of BTP, a relay is defined as an incentivized

off-chain server that forwards messages and synchronizes the message verifiers between blockchains.

Relay messages are Recursive Length Prefix (RLP)-encoded BTP messages that include block updates, block proofs, and message proofs.[7] All relay messages are delivered in order, using a sequence number that prevents skipped or duplicated messages.[2]

4.4 Message Verifier

The message verifier is an on-chain light client of the source blockchain, stored in a smart contract on the destination blockchain. It sequentially decodes and verifies relay messages using block headers and validator signatures from the source blockchain.

Message verifiers track block headers, as opposed to full blocks, and receive block updates from message relays to arrive at the latest state. This state is typically the merkle root of the old block headers. With this information, the message verifier can prove inclusion of a given block header using only the security assumptions of the source blockchain.

5 Integration of Components

This section will walk through an example transaction of sending a BTP-enabled token from Blockchain A to Blockchain B using the Arbitrary Call Service and each of the aforementioned components.

5.1 Arbitrary Call Service

The following example uses the Arbitrary Call Service to enable a proof of concept of a "BTP Token." A wrapped token is collateralized by native tokens on the source blockchain locked in a smart contract awaiting redemption, while a BTP token has mint and burn permissions triggered by BTP Messages as illustrated below. Note that certain transactions should be able to be reverted in the event of cross chain message failure. For a BTP token, this means refunding burned tokens if the cross-chain transaction fails.

5.2 Example

Let's say a user wants to perform cross chain transaction CCT_x to transfer x amount of BTP-enabled token TK[3] from blockchain A to blockchain B. The user's wallet address on A is A_{addr} and the user's wallet address on B is B_{addr}. We represent this transaction as payload P with ordered unique identifier N.

[2]For more information about the relay message data structure, visit `https://github.com/icon-project/btp/blob/iconloop/doc/icon.md#relaymessage`

[3]A BTP-enabled token transfers tokens across blockchains by burning tokens on the sending blockchain, and minting tokens on the receiving blockchain.

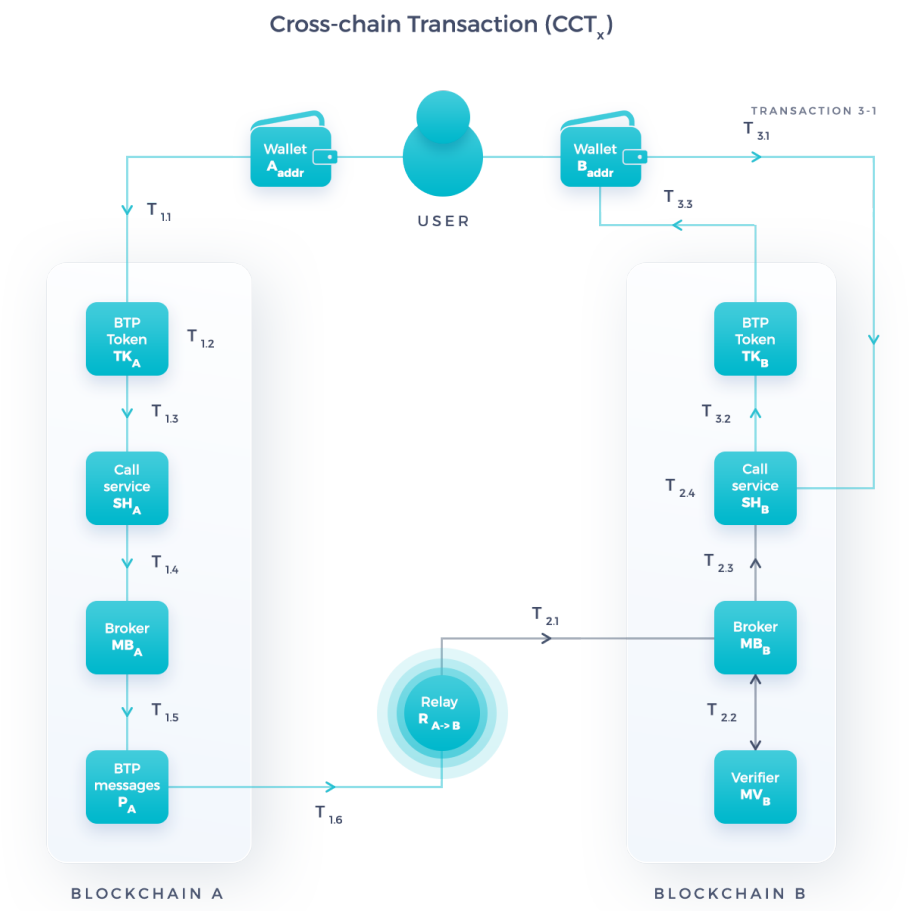

Figure 2: Cross-chain transaction

Atomicity

CCT_x consists of 3 on-chain transactions. CCT_x is successful if and only if the following 3 transactions are all successful.

$$T_1 = A_{addr} - x; P_A \rightarrow P_R$$
$$T_2 = P_R -> P_B; verify(T_1)$$
$$T_3 = B_{addr} + x$$

Otherwise, CCT_x is rejected, the process backs out, and x user funds at A_{addr} are intact.[4]

[4]Note that certain transactions should be able to be reverted in the event of cross chain message failure. For a BTP token, this means refunding burned tokens if the cross-chain

T_1: Initiated on the source chain by the user

1. User calls $TK_A.send(B_{NID}, B_{addr}, x)$ to initiate T_1. B_{NID} is B's network ID.
2. TK_A burns xTK from A_{addr}.
3. TK_A calls SH_A.sendCallMessage(B_{NID}, P_A, E), where P_A is the CCT_x payload on A and E is the rollback function for error handling associated with $TK_A.send$.[5] SH_A assigns monotonically increasing sequence number N to P_A.[6]
4. SH_A calls $MB_A.sendMessage(B_{NID}, SH_{B_ID}, N, P_A)$, where MB_A is the message broker on blockchain A, and SH_{B_ID} is the ID of the service handler on blockchain B.
5. MB_A emits P_A.
6. Relay $R_{A\to B}$ listens for P_A.

$T2$: Initiated on the destination chain by $R_{A\to B}$

7. $R_{A\to B}$ routes P to MB_B via $MB_B.handleRelayMessage(MB_{A_ID}, P_R)$, where P_R is the CCT_x payload RLP encoded and includes merkle proof that $P_A \exists A$.
8. MB_B calls $MV_B.handleRelayMessage(MB_{B_ID}, MB_{A_ID}, N, P_R)$ which decodes and verifies P_R.
9. MB_B calls $SH_B.handleBTPMessage(A_{NID}, SH_{B_ID}, N, P_B)$ to trigger P_B.
10. SH_B emits P_B to notify the user of T_2's execution.

T_3: Initiated on the destination chain by the user[7]

11. User calls $SH_B.executeCall(N)$ to begin the minting process.
12. SH_B calls $TK_B.mint(x, B_{addr})$.
13. TK_B mints xTK to B_{addr}.

Note that in this example, CCT_x involves transferring x TK from A to B, but it could be a cross chain transaction to do anything supported by smart contract functionality. $f(x) = A_{addr} - x$ and $h(x) = B_{addr} + x$ could be replaced with any other functions, and the rest of the steps are the same.

---

transaction fails.

[5] If $\forall(T_1, T_2, T_3)$ fail, a failure message routes to SH_A which calls E to revert A's state.

[6] For simplicity, we assign the service handler's sequence number to be the same as the message broker's sequence number. In practice, these parameters are different because there can be many independent service handlers routing to the same message broker.

[7] Note that T_3 could be initiated by any account, but in this example it is initiated by the user.

6 ICON Blockchain Modifications

BTP's light client architecture would typically be prohibitively expensive because proof verification is computationally expensive and every block must be verified to keep track of changes to the validator signature list. Additionally, certain smart contract environments may not support the necessary hashing functions, which would require a network hard fork to support BTP.

However, the ICON blockchain will be modified[8] to reduce these costs and remove the necessity for network hardforks using two key features: BTP blocks and the Verifier Whitelist.

6.1 BTP Blocks

While ICON blocks are produced every two seconds regardless of BTP activity, BTP blocks are produced only when a BTP Message is sent. BTP blocks are validated by ICON's validator set and are verifiable through ICON blocks.

BTP blocks are not technically blocks themselves, but contain many similar properties. For example, all BTP blocks have a header, but the blocks themselves are not signed by ICON validators. The roots of BTP blocks are included in ICON blocks, which are then signed by ICON's validator set. See Figure 3 for a graphical representation of BTP blocks.

There will be separate chains of BTP blocks per target network. For example, BTP blocks for Blockchain 1 would only be produced when a user sends a BTP Message to Blockchain 1, while BTP blocks for Blockchain 2 would only be produced when a user sends a BTP Message to Blockchain 2.

There are two primary benefits of BTP blocks:

1. **Hashing Algorithms** - Destination chains do not need to support ICON's specific cryptographic hashing algorithm. Instead, ICON supports the hashing algorithms of the integrated destination chains.
2. **Update Frequency** - Verifiers on non-ICON chains only need to be updated when there is a BTP Message directed to that respective chain.

6.2 Hashing Algorithms

BTP blocks use cryptographic hashing algorithms that are supported by the smart contract environments of their respective destination chains. For example, BTP blocks with Ethereum as the target network will use the Keccak hashing algorithm since that is what is supported by EVM. Without BTP blocks, ICON would need destination chains to support the SHA3-256 hashing algorithm in their smart contract environments to verify ICON blocks with reasonable gas costs. This often requires a hard fork of the destination chain.

[8] At time of writing, the ICON Network does not have these modifications in production. All blockchain modifications require consensus of validators using network governance processes.

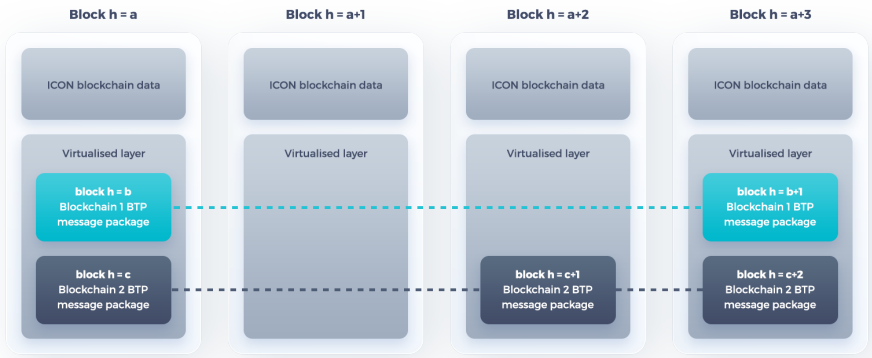

Figure 3: BTP blocks

6.3 Update Frequency

Message verifiers must receive all blocks in order to maintain their trustless nature. If there is no BTP message in a block, this would result in wasted resources on block update transactions with no relevance to BTP. ICON has 2-second block time, which could lead to many irrelevant block updates.

To solve this issue, the ICON Network has implemented the BTP block as an internal blockchain structure specifically dedicated to validating BTP messages. BTP blocks will not be produced unless there is a BTP message, meaning block updates will only be sent to a non-ICON message verifier when there is a BTP message directed toward that respective network. This results in considerable savings on gas costs.

6.4 Verifier Whitelist

ICON-based message verifiers will also require regular block updates from message relays. In order to mitigate the cost for message relays, the ICON Blockchain implements a whitelist for message verifier contracts that will remove the gas costs for all successful block update calls.

In order to monitor the computational resources used by message verifiers, the whitelist still calculates the gas cost of each transaction, but does not apply the fee on successful transactions. This mitigates the cost of maintenance while still monitoring network resource consumption.

An alternative to the Message Verifier Whitelist would be to add the message verifiers as precompiled contracts. This solution, however, would limit the extensibility of BTP and add significant complexity to ICON's core infrastructure. Adding new precompiled contracts would require a network hard fork when adding

a new BTP integration and the message verifiers would need to be written in Go, which is far more difficult than Java, Solidity or Rust.

7 Conclusion

Most existing cross-chain messaging protocols rely on trust assumptions to enable practical implementations. This paper proposes a hub-based light client model, in which an intermediary blockchain and an associated off-chain message relay system maintain on-chain light clients of all connected networks and route messages between the source and destination blockchains to enable cross-chain transactions.

The proposed solution includes the use of an on-chain light client responsible for verifying cross-chain messages. While on-chain light clients pose challenges of gas cost and maintenance, the ICON blockchain has been modified to support the proposed hub model, which mitigates the cost of maintenance and makes BTP the optimal solution for secure, cost-efficient and trustless cross-chain messaging.

Disclosure: The information described in this paper is preliminary and subject to change at any time. Furthermore, this paper may contain "forward-looking statements."[9]

References

[1] What is Wormhole?, `https://docs.wormholenetwork.com/wormhole/`

[2] Multichain introduction, `https://docs.multichain.org/getting-started/introduction`

[3] What is LayerZero, `https://layerzero.gitbook.io/docs/`

[4] Optics, `https://docs.celo.org/celo-codebase/protocol/optics`

[5] What is the Nomad Protocol?, `https://docs.nomad.xyz/`

[6] High-level Overview, `https://docs.cosmos.network`

[9] Forward-looking statements generally relate to future events or our future performance. This includes, but is not limited to, ICON's projected performance; the expected development of its business and projects; execution of its vision and growth strategy; and completion of projects that are currently underway, in development or otherwise under consideration. Forward-looking statements represent our management's beliefs and assumptions only as of the date of this presentation. These statements are not guarantees of future performance and undue reliance should not be placed on them. Such forward-looking statements necessarily involve known and unknown risks, which may cause actual performance and results in future periods to differ materially from any projections expressed or implied herein. ICON undertakes no obligation to update forward-looking statements. Although forward-looking statements are our best prediction at the time they are made, there can be no assurance that they will prove to be accurate, as actual results and future events could differ materially. The reader is cautioned not to place undue reliance on forward-looking statements.

[7] RLP, https://eth.wiki/fundamentals/rlp