Timber Network Session Protocol:
A Trustless Incentivized Message Bridge-Encapsulating Router

Tomo Saigon, *Timber Network*

Abstract

We present the Timber Session Protocol, a comprehensive standard protocol for the establishment of persistent cross-chain communication sessions, characterized by trustlessness and agnosticism to the underlying transport protocol utilized for message bridging. The protocol functions by establishing a trust-minimizing connection through an incentivized message bridge, encapsulating a designated route over the bridge. The session established by the protocol is connection-oriented, bi-directional, long-lasting, and designed to provide reliable guarantees for quality of service, as well as protections against data corruption during message transit.

The objective of the Timber Session Protocol is to offer a general cross-chain communication solution, enabling developers to interface with a standard contract API for contract-to-contract messaging, by treating contracts on the local chain as virtual contract interfaces to the remote chain. The protocol delivers robust security through the combination of the session protocol and the underlying bridge facility.

1. Background

The proliferation of blockchains, many of which possess distinct features, has driven up the value of interconnecting these blockchains into a network to leverage these unique capabilities. This growth has been accompanied by an increase in cross-chain communication facilities, each with its own set of distinctive features and advantages. With the emergence of truly decentralized cross-chain messaging protocols, such as Axelar, there arises the requirement for standardization of their interfaces to facilitate the development of cross-chain applications.

In response to this requirement, we propose the establishment of a standard session protocol, layered on top of general cross-chain messaging transport protocols. This session protocol would provide application developers with the capability to securely transmit payloads to smart contracts on remote chains through interaction exclusively with their local chain.

The exponential increase in the number of blockchains, from a single instance in 2009 to thousands today, is expected to continue in the future. The emergence of new blockchains, each offering unique features and capabilities, has created a highly competitive marketplace, where bridging blockchains has become a crucial aspect to attract users to move from existing blockchains to new ones.

Factors such as scalability, privacy, interoperability, Turing-complete smart contracts, and governance are among the key differentiators among the newer blockchains. The development of cross-chain bridges has progressed over the years, with early generations focusing mainly on token transfers, while the later generations have expanded their capabilities to support general message passing and remote logic execution.

The mechanism of cross-chain message bridges operates similarly to the communication of computers in sending email. When a computer sends an email using an application, it first creates a connection and establishes a session with an email server, and the email content is transmitted through multiple layers of abstraction and protocols as described by the seven layer OSI model before reaching its destination. The same holds true for messages sent from an application on a blockchain to another blockchain, which are processed through various intermediate data formats and protocols before reaching their final destination.

The lower-level networking protocols, such as TCP, are utilized by both email and blockchains, which themselves act as network protocols. Application developers on smart contracts operating on blockchains only need to conform to the standard interfaces for the blockchain and its virtual machine, freeing them from the responsibility of managing the blockchain protocol itself.

However, the current landscape lacks a standard for sending cross-chain messages, with each message bridge offering its own bespoke API. This imposes a challenge for smart contract developers who aim to send messages to smart contracts on different blockchains, as they must learn the specific API of the message bridge they plan to use, such as Axelar.

2. Assumptions

Assuming the existence of an application that leverages the accounts, data, assets, and features of one blockchain in conjunction with those of another, the need arises for a

means of copying and transporting arbitrary data between blockchain protocols. The Timber Voting Protocol describes an example of such an application. The second assumption is that a message bridge, which may be decentralized such as Axelar or centralized such as POA Network's TokenBridge, will be available to facilitate this transfer of data. It is further assumed that the use of such a message bridge will require payment for service, but will not necessitate obtaining permission from any third party. It is sufficient that the existing message bridge operates on payment using a widely available currency on the source chain.

2. Session protocol

The proposed Timber session protocol offers a unified, standardized interface for cross-chain communication. It sits on top of any existing arbitrary cross-chain messaging transport protocol, providing an extra layer of security, authentication, and message ordering to the underlying transport.

The session protocol uses a Berkley sockets-like interface, making it easy for developers to use without having to understand the intricacies of the underlying cross-chain messaging transport protocol. By using the session protocol, developers can send payloads to smart contracts on remote chains with ease, interacting only with their local chain, while also protecting against spoofing and denial of service attacks, and providing authentication of data integrity and reducing the trust requirements of the underlying transport protocols.

The session protocol uses a message format that includes a header field describing the rest of the message, sequence numbers, connection hashes to protect against cross-session replay attacks, a standard formatted signature, and container for the raw message. The bidirectional nature of the session protocol allows for response and control messages to be sent back to the source. The protocol also maintains internal state that can be verified externally, providing an added layer of security and authenticity.

Thus, the proposed Timber session protocol provides a simplified and standardized way for developers to take advantage of cross-chain communication, without having to navigate the complex landscape of different cross-chain messaging bridges and protocols. The standardization of the session protocol would allow application developers to focus on writing cross-chain applications rather than the low-level details of communication between chains. The session protocol would hide the complexities of the underlying messaging transport protocol, making it much easier to write cross-chain applications securely.

3. Message ordering

Message bridges may not guarantee the order in which messages are delivered, and assume that the messages themselves are atomic and not interdependent. However, applications may require messages to arrive in a specific order for their logic to work correctly. As an illustration, consider an account that holds 100 tokens and is instructed to first double its token count and then subtract 100 tokens. If the instructions are received in the wrong order, the account may end up selling all of its tokens before doubling the amount, rendering the latter operation useless.

To maintain the proper order of messages, the session protocol will start with a randomly generated sequence number that is synchronized with the remote chain. This number will be confirmed and kept up-to-date on both sides to ensure a consistent order of messages. In case of an out-of-order delivery, the messages will be held in a queue and processed only after all intermediate messages have been delivered.

The session protocol will filter any incoming messages that are not part of the established session, preventing a flood of messages from affecting the state of the contract. Any such messages will be thrown out with costs incurred by the attacker.

4. Data integrity

In order to ensure consistency between both sides, each new message is hashed with a previously saved hash, forming a chain that can be verified. A forgery of a message on the remote side would result in a forked chain hash, which can be easily detected through external means. The messages sent within the session protocol include a signature from the sender, which verifies the source, intended destination, and that both sides have a consistent chain hash.

To prevent attempts of injecting messages with unauthorized privileged instructions, the session protocol includes a Merkle witness. This witness proves that the signer is part of a Merkle tree of approved signers, which is established when the session is first initiated.

5. Spoofing attacks

The session protocol includes measures to detect malicious or unauthorized payloads which anyone can send to contracts on a public blockchain. The protocol checks the source of the message against the source defined during the session initialization, as well as verifying that the message contains the correct connection hash. This ensures that messages are only accepted from the intended sender and

prevents spoofed messages from being processed. The sender's signature is also verified to ensure data integrity. The protocol filters out messages with incorrect or already processed sequence numbers to provide an additional filter against attack. Further data integrity is described in the previous section.

6. Conclusion

In conclusion, the advancements in arbitrary message bridges exemplified by Axelar have opened new doors for developers to create innovative cross-chain applications. The proposed Timber session protocol adds an extra layer of security and reduces the trust requirements on these message bridges, making it easier for developers to securely build cross-chain applications. With its features such as message ordering, protection against spoofing and denial of service attacks, and data integrity authentication, the session protocol enables developers to safely take advantage of the combined features of multiple blockchains in their applications.