HDS Serenity Ledger: Developing a High-Dependability Blockchain System

André Jesus (110860) André Páscoa (110817) Nyckollas Brandão (110893)

Instituto Superior Técnico, Universidade de Lisboa
{andre.f.jesus, andre.pascoa, nyckollas.brandao}@tecnico.ulisboa.pt

Abstract

The project "HDS Serenity (HDS^2)" aims to develop a high-dependability permissioned blockchain system using the Istanbul BFT Consensus Algorithm (IBFT). This report details the design, implementation, and testing of HDS^2. The system includes a client-side library, server-side logic, communication, and cryptographic utilities, focusing on static membership, fault tolerance, and network reliability. Implementation follows Java standards and emphasizes security through cryptographic primitives. A comprehensive testing suite ensures system correctness and robustness under Byzantine attacks. Overall, the report showcases the development process and the achieved dependability of HDS^2.

1 Introduction

In the initial phase of our project, our focus was on solidifying the implementation of the Istanbul BFT Consensus Algorithm (IBFT) [1] with the integration of the round change mechanism. Alongside, we emphasized ensuring the integrity and authenticity of message exchange through a Public Key Infrastructure (PKI). Additionally, significant efforts were dedicated to developing a client library and clients, complemented by thorough testing and evaluation of the system's functionality.

Building upon the achievements of the first stage, this second phase of our project aims to further enhance the HDS Serenity (HDS^2) blockchain system. Our primary objectives include refining application semantics and dependability guarantees while extending the system's capabilities to incorporate a cryptocurrency application. Strengthening the system's Byzantine fault tolerance mechanisms to withstand potential attacks, including those orchestrated by malicious clients, is also a core focus.

This paper presents the roadmap for the second stage of our project, outlining the key goals, methodologies, and outcomes.

2 System Design Overview

Our system is composed of multiple nodes responsible for maintaining the blockchain, which communicate with each other using the IBFT. Clients interact with these nodes through a client library, functioning as an API for executing operations on the blockchain. Currently, the client application offers two operational modes: when provided with a script as an argument, it executes commands read from the script; alternatively, without a script argument, it operates as a command-line interface, allowing users to directly input and execute commands.

3 Implementation Details

3.1 Communication

3.1.1 Authenticated Perfect Links

Communication between all entities is facilitated through Authenticated Perfect Links. We utilize the provided perfect link implementation, that works over UDP, and then employed a digital signature algorithm to guarantee authentication, integrity and non-repudiation. This involves each sent message being signed with the sender's private key and subsequently verified by the receiver using the sender's public key. We opted for digital signatures over Message Authentication Codes (MAC) due to the absence of infrastructure for symmetric key sharing necessary for MAC usage. Additionally, digital signatures offer enhanced security by guaranteeing non-repudiation.

Furthermore, to enable secure communication, all entities possess access to the public keys of the other nodes beforehand, retrieved from configuration files.

3.1.2 Message Serialization and Types

Messages sent via Authenticated Perfect Links are of the type `Message` which include a `senderId`, `messageId` (to distinguish messages from the same sender), and a `type` (identi-

fying the message's purpose in the system, which could be one of the consensus messages like `PREPARE` or `COMMIT`, or operations from the client library like `BALANCE` or `TRANSFER`). The `Message` object is serializable.

We have also introduced a type called `SignedMessage` which includes the message and its corresponding signature, facilitating transmission via the Authenticated Perfect Link.

Furthermore, we have created several extensions/abstractions of the `Message` type, including `ConsensusMessage` which represents a message in the consensus algorithm, and `SignedLedgerRequest`, representing a request sent by the client library to the blockchain system. This `SignedLedgerRequest` represents a transaction in the system, and as the name implies, this request is also signed to ensure it was made by the client (ensuring authentication, integrity, and non-repudiation).

3.2 Consensus Algorithm

In the first stage, the majority of the consensus algorithm was already implemented, and we focused on implementing the round change mechanism according to the IBFT [1] algorithm, with the round-change being triggered upon the expiration of a timer for the consensus instance.

In the second stage, some improvements/changes were made:

- **Consensus Value Type:** The consensus value type is now a `Block` instead of a string.
- **Message Buckets:** Initially, message buckets stored `Message` objects. However, in our implementation, they now store `SignedMessage` objects. This adjustment proves beneficial in certain resynchronization mechanisms, such as when sending a quorum of commits upon receiving a round change for an instance already decided. It's crucial to send the message with its signature to ensure authenticity and integrity. If the messages in the quorum didn't keep their signatures, the sender of the quorum would be seen as an impersonator of the initial senders. What happens instead is the messages are seen by the receiver as being sent by the initial senders of the message.
- **Justification Piggyback Mechanism:** We implemented the piggyback mechanism for justification in the round change process. A round change must be justified; therefore, when a round change is sent, the sender should include a set of messages that justify that round change and are processed before the round-change message.
- **Leader Election Round-Robin:** In each new round or instance, a new leader is elected using a round-robin approach. This prevents a node from being a leader for an extended period, thereby ensuring fairness in block creation and distribution of block creation rewards (fee).

These improvements enhance the robustness and efficiency of the consensus algorithm, ensuring smoother operation and faster decision-making in the system.

3.3 Blockchain

3.3.1 Message Accumulation and Consensus Initiation

Like a classic blockchain system, all received transactions are accumulated in a message accumulator, oftentimes called a memory pool or "mempool".

Once accumulated, there are two ways to start a consensus with a block:

1. a threshold is met, and a block is created with the accumulated messages;
2. a timer elapses and all the messages accumulated are used in the block. This ensures that if the system does not receive many transactions in a while, the timer will elapse, and the block can be created with the accumulated messages until now, instead of waiting indefinitely for the threshold to be achieved.

After one of these events occurs, a block is created and used to initiate a consensus instance.

The accumulated messages are only removed when a block is decided, this guarantees that the client's requests are always eventually decided if they are valid.

3.3.2 Ledger

Upon the occurrence of a decision event, the block is added to the ledger if it is valid. A valid block ensures that all transactions inside it are legitimate (signed by the client). Additionally, for transfer requests, it verifies if the sender's account has a balance greater than or equal to the sum of the requested amount and the transaction fee, as accounts cannot have a negative balance. If the block is deemed valid, all transactions within it are executed.

After adding the block to the ledger, the block's requests are removed from the message accumulator. Since one can only validate a block if the previous consensus instances have been decided, this guarantees that the accumulated messages are not decided multiple times.

The accounts within the ledger are established when the system initializes, with one account allocated for each entity participating in the system. These accounts are uniquely identified and accessed through the hash of the respective account owner's public key. This design ensures transparency and security within the ledger, allowing for efficient tracking and validation of transactions across the network.

3.3.3 Transaction Fee

Transactions in the blockchain system incur a transaction fee. This fee serves as a payment for the leader to include the transaction in a block and process it promptly.

4 Security Measures and Resilience Against Attacks

4.1 Security Guarantees

In the context of our system, several security guarantees are provided to ensure the integrity and reliability of transactions. These guarantees encompass various aspects of communication and consensus protocols:

- **Authenticated Perfect Links:** As mentioned earlier, authenticated perfect links with `SignedMessages` ensure authentication, integrity, and non-repudiation in the communication between entities.
- **SignedLedgerRequest Guarantees:** `SignedLedgerRequests` also provide these guarantees, ensuring that a client sent the request. This guarantees, for example, in a transfer request, that the source account is from the client that sent the request.
- **Client Guarantees:**: To verify that the responses coming from the servers are valid, the client accumulates the responses until it receives f+1 **equal** responses, this guarantees that atleast one of the responses must be coming from a non-byzantine node.
- **IBFT Guarantees:** Furthermore, IBFT ensures additional guarantees such as agreement, validity, and termination. Through the use of quorums and cryptographic mechanisms described above, the system demonstrates robustness against various attacks, thereby enhancing overall security.

Moreover, it's important to note that the system exhibits strong safety properties, as the consensus algorithm employed ensures agreement on the state of the ledger, thereby minimizing the risk of double-spending or other malicious activities. Additionally, liveness is typically ensured in the majority of cases, ensuring the timely processing of transactions.

While confidentiality is not inherently guaranteed in our system, potential improvements could involve encrypting all messages exchanged between nodes and clients. However, we have not considered this a critical issue in the system design, as even in the worst-case scenario where an attacker can observe messages, they cannot modify them without being detected due to the cryptographic mechanisms in place.

4.2 Behavior Under Attack

To verify the behavior, correctness, and robustness of the system under various fault models, including crash fault and arbitrary fault scenarios, we have enhanced the configuration of individual nodes. These properties classify nodes as exhibiting normal behavior or simulating specific Byzantine behaviors, such as:

- **Non-Leader Consensus Initiation**: Initiating a consensus without being the leader to test if the system can handle unexpected actions. Other nodes will see the node is not the current leader and ignore the message.
- **(Leader) Impersonation**: Simulating being the leader (or other node), causing other nodes to fail signature verification, testing resilience against malicious actors. Other nodes will see that the node is not the leader because the message is signed with his private key, not with the leader's private key.
- **Corrupt Broadcasting**: Broadcasting different values to different nodes to assess the consistency and fault tolerance of the system. IBFT guarantees that this is not a problem by using quorums.
- **Corrupt Leader**: Leader sends different messages to different nodes. IBFT guarantees that this is not a problem by using quorums.
- **Node Crashes**: Simulating crash failures by abruptly terminating node processes after a fixed duration of operation. A round change will occur.
- **Robber Client**: Client attempts to transfer money from another account to its account (tries to steal money). Since the `LedgerRequest` is signed by the client, when the transaction is validated, it will not be valid, as the source account owner did not sign the request.
- **Quiet Leader**: The leader is elected but does not start a consensus. A round change will occur.
- **Leader Bullies a Client**: The leader bullies a client and ignores its transactions. However, the round-robin mechanism in leader election prevents this faulty leader from being the leader for an extended period.
- **Leader Attempts to Charge a Higher Fee**: The leader tries to charge a higher fee, effectively stealing money from the source account client. However, the quorum mechanism ensures that correct nodes have the correct balance in their replicated ledgers, preventing such fraudulent transactions.

Since the algorithm is well-implemented and there is always a quorum of correct nodes, our system is prepared for these situations. All tests ran without problems, proving that

our system provides safety, integrity, authenticity, reliability, and availability.

5 Conclusion

In summary, our project has successfully achieved all objectives outlined in both stages. Through meticulous design and implementation, we've established a robust decentralized system capable of securely managing transactions and maintaining a transparent ledger. However, there are areas where further enhancements could elevate the system's functionality and efficiency:

- Enhance the connection protocol between clients and nodes to optimize message dissemination. The current broadcasting method to all nodes might not be the most efficient approach, necessitating a more streamlined communication mechanism.
- Implement a robust mechanism to identify and exclude Byzantine nodes from the system promptly. Detecting and isolating such nodes would significantly enhance the overall reliability and security of the system, safeguarding it against malicious attacks or erroneous behavior.
- Explore the possibility of allowing clients to possess multiple accounts within the system. Introducing an account creation mechanism would provide flexibility and enable clients to manage their assets more effectively, catering to diverse use cases and enhancing user experience.

These proposed improvements aim to improve the system's performance, resilience, and user-friendliness, ensuring its continued success in facilitating secure and efficient decentralized transactions.

References

[1] Henrique Moniz. The istanbul bft consensus algorithm. *arXiv preprint arXiv:2002.03613*, 2020.