

HDS Serenity Ledger

Delivery 1

Highly Dependable Systems

Group number: 08

95579: Francisco Sousa

95641: Miguel Porfírio

95674: Sara Aguincha

Computer Science and Engineering
MEIC-A

2023/2024

Contents

1	Design Analysis	1
1.1	Communication	1
1.2	Round Changes	1
1.3	Future Implementations	2
2	Dependability Guarantees	3
3	Tests	4
	Bibliography	5

Chapter 1

Design Analysis

HDS Ledger is a simplified blockchain system with high dependability guarantees. The system consists fundamentally of Clients and Nodes. The client application makes calls to a Library, which translates them into requests to the service, to trigger instances of the consensus protocol.

1.1 Communication

The base code came with a Link class implemented for handling the communication. After reviewing the code, we concluded that on top of the **Fair Loss Link**, the **Stubborn Link** and **Perfect Link** layers were already present as well. We implemented authentication by the use of signatures, thus making it an **Authenticated Perfect Link**. Every message is hashed and asymmetrically signed by its author. This way, all messages are authentic and preserve integrity. The hash algorithm used for signing is SHA-512 and the asymmetric encryption is ensured by RSA. Hence, all communications in the system offer non-repudiation.

1.2 Round Changes

The base code already provided an implementation of the IBFT consensus algorithm [1], though incomplete, as there were no round changes implemented. This generates a liveness problem whenever the leader becomes faulty/byzantine. To implement the round changes, we've created a new type of message (*ROUND-CHANGE*) and

implemented a *timer*, that when expired, triggers a round-change request from that node, followed by an increment to the current local round and a broadcast to all other nodes, requesting a round-change for the new round value. Once a quorum of round-change messages has been achieved, the round-change is triggered and a new leader is implemented by all correct nodes. The leader is chosen by being the next one in the sequence of node IDs. The new leader then starts a new round of consensus by broadcasting a *PRE-PREPARE* message with either the highest prepared value of the quorum, if it exists, or the input value of that instance, thus assuring protection against byzantine nodes/leaders. This implementation follows the IBFT algorithm closely and the justification of messages is ensured by what was previously written, with all messages offering signatures and the authenticity of any quorum being easily verifiable.

The base code also included mechanisms to prevent a byzantine node from trying to prepare/commit a different value than the one pre-prepared by the leader. This is done in the form of Quorums, which are used to check if a majority of nodes ($2f + 1$) are trying to perform the same *correct* action. This means that *correct* nodes will only accept prepare/commit values that were initially pre-prepared by the leader.

1.3 Future Implementations

Our current implementation is still prone to some byzantine faults.

In this implementation, a client sends the value that it wants to append to all nodes and does not check if it was successful. We will later implement a verification that checks if at least $f + 1$ nodes have committed the value, as this implies that at least one correct process was successful.

There is also the problem where the client sends its value to a byzantine node, which can then arbitrarily change it. Later on, we will sign the value that the client wants to append with its private key, in order to ensure that no byzantine leader is able to change that value.

Chapter 2

Dependability Guarantees

Whenever a round-change is triggered, the new proposed value chosen by the leader is safe and is ensured by the a justification for both *PRE-PREPARE* and *ROUND-CHANGE*. Our implementation follows algorithm 4 of the IBFT specification [1].

The addition of round changes ensures liveness in the system, as it is guaranteed that a faulty/byzantine leader will eventually be changed to a correct one and the system will maintain its availability.

Adding authenticity and integrity to all messages is essential to prevent any impersonation attacks and byzantine processes trying to alter values throughout the execution of the protocol.

Chapter 3

Tests

Our system has proven to withstand the following byzantine behaviours:

- Node **impersonating** a leader;
- Leader/Node **crashing** or **not participating**;
- Node sending *PREPARE* or/and *COMMIT* with a **different value**;

Bibliography

- [1] H. Moniz, “The istanbul bft consensus algorithm,” *arXiv:2002.03613v2 [cs.DC]* 19 May 2020, 2020.