HDS Legder - 2nd Stage Report

Bruno Mateus (95543)[1], João Pereira (95600)[1], and José Afonso (95613)[1]

[1]Group 31

1 Introduction

This report will describe our implementation, based on Henrique Moniz's paper[1] without leader election, used to implement state machine replication in the Quorum blockchain.

2 Design

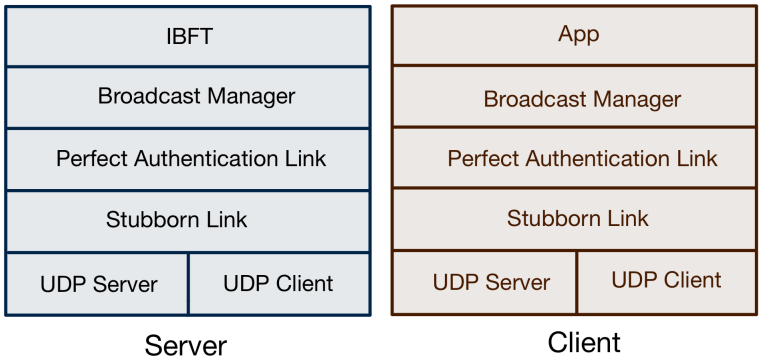

Figure 1. Structure for both Server and Client

The design model used for this stage is the same as the model (Fig. 1) used for the first stage as most of the work developed for the second stage leverages the existing functionality that was previously developed.

3 Blockchain and Operations

In this stage, 3 new operations were introduced: create an account, read an account's balance, and make transactions from one account to another. From these, only transaction operations requested by clients will affect the state of the blockchain, therefore only these operations will be appended or attempted to be appended, to the blockchain. Since demand for transactions is high and, performing a consensus round for each transaction would cause performance issues, transactions are aggregated into blocks of fixed size and each consensus instance is performed on these transaction blocks.

3.1 Transaction Validation

The effect of each transaction is only reflected on the accounts' balance after the consensus for the block (where the given transaction is inserted) is performed. For validating the transactions of clients and also the transactions of the leader, since the leader may be faulty, the servers use another balance called TEMPBALANCE. After every round of consensus, this TEMPBALANCE starts equal to the actual balance and it is used to keep the state of the validation of transactions, once a transaction is validated TEMPBALANCE is updated as if the transaction was made, in order to properly validate the transactions that will follow. It is important to note that any update to TEMPBALANCE is not propagated to the correct Balance of the account.

3.2 Accounts

Other than the blockchain, servers will also keep the state of a set of accounts. Each of these accounts is associated with the public key that was used to create it. Meaning that every account must belong to an entity holding a unique (privateKey, publicKey) pair. In order to perform actions to an account, the entity trying to do so must be properly authenticated by communicating using that same key pair.

In our project, only clients can create accounts, with the exception of the leader server's account, which in our project is fixed. This account will be used to charge a fee to the clients who perform transactions that are appended to the blockchain.

3.3 Reads

In order to check the balance of an account, a client will have two options to do so. By performing a weak read or a strong read. These two operations offer different semantics.

3.4 Weak Reads

A weak read ensures the client that he will receive a correct output, in our case a correct balance; however, this value might not be the most updated one. This operation only requires the client to contact one server, and the response it obtains, specifically the freshness of the output will vary from server to server.

In order for the state to be correct and for the client to have the possibility of verifying it, our system implements another type of consensus, described in subsection 3.6.

3.5 Strong Reads

With strong read requests, the client sends the last block it read. When the servers receive this request, they verify if the number of committed blocks in their respective blockchain is greater than the last block read by the client, and if so, send the respective account balance and the number of committed blocks. When the client receives a quorum of responses it accepts the balance as a valid value and updates its last read value according to the messages received.

This way, we ensure that strong reads allow reads to execute faster than blockchain update operations while still obeying the same atomic (linearizable) semantics that are provided by the Istanbul BFT[1] protocol. In other words, we still ensure that when a client reads a value, the same client won't read an older value (a value that was committed before the one already read).

3.6 Snapshot Block Consensus

This new consensus will be made for a block that possesses the current state of the accounts (the accounts and their balance). Like the other consensus, this one can only be submitted by the leader and must be validated by the other servers. However, in order to allow the client the possibility of verifying the correctness of the output they received, whenever a server validates this state block from the leader, it will perform a signature using its own key pair, sending it when it broadcasts the replies. Whenever a server receives a new reply with a new correct signature, a valid one, it also appends it to its list of servers that have validated this state block. Once a server reaches a quorum of signatures, it appends the state of accounts received by the leader to the blockchain, but also the set of $2f+1$ correct signatures it collected during the consensus. Once a client requests a weak balance read, the server will check the last state block that was appended to the blockchain and return the state of the accounts and the signatures collected. Once the client receives the output, it verifies if the signatures are from known servers and match the state of accounts it received.

4 Threats

One of the other changes from the 1st stage to this one is that in this stage clients can also be byzantine. This means that a client can try to make transactions on behalf of another, for example.

To address this threat we make sure that every operation that is performed by a client is validated. To validate these operations, we use digital signatures that are appended to every operation request message. The signature consists of a message digest signed by the client with its private key, so, for a client to perform an operation it must have access to the private key of the account and this means that it is the owner of the account. To prevent replay attacks, messages have a NONCE field and, since servers reject duplicate messages, if an attacker were to intercept a message in the network and resend it, impersonating the victim, he would be unsuccessful as the server would discard this message.

Servers also sign every message they exchange, this means that attacks on behalf of byzantine servers are also dealt with accordingly.

References

1. Moniz, H. The istanbul bft consensus algorithm. *arXiv preprint arXiv:2002.03613* (2020).