1 Introduction

We present a blockchain system composed of N authenticated members that run the Istambul Byzantine Fault-Tolerant (IBFT) algorithm in order to implement state machine replication. We simplify the IBFT algorithm by setting a fixed leader that never crashes (but may behave arbitrarily). This simplifies the implementation while still ensuring Safety (by tolerating up to f Byzantine processes in the system, assuming that $N > 3f$. Fault tolerance is achieved by the use of Quorums of minimum size of $2f + 1$.

The permissioned blockchain requires its clients to authenticate with the use of assymetric cryptography in order to use it. Clients can interact with 'smart' contracts that exist in this blockchain, namely the Token Exchange System.

2 Design

The system is divided in six different layers:

1. **FairLossLink** - takes care of the lowest UDP level and sends any message to the corresponding destination

2. **PerfectLink** - Ensures sent messages are delivered exactly once by keeping track of ACKs and sequence numbers from each individual process that it communicates with.

3. **Authenticated Perfect Link** - Ensures a received message was sent by the source in the message, and also mitigates replay attacks with the use of timestamps in the messages. It provides to the upper layer (IBFT) the authenticated sender that sent the received message.

4. **IBFT** - Main consensus algorithm. Uses the lower layer APL for broadcasts, and the upper layer (App) to validate the value to be decided.

5. **Application** (Blockchain) - Stores all blocks, provides some functions to the IBFT layer to validate or decide consensus values and an API to communicate with the contracts that are stored in its state - Our implementation is open to the addition of new contracts. Its worth noting that we opted to execute transactions on the final phase of validation (when IBFT calls the final validate after getting the COMMIT quorum) at the same time we validate the transactions. We also introduced gas to the blockchain transactions such that they may be ordered byt heir gas value.

6. **Decentralized Applications** ('Smart' Contracts) - The apps than run on top of the blockchain and which the client can interact with using the blockchain API to send transactions - for example Token Exchangle System (TES). For the specific case of TES, we had to implement two type of reads - Strong (obeys to the the same atomic (linearizable) semantics that are provided by the Istanbul BFT protocol) and Weak (may return

an older but correct value and must work even when contacting a single blockchain member). Our solution lies in keeping the last decided block and a quorum of member signatures of that block stored on all client accounts affected by it. This way a client can contact a single replica and validate the signatures of the block, and compare the received balance as explained in Appendix A. Strong Read functions exactly the same way as when clients wait for response messages from any update operation - The client must wait on $F + 1$ equal responses from different processes.

3 Assumptions

The presented system was built under the following assumptions:

1. The system is partially synchronous.
2. The leader does not crash, but may behave arbitrarily.
3. The number of members on the blockchain is fixed.
4. There is only one consensus instance at a time.

4 Threats

We consider the following faulty behaviours in our system and explain the inherent threats and how the system mitigates them:

1. **Duplication** - Messages may be replayed.
    - **Link Layer** - Caught by using both sequence numbers and timestamps
    - **Application level** - Caught by using nonces for the transactions in the blockchain.
2. **Corruption / Byzantine** - Messages may be tampered or byzantine.
    - **Weak Reads** - Solved by including in the response of the read a balance proof (Appendix A.)
    - **Strong Reads / Updates** - Solved by waiting for the first $F + 1$ equal responses from different processes. Each message is also signed to ensure members cannot lie about their source. This way a byzantine member cannot force any other member to believe solely on him by sending $F + 1$ messages himself.
    - **Invalid block proposal** - Blocks can be proposed from a forked chain (and thus have a different hash, which will be validated by every member when they receive a valid PrePrepare from the leader where each member decides on whether or not to accept the proposal) or they can contain invalid transactions (transactions that were Rejected or not properly signed).

- **Transaction Validity** - Transactions carry with them a signature that signs all the transaction fields which is verified by the leader before he proposes a new block. This leader can be byzantine and thus validate transactions that are not properly signed by the clients, or add arbitrary transactions (which cannot be properly signed) which start being proposed in a new block all the way up until the commit phase is done. In the commit phase, by the time a member receives a valid commit quorum, the transactions are validated and executed, if valid. This validation changes a field in the transactions which will change its signature and the respective block's hash, which is verified to be equal to the hash of the blocks that are inside the commit quorum message.
- **Application / IBFT** - Solved by using signatures, and a agreeing only if received a valid quorum of COMMITS. In the consensus layer it is important to verify non-repudiation to ensure valid quorums and for this every consensus message carries with it a signature that signs the whole message (that includes in its content the block and the fields that are respective to the consensus instance).
- **Link Layer** - Solved by using signatures.

5 Dependability

- Confidentiality is not required given that clients submit a set of transactions to the blockchain
- Integrity is guaranteed in the messages by requiring the digital signature of every message and on the blockchain system by having in the current block a hash that includes the information of all the other blocks.
- Availability is only guaranteed until the leader acts byzantine. Once that happens, since there is no leader re-election, the system will not propose any new blocks if the leader does not follow the algorithm.
- Safety is guaranteed since IBFT provides that if a correct process chooses a value, that no other correct process can choose a different value.
- Reliability is guaranteed by our fault tolerance measure, as long as $N > 3f$
- Maintainability is assured by a proper design of the system, divided into different layers.

A Appendix - Weak Read Solution

We found that the way of believing a single member when requesting a read was by using a proof that contained a quorum of signatures from the other members (Figure 1 and 2). We do that by storing both the last decided block as well

as the quorum of member signatures and sending that to the client. Members also send the previous balance to the decided block so that clients may check the validity of that balance with the balance on one of the transactions from the proof sent - This implies that we store also the balance of the accounts in the transactions. To mitigate exposure of client balances around the blockchain we use instead the hash of the balance. The client then computes the hash of the received previous balance and runs it against one of hashBalance of one of the transactions that affected his account. This way a client can verify both that the received response is proven to be correct - has a quorum of signatures, thus was decided, and the value of the previous balance received can be checked against the balance in the transactions of the block. The client can then further use the transactions from the block (the ones that were validated) and compute the changes affecting his own account, getting the actual current balance.

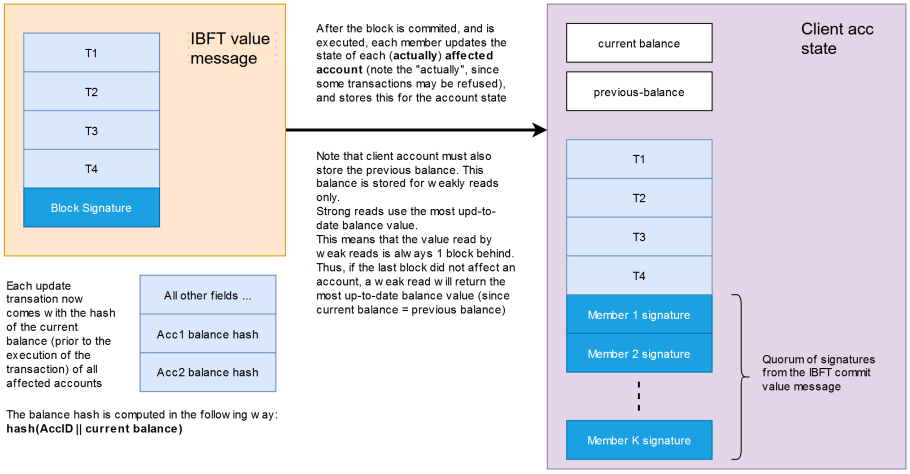
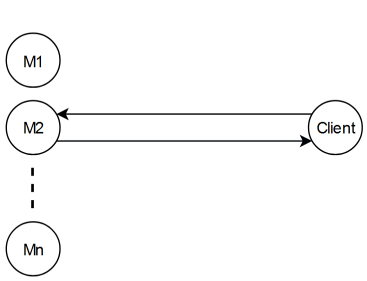


Figure 1: Weak Read Solution



When the client requests a weak read, the member provides the client the Previous balance, the list of the hash of all affected accounts' balance, and the quorum of signatures.

1 - The client then uses both its ID and the previous balance received and computes the hash.

2 - Then searches the list of all transactions, and for the first transaction that affected its account, the client compares the balance hash with the one he computed. If they differ, then the response message was tempered. Else, go to step 3.

3 - Compute the hash of the list of transactions.

4 - Check all signatures from the quorum.

5 - If all valid, the read is valid. Return to the client the previous balance.

Figure 2: Weak Read Execution