HDS Project Report - Group 29

Gonçalo Guerreiro 95581 | João Ramalho 95599 | Vasco Sebastião 95686

1. Broadcast

The broadcast algorithm used to communicate in the consensus is Best-Effort, similar to what is used for the client to send the append request to the blockchain. The reason for the group to choose this algorithm is because we implemented Perfect Links in the lower layer of the application, which will be explained further in the report.

2. Consensus Algorithm

The consensus algorithm implemented is the The Istanbul Byzantine Fault Tolerant Consensus Algorithm, namely, for this stage we implemented Algorithms 1 and 2, which the pseudo-code is featured in the paper of this consensus algorithm.

3. Perfect Links

The communication between the processes is done using UDP with two layers of communication abstraction built on top: Stubborn Links and Perfect Links.

The Stubborn Links are implemented using Datagram Sockets (which approximate the behavior of Fair Loss Links). Each Stubborn Link maintains a list of all the messages that have been sent but have not yet received confirmation (ACK) from the receiver. When a Stubborn Link receives a message, it checks if it is an ACK and, if so, removes the corresponding message from the list. These links also have another thread running which, every few seconds, re-sends all the messages in the list. This way, we guarantee that the messages keep being sent until it receives confirmation that they were delivered.

The Perfect Links are implemented using Stubborn Links. Each Perfect Link maintains a list of all the messages received. When a Perfect Link receives a message, it checks if it was already received before and, if not, it adds the message to the list. Repeated messages and ACKs are discarded, guaranteeing that only new requests are delivered. When a request is received, Perfect Links send an ACK back to the sender, confirming it was delivered.

4. Dependability and Security

Regarding this subject there are quite a few mechanisms implemented to ensure dependability of the system to the user.

Firstly, we made the library create a tuple that consists of the id of the client that sent the append request and the sequence number of the request. This tuple will be kept as a part of the structure of the message, even when being sent between members of the consensus, to ensure freshness to the protocol.

Then, for simplicity purposes we developed our solution using only 4 members for the consensus protocol, to ensure the minimum for a byzantine quorum is reached. And as mentioned we launch them from a config file. But, also, we created a directory holding all the asymmetric keys of each server used for digital signatures at the broadcast layer that are created at the broadcast layer of the application. This provides integrity, authenticity and non-repudation of the messages between members of the algorithm.

Lastly, when a client requests a new string to be appended, each member receives it and checks whether that request is already in the queue of strings already requested, or is in the blockchain, or is the request in the current consensus instance. Also, when receiving messages related to quorums, each member also checks if the respective sender of that message had already sent a quorum message to them, since there is only one instance of consensus running and the leader is always correct, it is suspicious behavior in this stage of the project if it receives two different messages from the same source in the same instance of consensus.