Distributed Fault Tolerance

Streamlet Consensus Algorithm (Phase 1)

Eduardo Proença 57551
Pedro Cardoso 58212
Ricardo Costa 64371

Prof. Alysson Bessani

Report of the Semester Project
MSc in Computer Science and Engineering

November 2024

Contents

Chapter 1

Phase 1: Implementation of the Protocol with Fault Tolerance

1.1 Introduction

The primary goal of this phase of the project was to implement the Streamlet protocol with fault tolerance (nodes can crash). Streamlet is a consensus protocol that operates under partial synchrony, tolerating faults and node crashes without compromising the blockchain's integrity. It allows nodes to propose, vote, and finalize blocks in the blockchain, ensuring that the blockchain remains consistent and synchronized across all nodes.

1.2 Objectives

The main objectives of this phase were:

- To implement the Streamlet consensus protocol with crash fault tolerance in a distributed environment, allowing nodes to propose, vote, and finalize blocks in the blockchain.
- To ensure the blockchain remains synchronized and consistent across nodes despite node crashes.
- To test the protocol's functionality with multiple nodes, validating its fault tolerance and consensus properties.

1.3 Protocol Description

The Streamlet protocol is a method for achieving consensus in a distributed system, particularly useful for implementing distributed ledgers like blockchains. It operates in epochs, each consisting of rounds, where nodes propose blocks, vote on blocks, and finalize blocks.

Protocol Overview

- The protocol progresses in a series of epochs, each with a designated leader node randomly selected using a hash function known to all nodes. This leader is the one who proposes new blocks to add to the blockchain.
- In each epoch, the leader proposes a new block based on the longest notarized chain it knows of. The block contains the hash of the previous block, the epoch number, the block length and the transactions.
- A block becomes notarized when it receives votes from the majority of distinct nodes ($> n/2$).
- Finalization occurs when a node observes 3 consecutive notarized blocks with consecutive epoch numbers in the chain, with the second of those three blocks and all its predecessors being finalized.

Why Streamlet?

- **Simplicity**: Streamlet uses a unified propose-vote paradigm, eliminating the need for complex recovery paths found in traditional consensus protocols like PBFT [1] and Paxos [2].
- **Partial Synchrony and Partition Tolerance**: Streamlet guarantees consistency even in partially synchronous networks with message delays. It remains consistent even if the network is partitioned.
- **Liveness**: During periods of synchrony where messages are delivered within a specific timeframe, the protocol achieves liveness with an expected confirmation delay.
- **Fault Tolerance**: In this implementation of Streamlet, nodes can crash and recover without compromising the blockchain's integrity, up until $n/2$ crash faults.

1.4 Implementation

We implemented this protocol in Python [3], across multiple modules for better organization and modularity.

Data Structures

- **Transaction:** Includes the sender id, receiver id, transaction id and amount.
- **Block:** Contains the previous hash, the epoch number, the length of the chain and the list of transactions in the block.
- **Message:** Contains its type (`PROPOSE, VOTE, ECHO`) defined as an enum, the content (`Block` or `Message`) and the sender id.
- **Blockchain:** Maintains the ordered chain of blocks and is responsible for adding new blocks and managing votes and finalization. This class was implemented to achieve more modularity and reusability, allowing for easy integration with the Node class. This also improves the code's readability, maintainability and testing.
- **Node:** Encapsulates each network participant, handling for the broadcasting of transactions, messages receiving and running the protocol, enabling each node to act independently while remaining synchronized within the distributed network. It uses the `Blockchain` class to manage the blockchain and the consensus protocol, as well as the other classes to manage data.

Other Modules

args.py Contains the program argument and the command-line argument parsing, which allows to configure for example the number of nodes to start and the epoch duration.

start_nodes.py Initializes multiple nodes and starts the protocol, simulating a distributed network environment in different processes, each with a console.

workload.py Simulates transaction workloads, generating random transactions and broadcasting them to the network.

main.py Main script that initializes the nodes and starts the protocol, simulating a distributed network environment.

1.5 Testing

TODO

1.6 Conclusion

TODO

Bibliography

[1] Barbara Liskov Miguel Castro. Pbft, 1999.

[2] Leslie Lamport. Paxos, 1998.

[3] Guido van Rossum. Python, 1991.

[4] Elaine Shi Benjamin Y. Chan. Streamlet: Textbook streamlined blockchains, 2020.

[5] Alysson Bessani. Project description, 2024.