DepChain - Stage 1

Simão de Melo Rocha Frias Sanguinho$^{1[102082]}$, José Augusto Alves Pereira$^{1[103252]}$, and Guilherme Silvério de Carvalho Romeiro Leitão$^{1[99951]}$

Instituto Superior Técnico, Lisboa, Portugal

Abstract. This project aims to develop a simplified permissioned blockchain system, named Dependable Chain (DepChain), with high dependability guarantees. The system is designed to be built iteratively, with the first stage focusing on the communication and consensus layer of a simple blockchain implementation, and the development of a client and a library that can interact with the blockchain system. The project leverages the Byzantine Read/Write Epoch Consensus algorithm, with simplifying assumptions such as static system membership, a predefined leader process, and a Public Key Infrastructure (PKI). For message communication, the implementation will use authenticated perfect links, with the assumption that the network is unreliable: it can drop, delay, duplicate, or corrupt messages, and communication channels are not secured. The implementation is structured in Java, utilizing the Java Crypto API for cryptographic functions, and is designed to handle malicious behavior from a subset of blockchain members while ensuring safety and liveness under the assumption of a correct leader. The final submission for stage 1 includes a self-contained zip archive with the source code, demo tests, and a concise report detailing the design, threats, and dependability guarantees of the system.

Keywords: Blockchain · Byzantine Fault Tolerance · Consensus Algorithm · Dependability · Java Implementation

1 Introduction

This report presents the design and implementation of a Byzantine Fault Tolerant (BFT) blockchain service, designed to withstand malicious behavior from a subset of blockchain members and operate reliably in an unstable network environment. The system is resilient to arbitrary (Byzantine) behavior from faulty nodes and can handle unreliable network conditions, including message drops, delays, duplication, and corruption, without relying on secure communication channels. To achieve consensus in such adversarial conditions, the project uses the Byzantine Read / Write Epoch Consensus algorithm, as described in the course book [1] (Algorithms 5.17 and 5.18). The report outlines the system architecture, which includes the network, client, library, blockchain, and consensus layers. It also describes how the system addresses various Byzantine attack scenarios, ensuring safety and liveness under the assumption of a correct leader. Finally, the report concludes with key findings, lessons learned, and potential areas for future improvement.

2 Architecture

2.1 Network Layer

The network layer is responsible for managing the communication any two processes (members or clients) in the system. By replicating Authenticated Perfect Links, the network layer ensures that messages are guaranteed to be eventually delivered to the intended recipient, with it's integrity and authenticity preserved. There are three main components in the network layer:

- Message, which encapsulates the message content and metadata. It contains the type field to identify the message type (e.g., READ, STATE, ACK, etc.). It's also this component that will be signed by the sender and verified by the receiver.

- PerfectLink, which implements the core communication logics, including sending, receiving and managing sessions (explained next). To simulate the unreliable network, we use UDP sockets.

- Session, which represents a communication session between two processes. The session contains information like the destination process ID, address, session key, and counters for tracking sent and acknowledged messages. The session key is used for encrypting and signing messages (because using the public key for every message would be too expensive).

Session Establishment Before any communication can occur, a session must be established between two processes. The session establishment process is as follows: A process initiates a session by sending a STARTSESSION message. Then, the recipient responds with a ACKSESSION message, containing an encrypted session key. Once the session is established, all subsequent messages are signed and verified using the session key to ensure authenticity and integrity.

2.2 Client and Library Layer

The client layer and library layer work together to enable clients to interact with the distributed system, specifically to append messages to the blockchain. The client layer handles user interaction and initialization, while the library layer manages the communication logic with the system's leader process.

Normal Workflow A user will issue the append <message> command via the client's CLI. The client layer will delegate the request to the library layer. Then, the library will construct a CLIENTREQUEST message and sends it to the leader process (known beforehand) using PerfectLink. Next, the library will wait for CLIENTREPLY from F+1 processes.

2.3 Blockchain Layer

liquam facilisis ante lacus, at scelerisque libero iaculis non. Etiam ex velit, iaculis blandit tristique ac, imperdiet a lectus. Etiam condimentum pharetra lectus non elementum. Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Integer imperdiet urna ipsum, nec tempor libero tincidunt ut. Proin in sapien arcu.

2.4 Consensus Layer

Proin sed vehicula magna. Cras a iaculis velit, eu fermentum nisl. Pellentesque at magna in massa scelerisque suscipit sed ac tortor. Vivamus a bibendum leo, eget blandit felis. Donec vulputate ultrices dignissim. Donec vel elit a massa pellentesque euismod eget at velit.

3 Implementation details

3.1 Detail XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

3.2 Detail XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

3.3 Detail XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

3.4 Detail XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4 Possible threats and corresponding protection mechanisms

To demonstrate the system's resilience against various Byzantine scenarios, the implementation was tested under multiple configurations, each representing a different type of attack. The following sections provide a detailed explanation of how each attack is executed and how the system effectively mitigates it.

4.1 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.2 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.3 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.4 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.5 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.6 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.7 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.8 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.9 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.10 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

4.11 Threat XYZ

Cras quis bibendum erat. Aliquam massa tortor, euismod a venenatis eget, convallis ut odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

5 Conclusion

Donec vulputate ultrices dignissim. Donec vel elit a massa pellentesque euismod eget at velit. Aliquam eget est et urna auctor sodales. Aenean vulputate finibus libero ac pretium. Etiam lacinia ultrices odio, vitae bibendum metus accumsan sed. Etiam sit amet condimentum eros, pulvinar pharetra nisi. Phasellus ac purus a libero euismod ultrices. Duis feugiat, mi id facilisis blandit, magna magna commodo ante, ut porta lacus justo ut neque. Donec eget nisl feugiat, pretium libero eu, mollis dolor.

References

1. Christian Cachin, Rachid Guerraoui, Luís Rodrigues: Introduction to Reliable and Secure Distributed Programming. 2nd edn. Springer, 2011
2. Java Crypto API, https://docs.oracle.com/javase/8/docs/api/javax/crypto/package-summary.html