Department of Informatics
King's College London
United Kingdom

7CCSMPRJ MSc Project

A FRAMEWORK FOR BLOCKCHAIN INTEROPERABILITY

This dissertation is submitted for the degree of MSc in Computational Finance

Abstract

Distributed Ledger Systems such as Blockchains have grown in prominence in recent years, both in businesses and academics. Today, Blockchains are not just used to power cryptocurrencies such as Bitcoin but have also been identified as a potentially disruptive technology in various other areas, including supply chain monitoring, digital voting, notary services, and healthcare. The increasing interest in blockchain technology has resulted in a plethora of research and development initiatives. Due to the inherent trade-offs between functionality, privacy, and efficiency, no one blockchain can be effective in all scenarios. Instead, Interoperability is necessary to connect disparate blockchains via the exchange of data and assets. Consequently, the current state of the blockchain ecosystem is highly fragmented, with consumers having access to a variety of incompatible technologies. Since Interoperability across different blockchains is not envisaged in current protocols and standards, functions such as transferring cryptocurrency between users or invoking and executing smart contracts may only be performed inside a single blockchain. The purpose of this project is to explore the importance of blockchain interoperability and how it may assist in promoting a transition from today's closed blockchains and toward an open ecosystem in which participants can communicate across blockchain borders. This project proposes a Python API which enables cross-chain communication; the user can insert and fetch data across six different blockchains. To test the viability of the implementation and analyse design choices made during development, the framework was assessed on its Performance, Security and Transaction Data size.

Nomenclature

API	Application Programming Interface
BFT	Byzantine Fault Tolerance
dPoS	Delegated Proof-of-Stake
HTTP	Hypertext Transfer Protocol
PoA	Proof-of-Authority
PoET	Proof-of-Elapsed-Time
PoS	Proof-of-Stake
PoW	Proof-of-Work
RPC	Remote Procedure Call
HTLC	Hashed Timelock Contract
SCP	Stellar Consensus Protocol
SQL	Structured Query Language
SPV	Simplified Payment Verification
IoT	Internet of Things

Contents

1. Introduction **10**

1.1 Motivation 11

1.2 Scope 11

1.3 Aims 12

1.4 Objectives 12

1.5 Report Structure 12

2. Background **13**

2.1 Definitions 13

2.2 Literature Review 17

3. Design **20**

3.1 Solution Methodology 20

3.2 Requirements 20

3.3 Specifications 21

3.4 Process 23

3.5 System Architecture 24

3.5 Technology Stack 25

4. Implementation **26**

4.1 Overview 26

4.2 Application Programming Interface (API) 26

4.3 Blockchains 26

4.4 Database 30

4.5 User Interface 31

4.6 Implementation Issues 31

4.7 Testing 32

4.8 Limitations 35

5. Evaluation **36**

5.1 Results and Analysis 36

5.2 Comparison with Existing Solutions 39

6. Legal, Social, Ethical and Professional Issues **41**

7. Conclusion & Future Work **42**

7.1 Conclusion 42

7.2 Future Work 42

8. References 44

9. Appendices 48

Appendix A: Extra Information 48

Appendix B: User Guide 51

Appendix C: Source Code 52

List of Tables

Table 1: Specifications 22
Table 2: Transaction Data size 39
Table 3: Comparison of proposed framework with existing solutions 40

List of Figures

Figure 1: Interoperability API flowchart 23
Figure 2: General Architecture of the Framework 24
Figure 3: Transaction Latency Comparison 36
Figure 4: Memory Usage Comparison 37

1. Introduction

The idea of blockchain came into existence in 2008 with the release of Bitcoin [1]. Initially, blockchains have been chiefly perceived as the underlying technological medium to carry out monetary transactions in a completely decentralised manner, thus enabling cryptocurrencies. Bitcoin was released to facilitate peer-to-peer online payments [1]. In contrast, Ethereum aimed to execute almost capricious software functionalities within the blockchain, using so-called smart contracts [2]. Bitcoin surpassed the total market capitalisation of $100 billion in 2017, a remarkable feat for a cryptocurrency [3]. Following Bitcoin's launch, many cryptocurrencies were invented to capitalise on blockchain technology and Bitcoin's success. As of January 2021, more than 4000 cryptocurrencies are in circulation, some of which follow the Bitcoin blockchain protocol, but many use their blockchain protocol and implementation [4].

A blockchain network is distributed and tamper-proof; it uses cryptography to ensure zero dependence on a centralised intermediary to execute and store transactions within such systems. Blockchains have the potential to be used in a variety of fields due to their features. These sectors include general industrial applications to more specific use cases in BPM, anti-counterfeiting, and healthcare. To sum it up, blockchain networks can cope with any situation which requires the storage of any arbitrary data in an entirely Decentralised manner. [5]

Several blockchain networks have emerged over the years, with each focusing on solving specific limitations of the first and most prominent blockchain project - Bitcoin. On the one hand, some implementations address the scalability problem by improving the transaction throughput time (TPS) by optimising transaction-relevant features [6]; however, this could cause centralisation. Some implementations, on the other hand, focus on specific characteristics like smart contract support [2], privacy [7], or digital identities [8]. Others enable businesses to construct private blockchains in which only a limited number of companies can join [9]. Each of these implementations is accompanied by its cons. For instance, a higher level of decentralisation leads to increased transaction confirmation times, and privacy limits the identity features. As a result, blockchain initiatives must establish a vision and focus on a specific set of applications to get traction in the market. Considering the broad spectrum of use cases discussed in [10], it is unlikely that one prominent blockchain will exist, which perfectly accommodates the needs of every use case.

The isolation of blockchain networks is a significant issue that has arisen as such networks have grown. As a result, providing a medium to bridge the gap between disparate blockchain networks would undoubtedly significantly influence users. Users would pick and combine blockchains based on their current needs rather than being confined to a single platform. However, the methods in which disparate blockchains could interact with one another are largely unknown. Most importantly, the following tasks can only be carried out within a single blockchain at the moment: Transferring Tokens between Participants, Execution of Smart Contracts, and Assuring the integrity of data stored on a Blockchain. The underlying complex protocols and technology used by cryptocurrencies means that they cannot be traded between other cryptocurrencies, nor can their current state or recent events be communicated. As a result, blockchain interoperability is a potential field to resolve such a challenge. [11]

The capability to connect different blockchain networks in order to exchange information and take action on that information while keeping the blockchain trustless is known as interoperability. Interoperability also refers to the objective of a user (e.g., an application developer) working with any blockchain without limits in a more realistic and user-focused environment. Users must hold a deep understanding of underlying technology to communicate with different blockchains, which is a considerable limitation. However, if blockchains and cryptocurrencies become more extensively utilised and prevalent in everyday life, this complexity must be concealed from users and replaced with user-friendly interfaces.

As a result, this project presents a method for developing an intuitive interface to interact with various blockchains. The project's primary contribution is a cross-chain interactive system framework proposed to address Blockchain Interoperability. This framework provides an easy-to-use interface for participants to connect with various private and public blockchains through a Python API. It allows users to insert and fetch data on a particular blockchain without understanding the underlying complex blockchain architecture. It offers this functionality by employing an interoperability method called the "Notary Scheme", discussed in Section 2.1.5. The framework is hosted on a trusted server that acts as an interface for software developers and consumers in this approach. Finally, the proposed framework is analysed, including comprehensive testing to determine its performance and security efficacy. The results demonstrate the effectiveness and efficiency of the framework.

1.1 Motivation

The motivation for this project was drawn out of sheer interest in understanding the concept of Blockchain Interoperability. Interoperability is one of the three main future development areas (besides Scalability and Privacy) in Blockchain. With the increasing popularity of cryptocurrencies, there is a need for significant innovation in Blockchain technologies to bring it to the next stage. Although extensive research has been done to address Scalability and Privacy issues in the Blockchain, the Interoperability problem remains mostly unexplored. Blockchain Interoperability is critical since it will make it easier for individuals to transact on other blockchains seamlessly. A user can only transact on one Blockchain at a time, for example, Bitcoin or Ethereum. It is, however, impossible to transmit data across two separate Blockchain Networks. Because of Interoperability, a variety of functionalities should emerge. Participants will be able to make payments across several blockchains, for starters. A fully functional blockchain interoperability project should be at the centre of the digital economy in the future. Due to the emergence of multi-token wallet systems, blockchain interoperability should also lead to multi-token transactions. Due to this breakthrough, users will depend on a single wallet system for token storage and transfer across several blockchains. To sum it up, Interoperability across different blockchain networks is necessary to achieve higher economic value and potential.

1.2 Scope

This project addresses the Blockchain Interoperability Problem and proposes a method for interacting with numerous blockchains with ease. It studies the current state-of-art in Blockchain Interoperability, taking into account theoretical work and active efforts in the field. The primary motive of this project is to implement and evaluate an Interoperability Application Programming Interface (API) that allows interaction between multiple Blockchains.

1.3 Aims

Given the ever-increasing necessity, many developers are already working on ideal methods to allow cross-chain communication between multiple blockchain networks. Since developers want to expedite blockchain mainstream adoption, blockchain interoperability initiatives are on the rise. However, almost all of them require a deep understanding of the underlying complex protocols. This project aims to provide a simple Python API that allows participants to communicate with numerous private and public blockchains. It will enable the users to insert and fetch any data (in the form of a text message) on a specific blockchain. Lastly, the interoperability API will be tested and evaluated, including measurements of the various blockchain connectors and discussions of features such as privacy and centralisation.

1.4 Objectives

The primary objectives in the development of the project are:

- Study the existing research papers/journals on Blockchain Interoperability in order to gain a deeper understanding of this research area and define the problem scope.

- The acquired knowledge will be used to provide an overview of related work and state-of-the-art techniques in the field of blockchain interoperability. Additionally, many intriguing initiatives will be highlighted.

- Development of a Blockchain Interoperability Framework that allows cross-chain communication.

- Performing testing and evaluation to ensure that the developed framework serves the intended purpose and fulfils the specified requirements.

1.5 Report Structure

This report is organised in the following manner. Chapter 2 provides background information necessary to understand the full scope of the proposed solution. This chapter discusses the fundamentals of blockchain technology, with a focus on consensus protocols. Additionally, it describes the blockchains selected for the framework and their primary characteristics. It then goes on to classify interoperability use cases and methods. Finally, it discusses relevant work in the field of blockchain interoperability, including state-of-the-art techniques and initiatives. Chapter 3 delves further into the project's Requirements and Specifications. Chapter 4 is the Design phase, in which the abstract method is described, accompanied by comprehensive diagrams. Chapter 5 is the Implementation phase, in which the Interoperability framework and the rationale for the design are given. Chapter 6 is the Evaluation phase, in which the framework is analysed for performance, security, and centralisation, and the implications of design choices are discussed. Chapter 7 will examine the Legal, Social, Ethical, and Professional Issues that arose throughout the project and how these issues were addressed per the British Computer Society's Code of Conduct and Code of Good Practice. Finally, Chapter 8 brings the dissertation to a close and provides a perspective on future work.

2. Background

This part of the report deals with the summary and analysis of the background for this project. It includes a brief insight into the world of Blockchain Interoperability.

2.1 Definitions

This section defines key terminology and concepts that will aid in understanding the ideas presented in the subsequent chapters and gaining a better grasp of Blockchain Interoperability.

2.1.1 Blockchain

A blockchain, also referred to as a distributed ledger, is a decentralised database that records and interconnects data blocks to enable data recovery and verification that it has not been altered. The data blocks are connected through hash pointers that link the current block to the preceding block until the block genesis is reached. The blockchain is maintained by all of the network's nodes that are in sync with it. Mining, analogous to gold mining, refers to the process of creating new blocks. Each block in the blockchain includes information on the transactions that occurred within a specific period, the preceding block's cryptographic address (hash key), and an arbitrary unique number (nonce). Each transaction includes the sender and recipient's digital signatures. The transactions are verified by "Validators" on this distributed network, who do not rely on trust. "Consensus Mechanism" is the name of the protocol that these validators utilise to verify transactions on the blockchain. [12]

2.1.2 Interoperability

When interoperability is discussed, it refers to creating a causal link between two blockchains' occurrences (isolated systems). There are two potential models for establishing connections between correlated events (the idea of interoperability) [15]: causation and dependency. Assume there are two blockchains, A and B, that need to communicate with one another:

1. Causation: When one blockchain experiences an internal event, another blockchain experiences a linked internal event. There are two types of causation: Forward causation and Reverse causation. Forward causation is when an event X on A affects event Y on B. Reverse causation is when an event C on B has an effect on event D on A.

2. Dependency: An external event results in a pair of internal events on both A and B. (i.e. both blockchain events correlated by the same external event). For instance, an external event Z leads to event X on B and event Y on A.

A dependence relationship may be sufficiently expressive in some circumstances and applications, for instance, an oracle transmitting the same data stream to various blockchains. When the term "Blockchain Interoperability" is used in the context of this project, it means that there exists a method for establishing causation between transactions sealed on disparate blockchains. [43]

2.1.3 Node Provider

To enable Blockchain nodes to communicate with the overlay that defines the network, a Node Provider is necessary. It enables a blockchain to function by allowing nodes to execute transactions. Standard components of this programme include a code repository, an API documentation, and a digital wallet. Developers use nodes to develop blockchain-oriented applications. Furthermore, using HTTP or RPC, interfaces may be created to interact with the blockchain remotely [13]. Full nodes, Mining nodes, and Light nodes are the three distinct types of nodes. Full nodes keep track of every block in the chain and verify that transactions follow the consensus protocols. They ensure that no duplicate transactions are carried out inside a block, as well as verify that the signatures match the transaction's source [1, 13]. Miner nodes can propose new blocks and have access to the blockchain's entire history [1]. Unlike full nodes, light nodes simply examine the "block headers" of the longest chain and don't need to store or validate every transactions. To further confirm information recorded on the blockchain, light nodes are linked to full nodes. Different blockchains have different implementations and functionality for light nodes. A method known as Simplified Payment Verification (SPV) is often utilised.

2.1.4 Consensus Protocols

A *consensus protocol* is a method through which all participants in a Blockchain network achieve a consensus on the current state of the Blockchain. Consensus algorithms enable blockchain network resilience and build trust amongst unknown participants in a distributed computing environment. In essence, the consensus protocol ensures that every new block "added to the Blockchain is the only version of the truth that" all nodes in the Blockchain agree on. The Blockchain consensus protocol has particular goals: reaching an agreement, coordination, cooperation, equal rights for all participants, and each user's required involvement in the consensus process. As a result, a consensus algorithm seeks to identify a common ground that benefits the whole network. Some of the most popular consensus protocols are Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Authority (PoA), Byzantine Fault Tolerance (BFT) and Delegated Proof-of-Stake (DPoS). [14]

One of the earliest consensus protocols utilised in blockchain networks was **Proof-of-Work**. This protocol uses a concept called mining to overcome the consensus issue. A miner wins the privilege to add a new block by solving a complex mathematical puzzle that requires significant computational power. If other participants accept the newly formed block to be legit, the miner gets paid in cryptocurrency.

Proof-of-Stake is another consensus protocol, which is becoming the more common alternative to PoW. This mechanism solves the energy consumption issue of PoW blockchains since there is no usage of expensive hardware. Instead, Minters invest in the cryptocurrency by locking up part of their tokens as a stake. Minters will verify blocks by betting on them if they find one that they believe can be added to the ledger. Minters get a return proportional to their bets based on the actual blocks added to the Blockchain, and their stake increases correspondingly. Finally, depending on their economic stake in the Blockchain, a Minter is picked to create a new block. As a result, PoS encourages participants to achieve a consensus via an incentive mechanism.

In private blockchains, **Proof-of-Authority** is often employed. It has been tuned for speed and minimal hardware requirements. A fixed set of block producers and verifiers are established in PoA blockchains. The downside of this protocol is that it is centralised. The selected block producers must be trusted, or adverse behaviour must be discouraged by utilising a reputation system. Because a failure of the authorised nodes might interrupt block creation, this strategy prioritises safety above liveness.

2.1.5 Interoperability Techniques

This section discusses the three main techniques for achieving Blockchain Interoperability.

A trusted agent acts as a mediator between different blockchains in **Notary Scheme**. The notary's job is to verify that an event occurred on blockchain A and then transmit that information to blockchain B. The key benefit of the Notary Scheme is its flexibility, since no modifications are necessary to the supported blockchain. Atomicity is another crucial element of the Notary Scheme; in other words, a transaction has to be accepted by all users as a whole. The drawback is the necessity for trust in the notary. One option is for both parties to choose a group of notaries they can trust. The output of the notaries might then be formed using consensus protocols like BFT. As a result, not every individual notary would need to be trusted, but just two-thirds of the group would. [15]

A **Sidechain** is a kind of "integration" blockchain that allows value and information to be exchanged across other blockchains. The integration layer in the sidechains-relay paradigm is the relay chain. For multiple sidechains to communicate with and share information, the relay chain can provide smart contracts. Furthermore, it typically offers more validation nodes than any of the sidechains, resulting in more robust security by ensuring that no value trades are double-spend. The integrity of the information is maintained as long as it is part of the overall "block header," which is (i) created in a cryptographically verified manner, most likely using Merkle trees, and (ii) recorded on the relay chain. In terms of data communication, a relay chain smart contract may work as a light client for a particular sidechain, allowing it to use the sidechain's standard verification mechanism to check any block headers provided into the contract. Asset mobility, atomic swaps, and various other more complicated purposes are all possible using relays. [16]

Hash-locking is a method of triggering an action on two blockchains simultaneously in an atomic manner, with both actions or none of them taking place. Without the involvement of a third party, hash-locking may be utilised to effectuate transactions between two or more participants. Because hash-locking techniques may be chained after each other, transactions can be made even if the trading parties do not have a direct link. Most decentralised exchanges nowadays employ this strategy. Unlike the notary method, Hash locking does not need a prior trust connection between the two blockchains or a third party. [15]

2.1.6 Interoperability Use Cases

Interoperability may be used for various purposes, including asset mobility, atomic swapping, cross-chain oracles, asset encumbrance, and cross-chain contracts. Asset portability enables users to transfer their tokens across blockchains. This connection should be bidirectional, enabling the asset to be retrieved at any time. Atomic swaps allow the exchange of two tokens from different blockchains between two parties. Both parties must have a blockchain account or address on the other's blockchain. Following that, transactions are processed on both blockchains concurrently. The requirement is that either both both transactions occur or none occur; this is referred to as a "atomic" property. Atomic swaps are often used in decentralised commerce. An oracle feeds a blockchain with external data. A cross-chain oracle collects data from an external blockchain. For instance, an oracle may collect transaction data from an external blockchain and perform a smart contract function. Asset encumbrance refers to a blockchain's capacity to lock and unlock an asset when a condition on another chain is met. This may be used for a variety of purposes, including security deposits. Cross-chain contracts provide smooth interoperability across blockchains with distinct chains. For instance, a smart contract may potentially pay dividends based on user ownership on another blockchain. This is the broadest use case, and therefore enables many of the other use cases.

2.1.7 Chosen Blockchains

This section gives a summary of the blockchains used in the framework as well as their characteristics.

In 2014, **Ethereum** was introduced as the first Turing Complete Blockchain, which Bitcoin lacked in. As a result, Ethereum's most significant feature is the ability to build distributed applications via the use of "smart contracts”. Ethereum suffers from the same inefficiency concerns as Bitcoin since it is based on the PoW mechanism [24]. Ethereum's smart contract capabilities enable the development of new crypto-currencies. ERC20 is a set of rules and functions that may be used to generate a token on the Ethereum platform. Ethereum features a 15-second transaction confirmation time and probabilistic block finality. According to the white paper, seven confirmations are suitable for waiting [2].

Stellar is a cryptocurrency based on open-source technology that converts digital currency to fiat cash domestically and cross-border. The primary goal is to offer developing nations a reliable and secure payment mechanism. Stellar employs its Consensus Protocol which enables high transaction throughput time (3000+ per second) and helps in keeping transaction costs to a minimum while retaining a high degree of decentralisation. Stellar supports smart contracts but is not Turing Complete, limiting it to small-scale applications. Because of their SCP consensus protocol, Stellar features finality and has a transaction confirmation time of 3-5 seconds [26].

EOS bills itself as "the most powerful infrastructure for decentralised applications" and has been referred to as the "Ethereum Killer" [27]. EOS addresses the problems of PoW by combining dPoS and BFT. As a result, EOS offers a fast transaction throughput time, no transaction fees for basic coin transfers, and uses very little energy. Block finality and an average transaction confirmation time of 0.25 seconds are also features of EOS [6]. The centralisation caused by dPoS, with just 21 block producers and the potential to buy votes, is the true grievance levelled against EOS [28].

IOTA was designed to facilitate decentralised transactions for the IoT. While IOTA is not a blockchain in principle, its features are somewhat comparable. Instead of blocks, it uses an acyclic directed graph termed a "tangle." By replacing Bitcoin's Blockchain with Tangle, it hopes to overcome critical scalability and performance difficulties. While IOTA does have its scalability issues, there were also several security vulnerabilities associated with this Blockchain. The key benefit of IOTA is that it has no transaction fees and can handle many transactions at once. While it does not enable smart contracts, it does support applications running on decentralised peer-to-peer networks. One minute blocks and complete block finality are both presently hardcoded into the IOTA network. Once the coordinator is removed, the finality of the system would be uncertain [29].

Hyperledger Sawtooth is an open-source enterprise blockchain-as-a-service platform that allows users to execute bespoke smart contracts without understanding the underlying system's complex architecture. It will enable developers to pick aspects such as the consensus protocol based on the specific use case, thanks to a strong emphasis on modularity. It is default consensus mechanism is Proof-of-Elapsed-Time (PoET). Sawtooth also has multi-language support for writing smart contracts. Block finality exists in Sawtooth. The randomised waiting time has an impact on the block time. However, with a transaction confirmation time of 20 seconds, a desired waiting time may be defined. [30]

Multichain is another blockchain development platform. Its benefits include easy deployment of private blockchains and configuration choices such as consensus, block timing, and node permissions. It also comes with a pre-configured "Bitcoin-like" blockchain that uses the same interface as the Bitcoin Client. Multichain has no smart contract support, although "Smart Filters" provide some limited capabilities. Multichain is a PoA blockchain with finality and a 15-second transaction confirmation time. [31]

2.2 Literature Review

Blockchain Interoperability is both a fundamental and a vital research area. Due to this, several projects addressing this problem have been undertaken. In this section, the existing Blockchain Interoperability projects are researched and discussed.

Herdius is a "decentralised exchange platform" based on the Notary Scheme as discussed in 2.1.4. Herdius could be considered an intermediary layer between participants and root blockchains of all relevant cryptocurrencies. This way, Herdius allows for interaction between disparate Blockchains through sharing them. The private keys of both (sending and receiving) parties are encrypted with a private key generated within Herdius to achieve decentralisation; this key is a "Schnorr Signature" and is sliced up into pieces. These pieces are then distributed among the nodes in the network and reassembled whenever required using a multi-signatory threshold signature scheme. The assembler node is the one that collects the pieces and assembles them to sign a transaction and broadcast it to the main chain. By merging their private key pieces, several assembler nodes may sign a transaction on behalf of the participant. Additional security is implemented by making it impossible for the assembler node to fully decode the native private key. Rather than that, homomorphic encryption algorithms are used to sign the transaction. [17]

Aion offers a platform that relies on both a notary scheme and sidechain to design an interoperable network. By establishing a blockchain that supports bespoke blockchain networks, Aion offers a trustless method for cross-chain communication. Without the need for a centralised message broker, interoperability is accomplished via so-called bridges and connections between blockchains. Aion's designers envisioned a layered structure on top of blockchains to realise true interoperability. It establishes a communication protocol that a blockchain may use to join the Aion network. [18]

Cosmos is the most under-appreciated interoperability initiative for blockchains. It has made an effort to make itself interoperable since the release of its consensus mechanism, termed Tendermint, in 2014. The Cosmos network comprises numerous separate blockchains known as Zones linked to a central blockchain known as the Hub. Each zone in this example is powered by Tendermint, based on the Byzantine Fault Tolerant (BFT) consensus protocol that offers excellent performance, stability and security. Cosmos takes advantage of what they call its "Inter-Blockchain Communication Protocol". This protocol involves the use of sidechains to verify transactions from other blockchains, such as Ethereum. Because of this protocol, participants can transmit tokens from one zone to another in real-time and securely without the need of any intermediary; it also enables Blockchain to link various zones, such as public and private projects. [19]

Polkadot is a different interoperability project with a similar premise to Cosmos. It uses required validators and the DPoS algorithm, which can result in a certain level of centralisation. Polkadot comprises many parachains with different features and allows communication between its relay chain and parachains. Additionally, by using numerous parallel chains, the scalability problem can also be tackled; this makes it different from other projects attempting to bridge the gap between blockchains. Because of the large number of chains on the Polkadot blockchain network, transactions can spread across a larger region. All of this is done while maintaining data integrity and security. Cosmos does not discriminate against any blockchains if their protocol is supported, but on the contrary, a significant quantity of the Blockchain's token "Dots" must be staked to join Polkadot. This way, Polkadot has greater control over their partner chains due to their ability to penalise harmful behaviour. [20]

The **ARK** Network consists of blockchains connected by a "SmartBridge" that enables cross-chain communication. The SmartBridge is a sidechain that connects various blockchains. ARK is an open-source framework that enables enterprises to develop their blockchains. ARK participants can utilise the network to execute transactions in various currencies and tokens, including ARK's native currency. DPoS is employed to reach consensus in ARK. The time required to create a new block on the ARK network is eight seconds, compared to 10 minutes to create a new block on the Bitcoin blockchain. ARK also includes a feature known as Push Button Blockchain Deployment, which lets users develop and launch a blockchain in just a few steps. This newly developed Blockchain would be linked to the ARK ecosystem through the SmartBridge and operate identically to the main Blockchain. ARK provides a simple and easy-to-use interface for those interested in creating their interoperable blockchains. Another feature to be added is their intended integration of cryptocurrency exchanges (Shapeshift) to convert between different cryptocurrencies, creating a Notary Scheme. [21]

Wanchain bills itself as the world's first interoperable online blockchain system, boasting safe multi-party computing and HTLC capabilities. Additionally, Wanchain aims to reconstruct finance by centralising all digital assets on a single blockchain. By employing cutting-edge cryptography research, the blockchain interoperability project assures cross-chain capabilities. Additionally, it utilises a unique protocol to link private, public, and consortium chains using storeman nodes and the T-Bridge architecture. The storeman node system combines two cryptographic ideas that assure the security and privacy of network transactions: safe multi-party computing and "Shamir's secret sharing." The connectivity enables the easy transfer of digital assets between two disparate blockchains. They utilise "Locked Account" instead of the HTLC. This strategy is used to establish a temporary account that will be locked throughout the two transactions. It does this by segmenting the account's private key and distributing it over various Wanchain nodes. The nodes will transfer funds from the locked accounts in a coordinated manner and will guarantee that transactions are atomic. Wanchain's Blockchain, which is based on Ethereum, also supports the implementation of smart contracts. These characteristics combine to make it an attractive blockchain option for developing distributed applications that need simple access to several blockchains. On the Blockchain, privacy is strengthened by the use of ring signatures and one-time stealth addresses. [22]

Interledger is an open protocol for transmitting payments across several blockchain networks and was initially inspired by the Internet Protocol. Interledger enables the transfer of XRP to someone requesting ETH or USD to someone requesting EUR. It enables the transfer of funds across multiple ledgers through connectors and has low requirements for the underlying ledgers. Unlike many of the initiatives mentioned above, Interledger is not tied to any blockchain, token, currency or central authority since it does not feature identity management, liquidity management, public key infrastructure, or other services. Rather than that, it leverages a network of connectors to facilitate decentralised trades by linking HTLCs across many blockchains. If none of the two ledgers supports HTLCs, time-based multi-signature is employed. Another critical aspect of Interledger is that it lacks native tokens, which means that any two ledgers wishing to join must support Interledger to allow cross-chain communication. [23]

The existing projects discussed in this section demonstrate that several efforts are being made to address Blockchain Interoperability. Although their designs vary, most of them use the sidechain technique. Sidechain is typically the most promising strategy in terms of the range of possible use cases. However, it often necessitates adjustments to blockchains' underlying architecture since the majority of blockchains do not allow the locking or burning of coins. To circumvent these constraints, several initiatives use a mixture of different techniques. Interledger, for instance, makes use of both sidechain and notary-scheme. Most initiatives prioritise interoperability inside the network they are developing rather than facilitating interoperability across existing blockchains. Given that most of the initiatives discussed here are still in their infancy, their strategy and methodology are expected to evolve as interoperability research advances.

The proposed framework offers a unique approach to Blockchain Interoperability; it is based on a Python API, which is intuitive, versatile and modular. The projects mentioned in this section do not share all of the proposed framework's features. For instance, Herdius needs several assembler nodes to sign a transaction, complicating the user experience by requiring these nodes to be configured and constantly accessible. The versatility of projects like Cosmos, Aion, Ark, and Polkadot is limited because of the issues with modifying the underlying blockchain implementations. Although Interledger is built using a modular architecture, it only works with Blockchain's supporting Hashed-Timelock Contracts. Wanchain enables cross-chain communication; however, it does it via another Blockchain that must support hash locking methods. Thus, Interledger and Wanchain both need measures that limit the system's versatility. In this regard, the proposed framework is distinct from previous Interoperability initiatives in that it seeks to offer all of these features together.

3. Design

This chapter presents concrete requirements and specifications of the proposed framework followed by design decisions and documents the preliminary planning for the development of the framework. A Gantt chart (attached in appendix A.6) illustrates when the milestones will be achieved.

3.1 Solution Methodology

The overall goal of the project is to develop a framework to enable Blockchain Interoperability. In order to correctly implement the framework, several steps have to be carried out in order. The first step of this methodology is to use the literature survey (carried out in Section 2.5) to develop a suitable and general process design to model the Blockchain Interoperability Framework. As a result, the procedure must be slightly broader in scope than particular to guarantee the concept of interoperability's application and representation. The next step is to create a list of framework requirements and the specifications, which is accomplished in Sections 3.2 and 3.3. This assists in developing a clear understanding of the framework's essential functions, which will act as a checklist after the project and indicate the project's success or failure. After this is completed, thorough consideration of process flow and architectural decisions are presented. After the completion of all these steps, the framework is ready for implementation. The implementation of the project consists of two parts, deciding on an interoperability technique and implementing a framework using that technique. Based on research in the blockchain domain, a choice is made for a technique that complies with the requirements. Then a minimal-viable product is built to ensure that the solution works and meets the requirements. The development of the prototype is divided into three categories: Database, Blockchain Adapters and an API.

3.2 Requirements

The main idea behind this project is to research and address the problem of Blockchain Interoperability. The primary objective of the framework is to provide a simple and easy-to-use interface that allows communication between disparate Blockchains; this would be achieved by allowing the user to insert, fetch and transfer data between Blockchains.

The detailed requirements of the project are:

- The framework shall be implemented in any programming language as suitable.
- The framework shall enable cross-chain communication between blockchains.
- The framework should be versatile and adaptable.
- The framework should be intuitive and user-friendly.
- The framework should follow a modular architecture.

- The framework should support at least two blockchain networks.
- The framework must not cause a major disruption to the blockchain network.
- The framework must not affect the performance or the security of the blockchain networks.
- The framework should be less resource consuming.
- The framework should be compatible to run on Unix-based Operating Systems such as macOS and Linux.

3.3 Specifications

This section contains a thorough analysis of each requirement and its relative significance to the project's completion. A high significance requirement must be implemented first to finish the project, followed by the other requirement effectively.

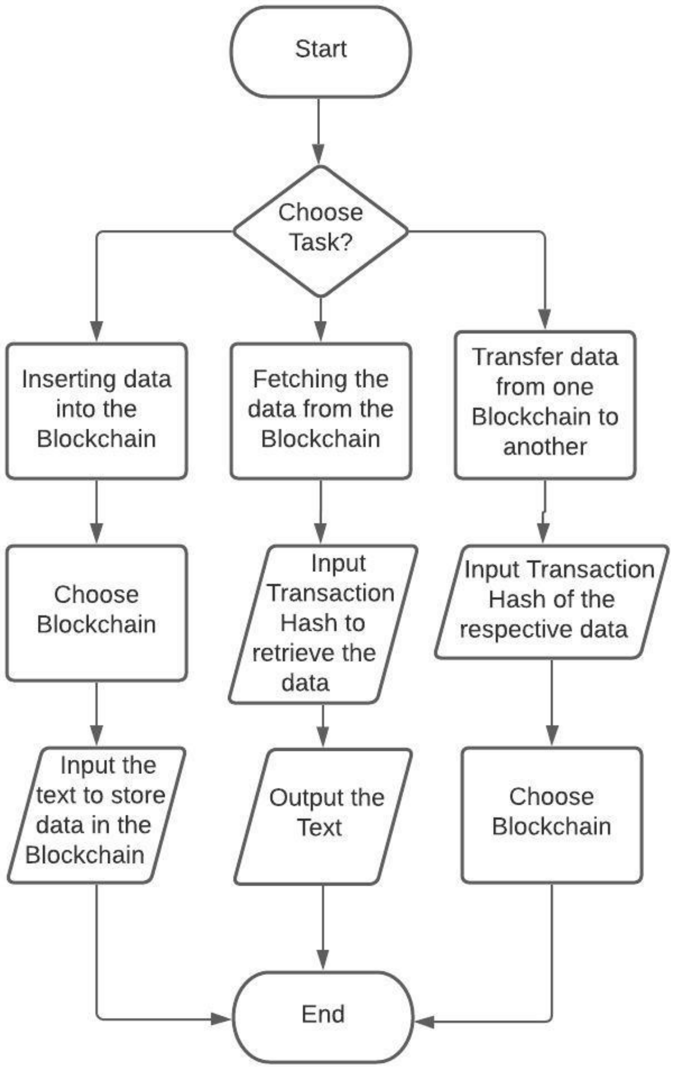
Requirement Details	Specification	Significance
The framework shall be implemented in any programming language as suitable.	The solution shall be implemented in any language as long as it fulfils the requirements. Python/C++ preferred.	High
The framework shall enable cross-chain communication between blockchains.	The solution should enable interaction between different public and private blockchains by allowing transfer of data.	High
The framework should be versatile and adaptable.	The solution should allow the ability to change the functionality of the software without making code changes, as well as is easily adapted and extended to supply new functionality without major redesign.	Med
The framework should be intuitive and user-friendly.	The solution should not require the end-user to have any knowledge on the underlying complex blockchain architecture.	Med
The framework should follow a modular architecture.	The solution should be divided into independent and interchangeable modules to increase readability and maintainability and to provide functional choices to developers.	Low

The framework should support at least two blockchain networks.	The solution architecture must support various types of blockchain networks, such as public or private architectures (Ethereum, Hyperledger Sawtooth, etc.).	High
The framework must not cause a major disruption to the blockchain network.	The solution should not require any modification to the blockchain networks to enable interoperability.	Med
The framework must not affect the performance or the security of the blockchain networks.	The solution security goals should be dependent on the security measures adopted by the underlying blockchain networks. The performance should be dependent on the metrics provided by the blockchain.	High
The framework should be less resource consuming.	The API should perform as efficiently as possible in terms of CPU utilisation and memory consumption.	Low
The framework should be compatible to run on Unix-based Operating Systems such as macOS and Linux.	The API should be able to run on defined operating systems with every prerequisite defined before submission.	High

Table 1: Specifications

3.4 Process

In this section, the process flow of the framework is illustrated. A proper analysis of all of the process flows discovered from the literature shows that the following process flow is most indicative of an Interoperability API.



Start
Choose Task?
Inserting data into the Blockchain
Fetching the data from the Blockchain
Transfer data from one Blockchain to another
Choose Blockchain
Input Transaction Hash to retrieve the data
Input Transaction Hash of the respective data
Input the text to store data in the Blockchain
Output the Text
Choose Blockchain
End

Figure 1: Interoperability API flowchart

3.5 System Architecture

This section provides an overview of the framework architecture, focusing on various aspects: API, Blockchain Nodes, and Network Topology. In order to enable Interoperability, a multi-layered architecture is proposed that integrates multiple Blockchains. The proposed approach makes use of a Notary Scheme to facilitate communication between several blockchains. As such, it is meant to be hosted with the associated Blockchain Application on a trusted server. The Notary Scheme was chosen due to its practicality and simplicity in managing data stored on several Blockchains. The framework comprises three components: an Application Programming Interface (API), Blockchain Connectors, and a Database. Figure 2 illustrates the three components, the data flow between them, and how each component connects to the others. The first layer is the API, which exposes Python methods to insert data, fetch data, and transfer data. The API is in charge of translating the user input into a request. A connector was implemented for each integrated Blockchain, which connects the API to the Blockchain Nodes (Public or Private); it converts the user input into transactions then is sent to the Node. The Nodes are then in charge of transmitting the transaction to the network, where it is processed/confirmed by the validators. Additionally, a database is used to store the transaction hash; once a transaction has been confirmed by the Blockchain. The hash may subsequently be used to fetch the stored data. The data is exclusively kept on the Blockchain and not duplicated in the trusted server (database).

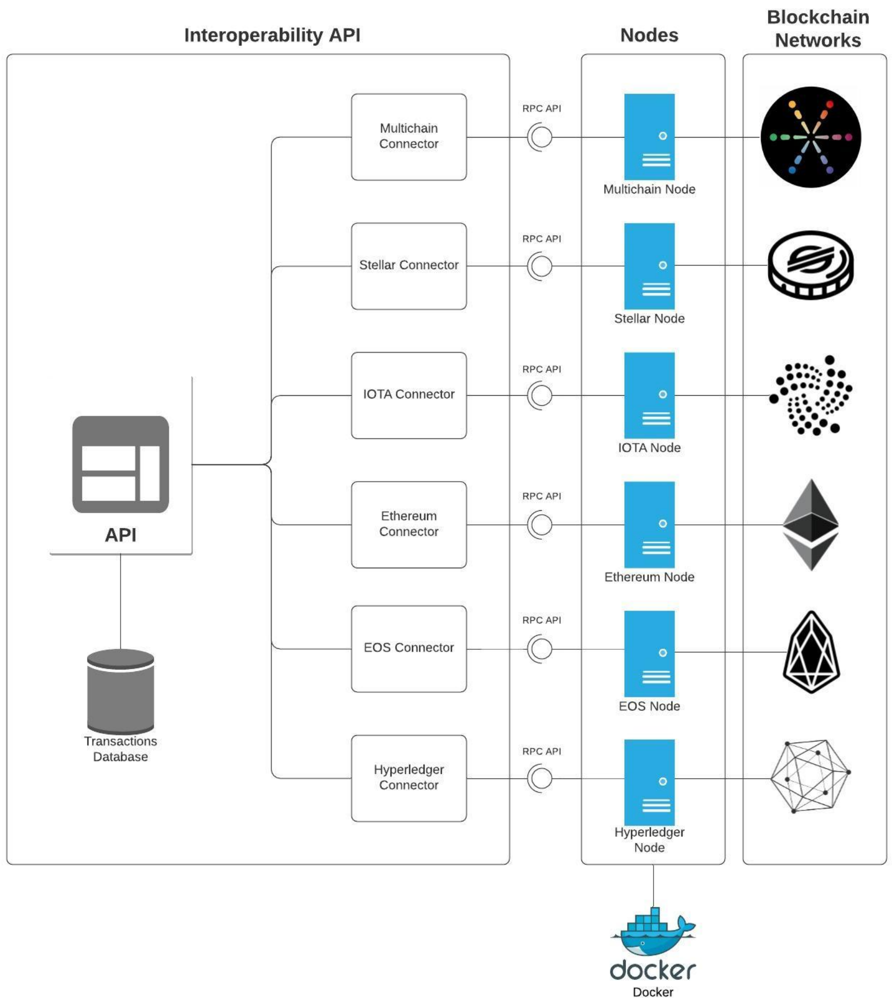


Figure 2: General Architecture of the Framework

3.5 Technology Stack

This section describes the programming language, along with the various technologies and external libraries that will be used in implementing the proposed framework.

3.5.1 Python

Python was chosen as the programming language for this project. Python is a high-level programming language that is interpreted, interactive, and object-oriented. Python is intended to be very readable and straightforward to implement. Guido van Rossum created it, and it was published in 1991. It is open-source, which implies that it is free to use for any purpose, even commercial ones. Python was selected because its characteristics enable fast development while maintaining clean and understandable code. Additionally, there is substantial support for the libraries required in this project, including those for cryptography, and remote procedure calls, which is critical and beneficial for this project owing to the problem domain. [32]

3.5.2 Docker

Docker is an open-source platform that enables the development, deployment, and management of containerised applications. Docker enables the packaging and execution of applications inside a loosely separated environment known as a container. These containers enable developers to mimic applications independent of the technological environment in which they are deployed. Docker is an excellent tool for rapidly booting up and operating a blockchain node without the need to set up each machine manually. Docker significantly simplifies the framework's understanding and usage; this enables the user to start a node simply by typing "docker-compose -f [name].yaml up" in the command-line/terminal. More importantly, this minimises compatibility issues since the nodes operate in a self-contained and replicable environment. Docker is responsible for running the network nodes for Ethereum, Multichain, and Hyperledger in the framework. [33]

3.5.3 SQLite

SQLite is used as a database for storing transaction hashes. SQLite is an in-process library that provides a self-contained, server-less, zero-config SQL database engine. It is a zero-configuration database, which implies that, unlike other databases, it does not need system configuration. This feature enables a high degree of accessibility: duplicating a database is as simple as copying the file containing the data, and sharing a database may be accomplished by sending an email attachment. The database is in charge of maintaining a list of supported blockchains and hashes of confirmed transactions. An SQLite 3 database has been used for the framework. SQLite was selected due to the ease with which the whole database can be stored in a single file, eliminating the requirement for a dedicated server. [34]

3.5.4 External Python Libraries

External Python libraries have also been used for communication with the Database and the Nodes on Blockchain Networks. The libraries used are as follows: *sqlite3* for SQLite, *Web3.py* for Ethereum, *py-stellar-base* for Stellar, *PyOTA* for IOTA, *eosjs_python* for EOS, *sawtooth_sdk* for Hyperledger and *mcrpc* for Multichain. Another external library used in the implementation is *PyInquirer*, a module for interactive command-line user interfaces.

4. Implementation

This section gives a detailed view of the implementation choices based on the requirements in section 3.1 and the Design detailed in sections 3.3 and 3.4. Furthermore, it will describe the development process of the proposed framework, the problems encountered during the development and steps taken to overcome them.

4.1 Overview

The implemented prototype aims to address the problem of Blockchain Interoperability. Blockchain Interoperability allows for sharing of information across multiple Blockchain Networks. The proposed solution is to implement a Python API (based on a Notary Scheme), which will allow the end-user to communicate with multiple disparate Blockchain Networks. The communication will be in the form of three main tasks: inserting data into a Blockchain, fetching data from a Blockchain and transferring data from one Blockchain to another. The iterative and incremental development approach was adopted due to its feasibility; there was a possibility that specific requirements would not be fulfilled on time, necessitating modifications to the framework. Sub-tasks were created based on the requirements and specifications, and every sprint signified a successful completion of those sub-tasks.

4.2 Application Programming Interface (API)

A Python file "*api.py*" is implemented, with three exposed functions: *insert*, *fetch* and *transfer*. The *insert* function is implemented to store data into the Blockchain; it takes in the Data to store, the Blockchain to store in and the Credentials as parameters. The function then connects to the respective Blockchain Node, and the Credentials are passed to connect to the network for interaction. The *insert* function of the connector is then used to send the transaction to the Blockchain, and the Transaction Hash is returned. The *fetch* function is implemented to retrieve the data associated with a specific Transaction Hash; it takes in the Transaction Hash as a parameter. The function then connects to the Transactions database to look up the Blockchain associated with the hash. In a similar fashion as the *insert* function, it connects to the node and calls the *fetch* function of the connector to return the data associated with the transaction hash. The *transfer* function is implemented to transfer data from one Blockchain to another Blockchain; it takes the transaction hash of the data to transfer, the Blockchain to transfer to and the credentials for that Blockchain as parameters. The function calls the *fetch* function to retrieve the data using the hash and then calls the *insert* function to store that data in the desired Blockchain.

4.3 Blockchains

4.3.1 Blockchain

A python file *"blockchain.py"* was implemented, which houses the *Blockchain* class. This class contains the names of all the chosen Blockchains as enumerations for improved readability and accessibility. It also contains the confirmation times for each of the six blockchains. The framework was evaluated on either the public test net of the Blockchain or on local private network. Although test nets' consensus protocol often differs from those of the main net, the confirmation times were adjusted to mimic the main net's behaviour. When sending a transaction to a Blockchain, there is no guarantee that the transaction will be validated and accepted by the network. In some scenarios, the transaction might be rejected by the network due to the transaction fee being minimal, or the confirmation failure. However, even if it is accepted, block finality must be ensured. Bypassing block finality will result in transactions being put on an abandoned fork; however, this is contingent on the blockchain's consensus protocol. This problem is addressed in this framework by including a delay before inserting the transaction into the database. The transaction will be ignored if it is not discovered within this period. For instance, IOTA requires a significant waiting period of 60 seconds to achieve a high chance of finality. Even with finality, transactions are not instantaneous. One of the advantages of proof-of-stake consensus is that it allows for many rounds of communication between nodes, enabling them to hash out issues until two-thirds of them agree on a solution. In order to account for any delays in confirmation, a minimum waiting period of 10 seconds was established. This class also contains variables such as *AWAITING_CONFIRMATION* and *TRANSACTIONS. AWAITING_CONFIRMATION* is used for the finality check of the transaction and is initially set to false. *TRANSACTIONS* contains the Blockchain Name and the Transaction Hash. It is just used for Multichain because it is dependent on a previously confirmed transaction.

4.3.2 Connector Implementation

Six connectors to different blockchains were implemented. Furthermore, a PostgreSQL connector was included as a reference and to enable data storage in a conventional database rather than a blockchain. For the implementation of Blockchain Connectors, an abstract class *Connector* was created in a python file *connector.py;* This allows for ease in extending the implementation to support more Blockchains. The functions *get_txinfo, retrieve_data, convert_string, create_tx, sign_tx, send_raw_tx* and *storeTx* are abstract methods, which have been implemented in base classes for each Blockchain Connector. The storeTx function is identical for all connectors and is used to store the transaction hash in the database with the blockchain ID and timestamp. The function *intialize_credentials* is implemented to initialise the credentials received from the user input. The function *fetch* is implemented to retrieve the data associated with the supplied transaction hash; this is accomplished by retrieving the transaction information for the network, then extracting data from it and finally converting it into a string. The function *insert* is implemented to store data into the Blockchain; this is done by first creating a transaction using the supplied data. The transaction is then signed, and finally, the signed transaction is sent to the node. If the transaction is awaiting confirmation, the confirmation time is verified, and an error is returned if it is not confirmed in time. Otherwise, upon confirmation, the transaction hash is returned. The function *store_time* is implemented for performance testing; it stores the time taken for each transaction to confirm in a csv file. The function *verifying_tx* is implemented to check if the transaction is confirmed after waiting for a defined period; this is done by looking up the blockchain confirmation time, then pausing the execution for that number of seconds; after that, the *fetch* function is called to determine if the transaction is on the network or not. The python files *eos_connector.py*, *eth_connector.py*, *hyperledger_connector.py*, *iota_connector.py*, *mc_connec tor.py*, *stellar_connector.py*, *postgres_connector.py* were implemented for each of the blockchains.

4.3.3 EOS Connector

A python class *EosConnector* is implemented to connect to the EOS Node. This class connects to the EOS Jungle3.0 Testnet, one of the most popular public EOS TestNet. The function *intialize_credentials* gets the Public key, the Private key and the Username to connect to the blockchain. The functions *create_tx*, *sign_tx* and *send_raw_tx,* are used to insert data into the blockchain. The function *create_tx* takes data as a parameter and creates a transaction. The transaction is composed of the sender's and recipient's usernames, the number of tokens being transferred, and the memo (data). The transaction is then signed and transmitted using the *send_raw_tx* function; the eosjs_python library's push transaction function is then used to push it to the network, and the transaction hash is returned. The functions *get_txinfo*, *retrieve_data* and *convert_string,* are used for the retrieval of the data. The function *get_txinfo* returns the transaction information from the node itself, *retrieve_data* is used to slice the information to return the stored data and finally, *convert_string* is used to convert the data into a string.

4.3.4 Ethereum Connector

A python class *EthConnector* is implemented to connect to the Ethereum Node. The Python library "Web3" is used to connect to a local Ethereum node, which is being run by a Docker Image called "Ganache CLI". The function *intialize_credentials* gets the Public key and the Private key to connect to the blockchain. The functions *create_tx*, *get_nonce*, *get_gas_estimate*, *sign_tx* and *send_raw_tx* are used to insert data in to the blockchain. The function *create_tx* takes data as a parameter and creates a transaction; the transaction is made up of the public key of the sender and the recipient, the gas price (cost of carrying out a transaction), value (value transferred for the transaction), data and the nonce (number of transactions made by the sender before this one). The function *get_nonce* is used to get the transaction count associated with a public key. The function *get_gas_estimate* is implemented to estimate the price of gas for a specific transaction. The function *sign_tx* is implemented to sign a transaction with the user's private key and return the raw transaction. The function *send_raw_tx* is implemented to send the raw transaction, and the transaction hash is returned in hex format. The functions *get_txinfo*, *retrieve_data* and *convert_string,* are used for the retrieval of the data. The function *get_txinfo* returns the transaction information from the node itself, *retrieve_data* is used to return the data sent along with the transaction, and finally, *convert_string* is used to convert the data into a string.

4.3.5 Hyperledger Connector

A python class *HyperledgerConnector* is implemented to connect to the Hyperledger Node. The Python library "sawtooth_sdk" is used to connect to a local Hyperledger Sawtooth node, which is being run by a Docker Image of the same name. The function *intialize_credentials* gets the Private key to connect to the blockchain. This private key is used to generate a new private key using the secp256k1 standard, which is called the signer. The signer wraps a private key and provides convenient methods for signing bytes and getting the private key's associated public key. The functions *create_tx*, *sign_tx* and *send_raw_tx*, are used to insert data into the blockchain. The function *create_tx* is implemented to create the transaction; the first step is to encode the payload where the value is randomised to avoid getting the same transaction hash every time, next the transaction header is created, then the header is signed, and finally, the transaction created. The function *send_raw_tx* is used to send the transaction to the network; the first step is to create the batch header and sign the header. Next, the batch is created and encoded in a batch list batches are submitted to the validator. The functions *get_txinfo*, *retrieve_data* and *convert_string*, are implemented for the retrieval of the data. The function *get_txinfo* returns the transaction information from the node itself, *retrieve_data* is used to slice the information to return the stored data and finally, *convert_string* is used to convert the data into a string.

4.3.6 IOTA Connector

A python class *IotaConnector* is implemented to connect to the IOTA Node. The python library PyOTA is used to connect to Load Balancer public node, which the IOTA Foundation hosts. The function *intialize_credentials* gets the Public key to connect to the blockchain. It does not require a Private key since zero-value transfers do not need a sender address. The functions *create_tx*, *sign_tx* and *send_raw_tx*, are used to insert data into the blockchain. The function *create_tx* is used to prepare a transaction object; it consists of the recipient public key, the value (which will be zero in our case), tag (used to search transactions for a specific tag value) and message (the data itself). The function s*end_raw_tx* is used to sign and send the transaction to the network using the *send_transfer* function from PyOTA. The functions *get_txinfo*, r*etrieve_data* and *convert_string*, are implemented for the retrieval of the data. The function *get_txinfo* returns the first bundle in the list of bundles associated with the specified transaction hash, *retrieve_data* is used to return the first element of a JSON-compatible representation of the bundle and *convert_string* is used to convert the data into a string.

4.3.7 Multichain Connector

A python class *MCConnector* is implemented to connect to the Multichain node. The Python library "mcrpc" is used to connect to a local Multichain node, which is being run by a Docker Image called "kunstmaan/docker-multichain" [42]. The function *intialize_credentials* gets the Public key and the Private key to connect to the blockchain. It also requires a Username and Password, which are predefined by the node. The functions *create_tx*, to_hex, *sign_tx* and *send_raw_tx* are implemented to insert data into the blockchain. The function create_tx is implemented to create the transaction; This is accomplished by calling the *createrawtransaction* function from the mcrpc library; it takes the array of transactions, array of public keys, and array of data in hex format as parameters. The function *to_hex* is used to convert a string into hex format using the utf-8 encoding. The function *sign_tx* is implemented to sign the raw transaction; this is accomplished by calling the function *signrawtransaction* from the mcrpc library. It takes the transaction hex string, an array of previous dependent transaction outputs and an array of base58-encoded private keys for signing. The function *send_raw_tx* is implemented to submit the raw transaction (serialized, hex-encoded) to the local node and network; this is accomplished by calling the *sendrawtransaction* function from the mcrpc library, which takes the hex from *the sign_tx* function as a parameter. The functions *get_txinfo*, *retrieve_data* and *convert_string*, are implemented for the retrieval of the data. The function *get_txinfo* returns the raw transaction information from the node, *retrieve_data* is used to slice the information to return the stored data in hex format and finally, *convert_string* is used to convert the data into a string.

4.3.8 Stellar Connector

A python class *StellarConnector* is implemented to connect to the Stellar node. The python library *stellar_base* is implemented to connect to the public test net, a horizon instance. The function *intialize_credentials* gets the Public key and the Private key to connect to the blockchain. The functions *create_tx*, to_hex, *sign_tx* and *send_raw_tx* are implemented to insert data into the blockchain. The function *create_tx* is implemented to build and sign the transaction. The function *send_raw_tx* is implemented to submit the transaction to the node and the network. The functions *get_txinfo*, *retrieve_data* and *convert_string*, are implemented for the retrieval of the data. The function *get_txinfo* returns the transaction information from the testnet, *retrieve_data* is used to slice the information to return the stored data and finally, *convert_string* is used to convert the data into a string.

4.3.9 PostgreSQL Connector

A python class *PostgresConnector* is implemented to connect to the PostgreSQL database. A Docker Image runs the PostgreSQL instance. The function *intialize_credentials* initialises the credentials to connect to the database, which are pre-defined. The functions *create_tx*, to_hex, *sign_tx* and *send_raw_tx* are implemented to insert data into the database. The function *create_tx* is implemented to return an INSERT statement for the supplied data. The function *send_raw_tx* is implemented to execute the INSERT statement; it returns the data index in case of successful insertion; otherwise returns an error. The function *connect* is implemented to establish a connection to the database and create the table for data storage; the credentials are hardcoded. If the connection attempt fails, an error is returned. The functions *get_txinfo* and *convert_string* are implemented for the retrieval of the data. The function *get_txinfo* uses the select statement to return the data associate with a specific index, and finally, *convert_string* is implemented to convert the data into a string.

4.4 Database

The database is in charge of maintaining information about compatible Blockchains and Transactions. A database named “store.db” is created, containing two tables, namely *Blockchains* and *Transactions*. The Blockchains tables has two fields for associating an ID with the Blockchain's name, for instance, ID = 1, Name = 'Multichain'. The ID serves as a connection between the blockchains and the transactions. The Transactions table has two columns stating the Transaction Hash and the timestamp at which the Transaction was confirmed. A python file “database.py” is implemented to connect with the database and perform various tasks.mIt has functions such as *connect_db(func), setup(), clear_db(), create_tables(), populate_bc(), populate_tx(), insert_tx(txid,bc), lookup_latestTX(bc)* and *lookup_bc(hash)*. The function *connect_db(func)* is a decorator function that allows modification of the original function and even replace it without changing the function’s code. The function *setup()* is implemented to initialise the database before running the framework; it calls the function *clear_db()* to drop already existing tables, *create_tables()* to create Blockchains and Transactions table, *populate_bc()* to insert all the blockchain names into the Blockchain table, and *populate_tx()* to insert the pre-defined transaction into the Transactions table. The function *insert_tx(txid, bc)* is implemented to insert every confirmed transaction into the database; INSERT SQL statement is used to insert the transaction hash, blockchain name and timestamp. The function *lookup_latestTX(bc)* is implemented to search the latest transaction in a specified blockchain; this function is mainly used for Multichain since it requires a dependent transaction to allow send and receive rights. The function *lookup_bc(hash)* is implemented to search the blockchain associated with a specified transaction hash; this function allows to connect to the correct blockchain connector to retrieve data.

4.5 User Interface

A python file “cli.py” is implemented to provide a simple interface for the end-user; it has functions such as *get_input(name)*, *get_credentials(selected_blockchain)*, *chooseTask()*, *caseInsert()*, *caseFetch()* and *caseTransfer()*. It also has a dictionary CREDENTIALS_TO_INPUT, which states what credentials are required by each Blockchain. The function *get_input(name)* is implemented to prompt the user for the credential input. The function *get_credentials (selected_blockchain)* is implemented only to prompt the user for the mandatory credentials and then store the credentials in a dictionary to be used by the connector; for instance, the function would ask the user to input the public key if IOTA is chosen. The function *chooseTask()* is implemented to allow the user to choose the task (Insert Data, Fetch Data and Transfer Data) to perform. The function *caseInsert()* is implemented to allow the user to store data (in the form of text) in the Blockchain; the user is provided with a list of blockchains to choose from, then prompted to input the data user would like to store and finally, prompted to input the required credentials. The function *caseFetch()* is implemented to allow the user to fetch data from the Blockchain; the user is prompted to input the transaction hash of the associated data, which returns the respective string. The function *caseTransfer()* is implemented to allow the user to transfer data from one Blockchain to another; the user is prompted to input the transaction hash of the data to transfer, then which Blockchain to transfer to and finally, the credentials for the recipient blockchain.

4.6 Implementation Issues

Complications and issues are unavoidable throughout the execution of the framework. These issues may occur as a result of inaccuracies or ambiguity in the requirements and specifications. To address these issues, a new strategy for development must be adopted. The primary objective was to build and deliver the framework precisely as designed.

4.6.1 Obsolete Nodes

Since some of the supported blockchains use public nodes, there were cases during the implementation where the nodes went obsolete; this means that the node URL would stop working due to the node being shut down. EOS was the primary blockchain facing this issue; the Jungle 2.0 test net was being used initially, but when EOS updated it to Jungle 3.0 test net, the node URL in the implementation had to be changed, and the need to create a new account on the blockchain.

4.6.3 Retrieving data from EOS

The process of extracting data from the transaction information was a bit complicated. After the transaction information was fetched (which was in the form of JSON), it had to be parsed before extracting the data. The information consisted of numerous dictionaries and lists; from there, it had to be indexed before returning the data. This took longer because the EOS architecture changed over time, due to which changes had to be made to index the correct data properly.

4.6.4 Credentials Privacy

In the initial phases of implementation, the credentials were being stored in the database, but this was not ideal because of the sensitivity of credentials. The end-user would have to trust the Notary (the person hosting the server/database). Also, in case of any cyberattack, the credentials could be stolen, causing problems for the user. To avoid this, a work-around was implemented by prompting the user to input their credentials when running the framework directly; this would not store the credentials in the framework and thus increase the privacy/security of the proposed solution.

4.6.5 Multichain Transaction

Another issue encountered was related to offline signing. Offline signing entails managing keys (particularly private keys) remotely, i.e. outside of a wallet. Offline signing offers higher security by storing the private keys on hardware devices, often called "cold storage." Certain functions, however, such as listunspent, which returns a list of all unspent transaction outputs associated with an address, are accessible only when a wallet is utilised. Offline signing needs a raw transaction and a private key to sign and send to the network. MultiChain raw transactions need at least one unspent transaction output as input. Without importing the private key into a wallet, an unspent transaction output can be retrieved by querying the database for the most recent transaction. The system, however, is not aware of any transaction when the database is first initialised. As a workaround, a seed transaction is preconfigured and saved in the database before starting the Multichain node. A new public key, private key and transaction hash has to be generated upon starting the node. The latest Multichain transaction hash has to be then replaced by the generated transaction hash in the database; this can be done by using a database browser for SQLite.

4.6.6 EOS requiring PowerUp

With EOS, sometimes, when trying to insert data, a "validate CPU usage" error is returned; this means staking to CPU is not working anymore. To overcome this error, account PowerUp is required. PowerUp is used to rent resources for 24 hour periods and allows an account to transact a limited number of times in that period. PowerUp can be performed at the Node's TestNet Monitor by visiting: https://monitor3.jungletestnet.io/#home. This was done every single time the error came up.

4.7 Testing

4.7.1 Unit Testing

The first software testing technique used was unit testing, which needed to be carried out while the framework was still being built. The unit testing method deconstructs software into separate components and then evaluates each component under specific inputs. Unit testing was critical in the development process, helping to uncover problems earlier. Each function was verified during unit testing to ensure it performed as intended. Additionally, ensure that each function is coordinated with the software's other characteristics (some components depend on each other to work). After completing a specific milestone, unit testing was done before proceeding to the next; this increased the agility of the development process.

4.7.2 Non-Functional Testing

Non-functional testing is used to evaluate the system's overall quality. It is entirely focused on non-functional characteristics such as performance and usability. The major non-functional characteristics and whether or not the framework has them are explained below:

1. Usability: The framework is straightforward to use and understand. Additionally, it is extensively documented, with comments outlining each function. Although the user is not needed to access the code, if they do, the functions and variables are appropriately labelled to prevent confusion. For example, the function *send_raw_tx* indicates that this method sends raw transactions to the network.

2. Modularity/Scalability: The framework enables modularity and scalability; it may be extended to support other blockchains and features. Modifying the code or adding new functions is straightforward. The framework functions may be combined with any newly developed features. Additionally, the framework may be integrated with components outside of its scope, such as Third-Party Content. The code is designed so that each function performs a particular job; this ensures that when new features are added, they do not add to the code's complexity or make it less understandable.

3. Supportability: Because the framework is pure code and not an application, it may be used and supported on any Unix-like operating system, including macOS, Linux, etc. For example, no changes to the code are required to move the framework from Linux to macOS or vice versa.

4. Reliability: The term "reliability" refers to a system's capacity to operate without failure for a certain length of time in a specified environment. The framework is complete in that it satisfies the criteria and accomplishes the tasks detailed in the Implementation chapter. While the API is in use, the computer is not hung, or the execution is not terminated.

5. Performance: The framework execution does not use a large amount of memory or system resources. It makes optimal use of the CPU's available resources, storage space, and system RAM. The system RAM allocation is generally constant and does not exceed 200MB. Please refer to section 5.1.2 for detailed analysis.

4.7.3 Requirement-Based Testing

In this Testing approach, the framework is tested using the test cases obtained from requirements stated in Chapter 3. The screenshots of the results have been attached in Appendix A, with the specific requirement they test and justify. Some of the requirements do not have any screenshot proofs since they are code-based.

R1 - The framework shall be implemented in any programming language as suitable: the framework was implemented in Python, and Visual Studio Code was used as a code editor. Python was selected for its simplicity and readability of code.

R2 - The framework shall enable cross-chain communication between blockchains: the proposed framework allows users to store data (in the form of text) on a Blockchain, which can subsequently be transferred from that blockchain to another. This was accomplished via a Notary Scheme, which enables a designated entity to certify the events of one chain on another. This method requires confidence in the Notary. A screenshot showing transfer of data from EOS to IOTA has been attached in Appendix A.1.

R3 - The framework should be versatile and adaptable: this is accomplished by enabling users to record a text message in the blockchain that may represent any kind of data. The proof for this is attached in Appendix A.2.

R4 - The framework should be intuitive and user-friendly: This is achieved by implementing a clean and straightforward command-line interface. The end-user can use the keyboard to interact with the framework. The user prompts and exposed functions are easy to read and understand. The use of Docker further enables user-friendliness since it allows the user to run a local node by using a simple command from the terminal. The screenshot of the command-line interface has been attached in appendix A.3.

R5 - The framework should follow a modular architecture: this is accomplished by implementing each Blockchain Connector from the abstract class, thus establishing a standard interface. As a result, the end-user may add or delete blockchain connectors as needed. Additionally, public test nets were utilised where feasible, making it simpler to verify confirmed transactions through a blockchain explorer website like https://testnet.steexp.com for the Stellar blockchain. A local private test net was used in the absence of a stable public test net or if using a public test net harmed usability, for example, if the procedure of obtaining test tokens was onerous. Appendix A.4 contains screenshots of the confirmed transaction on the Stellar explorer.

R6 - The framework should support at least two blockchain networks: the proposed framework supports six blockchains, namely Ethereum, Stellar, EOS, IOTA, Hyperledger and Multichain.

R7 - The solution must not cause significant disruption to the blockchain network: the proposed framework requires no modification to the underlying blockchain architecture (no repeated forking); thus, any disruption to the network is not possible.

R8 - The solution must not affect the performance or the security of the blockchain networks: as stated in requirement R7, there are no modifications made to the blockchain architecture; thus, the performance or security of the blockchain will not be affected. This is because the framework only connects to the nodes, which in turn connect to the network itself.

R9 - The framework must be less resource consuming: it must run smoothly and without crashing the machine. As shown in Section 5.1.2, IOTA consumes the highest memory, which is minimal at 0.9 Mb.

R10 - The framework should be compatible with Unix-based operating systems such as macOS and Linux: it has been tested on both Linux and macOS and works as expected.

4.8 Limitations

Since this is a preliminary version of the proposed framework, it does have certain limitations. The suggested framework is neither completely trust-less nor decentralised, which may be seen as a restriction since it contradicts the fundamental concept of blockchain technology, namely decentralisation. The framework is built on the Notary Scheme, which allows an external (notary) entity to make claims on one blockchain about another. Because the Notary has complete control over the cross-chain capability, the end-user must trust the Notary to make accurate assertions regarding another external blockchain.

5. Evaluation

This chapter evaluates the designed and developed framework using an evaluation setup and defined assessment scenarios. Evaluation is the process of determining if an outcome accomplishes the intended goal and fulfils the specified requirements.

5.1 Results and Analysis

In the following experiments, the framework is evaluated by examining the metrics of performance, assessment of security and transaction data size.

5.1.1 Performance Assessment

One of the most important criteria for a blockchain system regarding overall performance is how fast a transaction is verified and added. A performance analysis was performed to evaluate the framework's efficiency and stability. The tests were conducted on a MacBook Pro (15-inch, 2016) equipped with a 2.7 GHz quad-core Intel Core i7 processor, 16 GB of RAM, and macOS Catalina 10.15.1 running Python 3.7.1. In order to test the performance of the implemented framework, an analysis of the Transaction Latency of each blockchain was conducted, which included measurements for the time between transaction send and receipt of the transaction hash (this is when the transaction is recorded in the ledger). Each transaction was transmitted synchronously, meaning that it was confirmed one after the other. On each blockchain, 100 measurements were collected: this is evident from Figure 3, which illustrates the significant performance gap across blockchains; this is due to blockchains that use local nodes having much superior performance

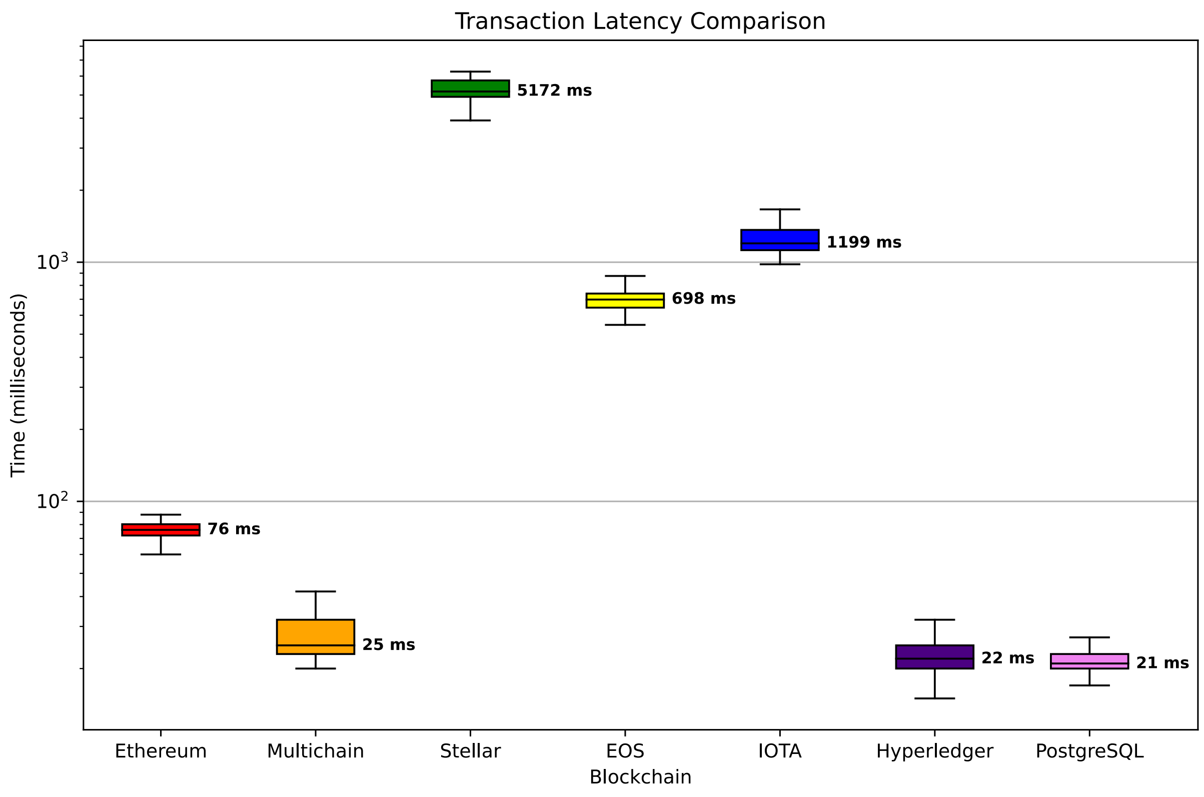


Figure 3: Transaction Latency Comparison

than blockchains that use public nodes. EOS, IOTA, and Stellar all utilise public nodes and take an average of 0.7 seconds, 1.2 seconds, and 5.1 seconds to complete, respectively. On the other hand, Ethereum, Multichain, and Hyperledger all utilise local nodes and take 0.076s, 0.025s, and 0.022s on average, respectively. The results of these blockchains' measurements were compared to a conventional database (PostgreSQL) to see how they stack up against it. Private and permission-based blockchains - Hyperledger and Multichain - achieve comparable speed to a database like PostgreSQL since both blockchains prioritise data streams. Both, however, need additional procedures (for example, the generation of raw transactions and cryptographic signatures) that PostgreSQL does not. Due to the high transaction costs associated with each blockchain's primary network, testing was limited to test networks. Still, the process is identical on the test networks, and the tests are carried out to verify that the prototype is operational.

5.1.2 Resource Consumption

The memory usage by the framework was also tested. The memory consumption when trying to store data into each blockchain was observed. Figure 4 presents the results. We compared the processing resource consumption by comparing the peak memory usage. IOTA requires the most memory in comparison to other blockchains, while PostgreSQL requires the least memory. Hyperledger and Multichain are very similar due to both being data stream-oriented. Ethereum requires more memory than other local nodes; this could be because of computations to estimate gas and get nonce. To sum it up, it can be observed that the framework consumes minimal memory.

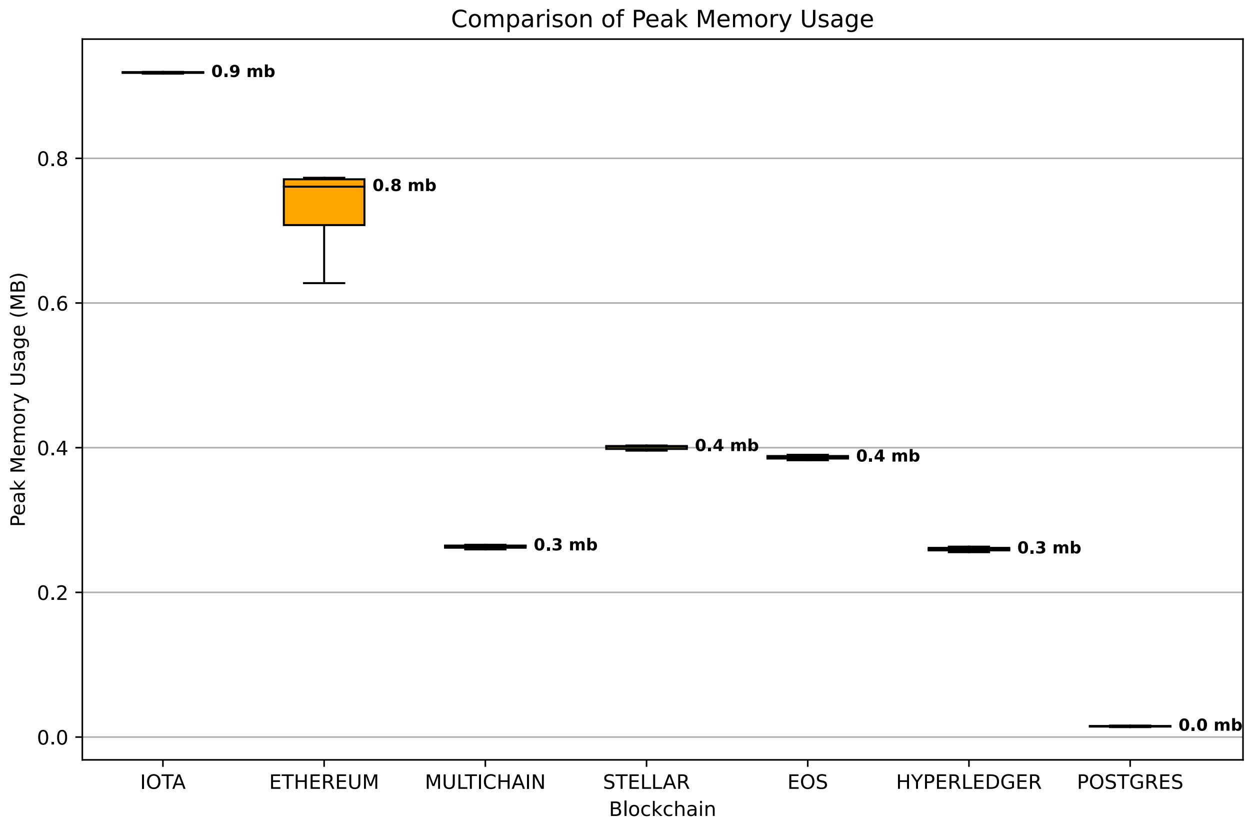


Figure 4: Memory Usage Comparison

5.1.3 Security Assessment

The essential security issues are covered in this section. Given that the proposed solution is generic and scalable to various blockchain networks, the security objectives are contingent on the underlying blockchain systems' security mechanisms. For example, an improperly designed and insecure blockchain network supported by the framework will remain vulnerable. Thus, part of the suggested solution involves investigating the security components to ascertain if the framework might be exposed to any security threats because of supporting Blockchain Interoperability. By doing so, it is presumed that the supported blockchains are secure. The framework is compared to the industry-standard CIA triad model of information security to determine how well it guarantees confidentiality, integrity, and availability. No participant blockchain entities would have to reveal private or sensitive information with the proposed approach. Since the fundamental design of the Blockchain is not being modified, there are no data privacy concerns. The framework fulfils the confidentiality property because response data is encrypted between the source and destination by the source's peers using the remote client's public key, guaranteeing that an untrusted parted cannot read or exfiltrate the data. The proposed framework eliminates the need for different third parties to interpret or transmit information and transactions across interoperating blockchain networks, thus ensuring their integrity. For instance, if a user transfers data from Ethereum to Stellar, the framework is solely responsible for translating the transaction and interacting with validator nodes to relay information. Additionally, the blockchain network regulates communication between the organisations involved, registering those that can create tokens and guaranteeing the security and authenticity of both sides' communication links. The framework is not completely safe against distributed denial of service (DDoS) attacks because the system relies on the notary scheme. The blockchain node's RPC or HTTP port being exposed is one risk factor. A transaction may be executed in case the node also stores credentials in a local wallet. The architecture avoids the node holding the private keys by relying on blockchain connectors that only locally sign transactions; this still does not exclude the possibility that malicious, unrelated transactions and DDoS attacks on the node may be sent over the ports. When public nodes are utilised, the node's owner can prevent a transaction from being executed. In conclusion, such security concerns may be addressed by using private local nodes and restricting the RPC/HTTP port to the local network or a locally hosted application. Additionally, DoS protection may be incorporated into the framework, shielding the user from such attacks. Regardless, Interoperability necessitates the establishment of trustworthy links between the participating blockchain networks. Another significant issue is to prevent attackers from getting access to the private keys; this will result in two consequences: The first possibility is that the tokens in the connected account may be spent. Because this prototype is built to store data on the Blockchain, just the transaction fees will need to be covered. Keeping just the money required for the transactions eliminates this danger. The second possibility is that an attacker may create distorted transactions by placing non-standard data, which is very difficult to tell apart from the legitimate transaction. This may be a factor to consider depending on the application; in most instances, the motivation for such acts is dubious. As described in Section 4.6.4, the end-user enters the private key directly. It is not saved anywhere else in the framework; this eliminates the possibility of an attacker obtaining access to the credentials. The major benefits of Blockchains are decentralisation and cryptographic characteristics that eliminate trust. As such, it would be beneficial if this characteristic was also applied to this solution. However, utilising a notary scheme means that a user must place confidence in the notary, which typically is not an issue since notaries are generally reputable and trustworthy. Nevertheless, because the notary can control the framework, they may alter transactions and data, such as altering the contents or even filtering certain transactions.

Blockchain applications, on the whole, are never free of trust and always come with many layers which need a certain level of confidence. This design uses the Blockchain's basic code and cryptographic properties as the first layer. The second layer may be comprised of an RPC server and an RPC client. Notaries may be referred to as the third layer of trust. At this point, many additional layers that require trust follow, for example, the operating system used by the end-user. In this light, the notary system is just one of several layers of trust. While transparency and open-source initiatives may help lower the level of trust required, such efforts can be complemented by peer reviews and using a decentralised application environment.

5.1.4 Transaction Data Size

Blockchains built for transactional purposes are not intended to hold arbitrary data in transactions, due to which the maximum quantity of data that may be included is limited. Table 2 details the total amount of transaction data that can be stored in each of the different blockchains that the framework supports. Ethereum's transaction data size depends on the gas limit. According to [35], the size limit for Ethereum could be calculated by the formula: (gas limit - fees per transaction) / fees for every non-zero byte of data. As of July 2021, the gas limit is 15000000, which means the size limit is 220279 bytes. Stellar has a data size limit of 28 characters [36]. Trying to store a 29 character string in Stellar returns an error; a screenshot showing the error is attached in the appendix A.5. EOS's transfer memo is limited to 256 characters and is a type string [37]. IOTA's message has a limit of 2187 trytes, which converts to 1300 bytes [38]. Multichain's metadata has a size limit of 2097152 bytes at default but is upgradable to 64 MB, subject to specific chain configuration [39]. Hyperledger's data size is limited to 20 characters and is of string type [40]. It is worth noting that in some blockchains, the data size limitation may be bypassed via smart contracts. Typically, storing significant quantities of data on blockchains is not very cost-effective. Alternatively, data may be stored in a (decentralised) database, e.g. BigChainDB. This way, storing just the URL to the data together with its hash in the Blockchain would suffice. Thus, the data's authenticity may be verified at any moment while being more efficiently stored.

Blockchain	Size Limit (bytes)
Ethereum	220279
Stellar	28
EOS	256
IOTA	1300
Multichain	2097152
Hyperledger	20

Table 2: Transaction Data size

5.2 Comparison with Existing Solutions

This section explores the differences between the proposed and existing solutions (presented in section 2.2). Table 3 illustrates the proposed framework and the existing solutions side-by-side. The comparison criteria are based on the standards and goals that would help connect different blockchains. Except for the solutions that use hash-locking, all current systems use a central authority to oversee transactions and guarantee that assets and data are correctly transferred. The Notary Scheme and Interledger Protocol solutions both include a blockchain network that blindly trusts another authority. Herdius is notary-scheme-based, but integrating the threshold mechanism (discussed in section 2.2) partially decentralises the system. By conducting atomic swaps across multiple blockchains, hash-locking systems may enable trust-less asset exchange at the chain level. Cosmos and Polkadot are a hybrid of Notary Scheme and Relay, which provide trustlessness as well. All systems ensure the integrity of communications sent between blockchains except Sidechain/Relay systems, like Cosmos and Polkadot; they require sophisticated methods for verifying the integrity of transmitted data, making them inefficient. Sidechain/Relay systems are also not versatile since they require changes in the underlying implementation of the supported Blockchains. Notably, all solutions have scalability issues, except for Cosmos, Polkadot, and Herdius, which use parallel blockchains to address this problem. Private and public networks with differing frameworks, such as Ethereum and Hyperledger Fabric, may benefit from cross-chain communication capabilities provided by framework-independent solutions. Ethereum is a cryptocurrency-based blockchain, and sidechain solutions enable cryptocurrency transfer between the primary and child ledgers. In contrast, the Hyperledger project lacks both a native cryptocurrency and the notion of child ledgers. Since they rely on a particular block header format, sidechain/relay systems theoretically are not framework-independent. Many solutions are domain-specific, making them more limited in their capabilities as they only allow token-based assets. Underlying complicated implementation and the inability to abstract it from the end-user contribute to the current solutions' lack of ease of use.

Solution	Decentralised	Trustless	Integrity	Efficient	Versatile	Scalable	Framework-independent	Domain-neutral	Ease of Use
Notary Scheme	No	No	Yes	Yes	Yes	No	Yes	No	No
Sidechain /Relay	No	No	Yes	No	No	No	No	Yes	No
Hash-Locking	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No
Herdius	Partially	No	Yes	Yes	Yes	Yes	Yes	No	No
Interledger	No	No	Yes	Yes	No	No	Yes	No	No
Cosmos/ Polkadot	No	Yes	Yes	No	No	Yes	No	Yes	No
Proposed Solution	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 3: Comparison of proposed framework with existing solutions

6. Legal, Social, Ethical and Professional Issues

The project conformed strictly to the British Computer Society's (BCS) Code of Conduct and Code of Good Practice [41]. Due to the small scale of this academic project, only a few of the Code of Conduct's principles apply. It was critical to guarantee that the Code of Conduct's standards were followed, or else it might have resulted in various legal and ethical repercussions. The most crucial regulation was to guarantee that no one's work was unlawfully obtained or utilised without consent. This report makes explicit references to the use of all third-party material (libraries or code). Using external libraries encourages code reuse, which expedites the project's implementation process; this did not affect the project's quality. This project is entirely the author's work; any open-source material utilised has been cited.

7. Conclusion & Future Work

In this Chapter, the report is concluded with a brief summary, and directions for future work are presented.

7.1 Conclusion

There has been a tremendous increase in the popularity of cryptocurrencies in the past few years, owing to their disruptive impact on the industry and, perhaps more significantly, their astronomical rise in price. The future will see many coexisting blockchains with differing functionality, privacy, and efficiency, each serving a portion of potential use cases. Additionally, various businesses (both public and private blockchains) will manage several kinds of blockchains that will need interoperability in a network. Blockchain technology will alter how people engage with governments, businesses, and organisations. Numerous businesses will use blockchains in order to reap the benefits of decentralised, immutable ledgers. As a result, numerous blockchains that can communicate with each other will realise the full potential of blockchain networks. In the future, people will probably be able to trade crypto assets across blockchain systems in the same way that people now transmit e-mail or messages between systems. Thus, interoperability is required to connect different blockchains via the exchange of data and assets. While there are various existing projects on cross-chain communication, the majority of them rely on sidechains; this method, however, is contingent on modifications to the blockchain's underlying architecture. Thus, the majority of the projects cannot offer interoperability across many blockchains in existence today. Another factor is that progress in terms of simplicity and clarity also has to be achieved. Access to clear and understandable interfaces is critical for developers and consumers to engage with multiple blockchains. The purpose of this project was to propose an application-based Blockchain Interoperability solution that can be easily adapted to a wide variety of use cases at a low implementation and deployment cost. This framework enables end-users to insert, fetch, and transfer data between blockchains via a notary scheme. The proposed solution was evaluated to ensure that it met the use case requirements for functionality and performance. Additionally, a security analysis was conducted, and the proposed interoperability framework was shown to present no new security risks. The proposed solution was briefly compared to other existing solutions; the comparison demonstrates that the proposed framework offers more flexibility, adaptability, and intuitiveness than current alternatives. Centralisation, on the other hand, is a concern. The proposed framework assumes that the notary is reliable and will not tamper with transaction data. Since centralisation increases the notary's exposure, they could be vulnerable to cyber-attacks. In this sense, the use of the Notary Scheme contradicts the concept of decentralised and trustless Blockchains. Regardless, it should be noted that all blockchain applications to date have required some degree of trust to be effective. However, since blockchain interoperability remains primarily unexplored, different innovative methods will likely emerge to provide blockchain technology to a broader audience. The proposed framework is a positive first step, but more work must be done to enhance cross-chain communication across blockchains.

7.2 Future Work

There are many areas where this framework may be enhanced beyond adding support for more blockchains. This section contains recommendations for extending the framework and also mentions some lingering problems which remain open for further consideration:

- In the future, zero-knowledge proof and multi-signature transactions may be coupled to improve privacy and security further.
- Usability could be improved by optimising error handling. For instance, suppose a transaction failed, this may be automatically attempted again later, and the user should be informed about it.
- Support for smart contracts may be added, bringing programmability to blockchains and thus increasing the number of potential use cases.
- Efforts can be made to operate this framework in a decentralised and trust-free way; this could be accomplished by integrating methods such as Sidechain or Hash-locking into the existing architecture.
- Further improvements like troubleshooting and bug removal, portability of the user interface, and database management can also be made.
- The proposed solution's performance may also be compared to those of the existing solutions mentioned in Section 2.2, which has not been possible in this project as they still lack connectors to the supported blockchains.

8. References

[1] Nakamoto, S. (2008) *Bitcoin: A Peer-to-Peer Electronic Cash System* [online]. Available from: https://bitcoin.org/bitcoin.pdf (Accessed 26 May 2021).

[2] Buterin, V. (2013) *Ethereum Whitepaper | ethereum.org* [online]. Available from: https://ethereum.org/en/whitepaper/ (Accessed 26 May 2021).

[3] Katz, L. (2021) *Bitcoin's Market Cap Surges Past $100 Billion* [online]. Available from: https://www.bloomberg.com/news/articles/2017-10-30/bitcoin-passes-more-milestones-as-market-cap-tops-100-billion (Accessed 26 May 2021).

[4] Conway, L. (2021) *The 10 Most Important Cryptocurrencies Other Than Bitcoin* [online]. Available from: https://www.investopedia.com/tech/most-important-cryptocurrencies-other-than-bitcoin/ (Accessed 27 May 2021).

[5] Schulte, S. et al. (2019) *Towards Blockchain Interoperability* [online]. Available from: https://www.dsg.tuwien.ac.at/team/sschulte/paper/2019_SSFB19_BPM_BF.pdf (Accessed 26 March 2021).

[6] EOS.IO (2018) *EOS.IO Technical White Paper v2* [online]. Available from: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md (Accessed 27 May 2021).

[7] van Saberhagen, N. (2013) *CryptoNote v 2.0* [online]. Available from: https://www.getmonero.org/resources/research-lab/pubs/whitepaper_annotated.pdf (Accessed 27 May 2021).

[8] NEO (2014) *Neo Documentation* [online]. Available from: https://docs.neo.org/docs/en-us/index.html (Accessed 27 May 2021).

[9] Greenspan, G. (2015) *MultiChain Private Blockchain — White Paper* [online]. Available from: https://www.multichain.com/download/MultiChain-White-Paper.pdf (Accessed 27 May 2021).

[10] Wang, H. et al. (2018) *Blockchain challenges and opportunities: a survey* [online]. Available from: https://www.researchgate.net/publication/328271018_Blockchain_challenges_and_opportunities_a_survey (Accessed 27 May 2021).

[11] BELCHIOR, R. et al. (2021) *A Survey on Blockchain Interoperability: Past, Present, and Future Trends* [online]. Available from: https://arxiv.org/pdf/2005.14282.pdf (Accessed 26 March 2021).

[12] Deloitte (2016) *What is a Blockchain?* [online]. Available from: https://www2.deloitte.com/content/dam/Deloitte/uk/Documents/Innovation/deloitte-uk-what-is-blockchain-2016.pdf (Accessed 30 May 2021).

[13] Neocapita (2017) *The Logical Components of Blockchain* [online]. Available from: https://medium.com/@neocapita/the-logical-components-of-blockchain-870d781a4a3a (Accessed 30 May 2021).

[14] Witherspoon, Z. (2017) *A Hitchhiker's Guide to Consensus Algorithms | Hacker Noon* [online]. Available from: https://hackernoon.com/a-hitchhikers-guide-to-consensus-algorithms-d81aae3eb0e3 (Accessed 2 June 2021).

[15] Buterin, V. (2016) *Chain Interoperability* [online]. Available from: https://www.r3.com/wp-content/uploads/2018/04/Chain_Interoperability_R3.pdf (Accessed 2 June 2021).

[16] World Bank (2020) *Blockchain Interoperability* [online]. Available from: http://documents1.worldbank.org/curated/en/373781615365676101/pdf/Blockchain-Interoperability.pdf (Accessed 26 March 2021).

[17] Herdius (2017) *Herdius Whitepaper* [online]. Available from: https://herdius.com/whitepaper/Herdius%20Lightpaper.pdf (Accessed 5 June 2021).

[18] AION (2018) *Aion AION whitepapers - whitepaper.io* [online]. Available from: https://whitepaper.io/coin/aion (Accessed 4 June 2021).

[19] Kwon, J. & Buchman, E. (2018) *Cosmos Network - Internet of Blockchains* [online]. Available from: https://v1.cosmos.network/resources/whitepaper (Accessed 5 June 2021).

[20] WOOD, G. (2017) *POLKADOT: VISION FOR A HETEROGENEOUS MULTI-CHAIN FRAMEWORK* [online]. Available from: https://polkadot.network/PolkaDotPaper.pdf (Accessed 5 June 2021).

[21] ARK (2018) *ARK Ecosystem Whitepaper* [online]. Available from: https://ark.io/Whitepaper.pdf (Accessed 5 June 2021).

[22] Wanchain (2018) *Wanchain: Building Super Financial Markets for the New Digital Economy* [online]. Available from: https://wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf (Accessed 6 June 2021).

[23] Thomas, S. & Schwartz, E. (2016) *A Protocol for Interledger Payments* [online]. Available from: https://interledger.org/interledger.pdf (Accessed 6 June 2021).

[24] Ethereum (2018) *proof-of-stake-faqs* [online]. Available from: https://eth.wiki/en/concepts/proof-of-stake-faqs (Accessed 9 July 2021).

[25] Frankenfield, J. (2021) *Binance Coin (BNB)* [online]. Available from: https://www.investopedia.com/terms/b/binance-coin-bnb.asp (Accessed 9 July 2021).

[26] Takyar, A. (n.d.) *What is Stellar Blockchain? A Complete Guide for Beginners* [online]. Available from: https://www.leewayhertz.com/what-is-stellar-blockchain/ (Accessed 9 July 2021).

[27] Bitcurate (2018) *EOS: Potential Ethereum Killer and Only Just Few Months Old?* [online]. Available from: https://bitcurate.medium.com/eos-potential-ethereum-killer-and-only-just-few-months-old-c9e5482ae5aa (Accessed 9 July 2021).

[28] Coindesk (n.d.) *EOS - CoinDesk* [online]. Available from: https://www.coindesk.com/crypto/

eos (Accessed 9 July 2021).
[29] Popov, S. (2016) *The tangle* [online]. Available from: https://iotatoken.com/IOTA_Whitepaper.pdf (Accessed 9 July 2021).

[30] FRANKENFIELD, J. (2021) *Hyperledger Sawtooth Definition* [online]. Available from: https://www.investopedia.com/terms/h/hyperledger-sawtooth.asp (Accessed 9 July 2021).

[31] Rizzo, P. (2015) *Multichain: A Build-Your-Own Blockchain Service for Banks* [online]. Available from: https://www.coindesk.com/multichain-build-ownblockchain-banks (Accessed 9 July 2021).

[32] Python (n.d.) *Welcome to Python.org* [online]. Available from: https://www.python.org/ (Accessed 12 July 2021).

[33] Docker (n.d.) *Empowering App Development for Developers | Docker* [online]. Available from: https://www.docker.com/ (Accessed 12 July 2021).

[34] SQLite (n.d.) *SQLite Home Page* [online]. Available from: https://sqlite.org/index.html (Accessed 12 July 2021).

[35] Afr (2016) *Is there a limit for transaction size?* [online]. Available from: https://ethereum.stackexchange.com/questions/1106/is-there-a-limit-for-transaction-size (Accessed 23 July 2021).

[36] Stellar (2021) *Transactions – Stellar Documentation* [online]. Available from: https://developers.stellar.org/docs/glossary/transactions/ (Accessed 23 July 2021).

[37] EOSIO (2018) *eos/eosio.token.cpp* [online]. Available from: https://github.com/EOSIO/eos/blob/72bb132973bd7c522b42a6e49cb2fda9740556dd/contracts/eosio.token/eosio.token.cpp#L36 (Accessed 23 July 2021).

[38] Schiener, D. (n.d.) *The Anatomy of a Transaction · IOTA Guide* [online]. Available from: https://domschiener.gitbooks.io/iota-guide/content/chapter1/transactions-and-bundles.html (Accessed 23 July 2021).

[39] Multichain (n.d.) *Customizing blockchain parameters | MultiChain* [online]. Available from: https://www.multichain.com/developers/blockchain-parameters/ (Accessed 23 July 2021).

[40] Interledger (n.d.) *IntegerKey Transaction Family — Sawtooth v1.0.5 documentation* [online]. Available from: https://sawtooth.hyperledger.org/docs/core/releases/1.0/transaction_family_specifications/integerkey_transaction_family.html#state (Accessed 23 July 2021).

[41] BCS [online]. Available from: https://www.bcs.org/upload/pdf/conduct.pdf (Accessed 27 July 2021).

[42] Kunstmaan (2015) *GitHub - Kunstmaan/docker-multichain: Multichain docker images* [online]. Available from: https://github.com/Kunstmaan/docker-multichain (Accessed 30 July 2021).

[43] Hejazi-Sepehr, S. et al. (2019) *Transwarp Conduit: Interoperable Blockchain Application Framework* [online]. Available from: https://arxiv.org/abs/1906.03256 (Accessed 13 July 2021).

9. Appendices

Appendix A: Extra Information (NOTE: ANONYMISED SO MISSING CODE EXAMPLES)

A.1 Data Transfer from EOS to IOTA

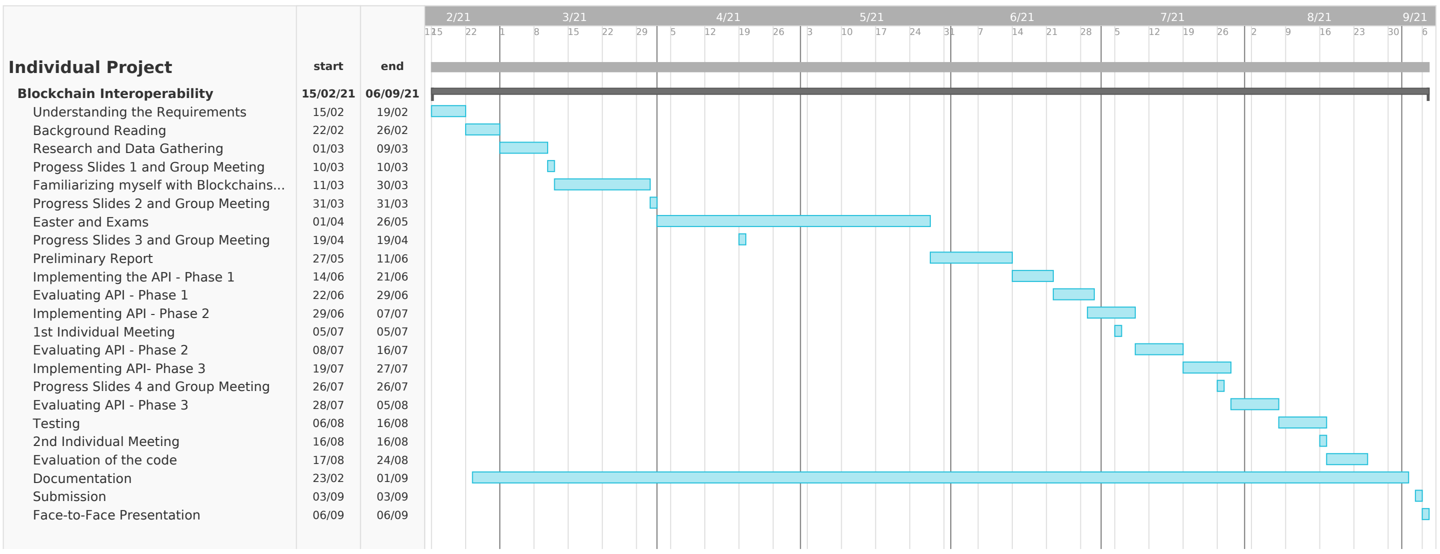
A.2 Inserting Data into Stellar

A.3 Command-line Interface

A.4 Transaction on Stellar Explorer

A.5 Data size limit on Stellar

A.6 Work Plan Gantt Chart



Individual Project	start	end
Blockchain Interoperability	**15/02/21**	**06/09/21**
Understanding the Requirements	15/02	19/02
Background Reading	22/02	26/02
Research and Data Gathering	01/03	09/03
Progess Slides 1 and Group Meeting	10/03	10/03
Familiarizing myself with Blockchains...	11/03	30/03
Progress Slides 2 and Group Meeting	31/03	31/03
Easter and Exams	01/04	26/05
Progress Slides 3 and Group Meeting	19/04	19/04
Preliminary Report	27/05	11/06
Implementing the API - Phase 1	14/06	21/06
Evaluating API - Phase 1	22/06	29/06
Implementing API - Phase 2	29/06	07/07
1st Individual Meeting	05/07	05/07
Evaluating API - Phase 2	08/07	16/07
Implementing API- Phase 3	19/07	27/07
Progress Slides 4 and Group Meeting	26/07	26/07
Evaluating API - Phase 3	28/07	05/08
Testing	06/08	16/08
2nd Individual Meeting	16/08	16/08
Evaluation of the code	17/08	24/08
Documentation	23/02	01/09
Submission	03/09	03/09
Face-to-Face Presentation	06/09	06/09

Appendix B: User Guide

The User Guide on how to use the Blockchain Interoperability Framework is very simple. The steps to use the framework have been described below:

1. The framework can be run by using the terminal or even by using an IDE.

2. Before running the framework, user must follow the “readme.md” file to setup the system for all the dependencies.

3. Once setup is complete, user can run the command “python3 cli.py” in the terminal; three choices namely “Insert Data”, “Fetch Data” and “Transfer Data” will show up. User can navigate using the navigation keys and press “Enter” key to choose the task to perform.

4. If “Insert Data” is chosen, the user will be prompted to choose the Blockchain to store data in (this can be chosen in the same manner as the tasks). Next, the user will be prompted to input the data to store in the Blockchain. Finally, the user will be prompted to input the mandatory credentials for the chosen Blockchain. If the transaction is successful, a transaction hash will be printed otherwise an error will be returned.

5. If “Fetch Data” is chosen, the user will be prompted to input the transaction hash of the data to retrieve. If the transaction hash is found, the data will be printed otherwise an error will be returned.

6. If “Transfer Data” is chosen, the user will be prompted to input the transaction hash of the data to transfer. Next, the user will be prompted to choose the Blockchain to transfer to. Finally, the user will be prompted to input the mandatory credentials for the recipient Blockchain. If the transfer is successful, data and transaction hash will be printed otherwise an error will be returned.

MSc project coordinator note: Appendix C (source code) removed as it proved too challenging to anonymise.