ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΠΡΟΗΓΜΕΝΕΣ ΤΕΧΝΟΛΟΓΙΕΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

<< Hyperledger Fabric: Μελέτη, υλοποίηση και σύγκριση >>

Χαράλαμπος Π. Κωνσταντινίδης
Α.Μ. mscacs22012

Εισηγητής: Δημήτριος Κόγιας, Επισκέπτης Καθηγητής

Αθήνα, Ιούλιος 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΠΡΟΗΓΜΕΝΕΣ ΤΕΧΝΟΛΟΓΙΕΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

<< Hyperledger Fabric: Μελέτη, υλοποίηση και σύγκριση >>

Χαράλαμπος Π. Κωνσταντινίδης
A.M. mscacs22012

Μέλη Εξεταστικής Επιτροπής συμπεριλαμβανομένου και του Επιβλέποντος.Η παρούσα μεταπτυχιακή διπλωματική εργασία εγκρίθηκε την 29/7/2024 από την κάτωθι τριμελή επιτροπή αξιολόγησης.

Α’ Μέλος	Β’ Μέλος	Γ’ Μέλος
Κόγιας Δημήτριος Επιβλέπων	**Λελίγκου Αικατερίνη - Ελένη**	**Βλάχος Βασίλειος**
Επισκέπτης Καθηγητής, Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής	Καθηγήτρια, Τμήμα Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής του Πανεπιστημίου Δυτικής Αττικής	Επίκουρος Καθηγητής, Τμήμα Οικονομικών Επιστημών του Πανεπιστημίου Θεσσαλίας
ΥΠΟΓΡΑΦΗ Dimitrios Kogias Digitally signed by Dimitrios Kogias Date: 2024.09.11 14:43:22 +03'00'	ELENI AIKATERINI LELIGKOU Digitally signed by ELENI AIKATERINI LELIGKOU Date: 2024.09.24 10:59:31 +03'00'	VASILEIOS VLACHOS 22/09/2024 21:45

UNIVERSITY OF WEST ATTICA

SCHOOL OF ENGINEERING
DEPARTMENT OF INFORMATION AND COMPUTER ENGINEERING

POSTGRADUATE PROGRAM ADVANCED COMPUTER SYSTEMS TECHNOLOGIES

DIPLOMA THESIS

<< Hyperledger Fabric: Study, deployment and comparison>>

Charalampos P. Konstantinidis
mscacs22012

Advisor: Kogias Dimitrios, Visitng Professor

Athens, July 2024

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Η κάτωθι υπογεγραμμένος Κωνσταντινίδης Χαράλαμπος του Παναγιώτη, με αριθμό μητρώου mscacs22012, φοιτητής του Προγράμματος Μεταπτυχιακών Σπουδών «Προηγμένες Τεχνολογίες Υπολογιστικών Συστημάτων», του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, της Σχολής Μηχανικών, του Πανεπιστημίου Δυτικής Αττικής, βεβαιώνω ότι: «Είμαι συγγραφέας αυτής της μεταπτυχιακής διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο.

Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης, αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου.»

Ο Δηλών

Κωνσταντινίδης Χαράλαμπος

(υπογραφή)

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα μεταπτυχιακή διπλωματική εργασία ολοκληρώθηκε μετά από επίμονες προσπάθειες, σε ένα ενδιαφέρον γνωστικό αντικείμενο όπως αυτό του Distributed Ledger Technology.

Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου κ. Κόγιας Δημήτριος, τον οποίο θα ήθελα να ευχαριστήσω θερμά για την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα και να αποκτήσω νέες γνώσεις, καθώς και για την πολύτιμη βοήθειά του και την καθοδήγησή του κατά τη διάρκεια εκπόνησής της.

Θα ήθελα επίσης να ευχαριστήσω την αγαπημένη μου σύντροφο και σύζυγο και τις αγαπημένες μου κόρες, για την κατανόηση και την υπομονή τους κατά τη διάρκεια των μεταπτυχιακών μου σπουδών.

ΑΦΙΕΡΩΣΗ

Στην Δήμητρα, την Μαριάννα και την Εύα.

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία παρουσιάζει μια ολοκληρωμένη επισκόπηση του Hyperledger Fabric, εμβαθύνοντας στην αρχιτεκτονική, τα δομικά του στοιχεία και τις λειτουργίες του. Το πρώτο μέρος της εργασίας εστιάζει σε μια λεπτομερή ανάλυση των βασικών στοιχείων, της ροής εργασιών και των ρόλων που παίζουν οι επιμέρους οντότητες όπως ο Πάροχος Υπηρεσιών Μέλους, οι ομότιμοι κόμβοι και η υπηρεσία ordering στο Hyperledger Fabric. Επιπλέον, αυτή η μελέτη επεκτείνεται και περιλαμβάνει τα Hyperledger Iroha και Sawtooth, παρέχοντας μια ενδελεχή εξέταση και σύγκριση με το Fabric.

Στο δεύτερο μέρος, η μελέτη παίρνει μια πρακτική τροπή καθώς αναπτύσσεται ένα δίκτυο Hyperledger Fabric. Μέσα σε αυτό το δοκιμαστικό δίκτυο, ένα έξυπνο συμβόλαιο υποβάλλεται σε ελέγχους και αξιολογείται η απόδοσή του. Η αξιολόγηση υπερβαίνει το Fabric, επεκτείνοντας σε μια εικονική μηχανή Ethereum, επιτρέποντας μια ολοκληρωμένη αξιολόγηση της λειτουργικότητας και της απόδοσης του έξυπνου συμβολαίου σε διαφορετικά περιβάλλοντα blockchain. Με τη βοήθεια του Hyperledger Caliper, θα πραγματοποιηθεί συγκριτική αξιολόγηση απόδοσης της αλυσίδας μπλοκ Hyperledger Fabric και μιας αλυσίδας μπλοκ Ethereum.

ABSTRACT

This thesis presents a comprehensive overview of Hyperledger Fabric, delving into its architecture, components, and functionalities. The first part of the thesis focuses on a detailed analysis of key components, workflow, and the role of entities such as Membership Service Provider, peer nodes, and the ordering service in Hyperledger Fabric. Furthermore, this exploration extends to include Hyperledger Iroha and Sawtooth, providing a thorough examination and comparison with Fabric.

The Hyperledger Foundation serves as a leading consortium for the development of open-source blockchain frameworks and tools. Chapter 1 explores the foundation's mission, governance structure, and its role in advancing enterprise blockchain adoption. It delves into the diverse range of projects under the Hyperledger umbrella, highlighting key initiatives and collaborative efforts driving innovation in the blockchain space. Chapter 2 focuses on Hyperledger Fabric, one of the flagship frameworks developed by the Hyperledger community. Fabric offers a modular and scalable architecture, catering to the diverse needs of enterprise blockchain applications. This chapter delves into Fabric's core components, including its permissioned network model, identity management system, and smart contract capabilities. It also explores Fabric's privacy and confidentiality features, consensus mechanisms, and real-world use cases across various industries.

In Chapter 3, the spotlight shifts to Hyperledger Iroha, a distributed ledger platform designed for simplicity and ease of integration. This chapter examines Iroha's approach to privacy, access control, and smart contract execution, offering insights into its design principles and use case scenarios. Chapter 4 delves into Hyperledger Sawtooth, another prominent blockchain framework renowned for its modular design and flexible architecture. The chapter explores Sawtooth's transaction execution model, scalability features, and its role in enabling diverse use cases ranging from supply chain management to healthcare data sharing.

Finally, Chapter 5 presents a comprehensive comparison of Hyperledger Fabric, Iroha, and Sawtooth, analyzing their respective strengths, weaknesses, and suitability for different enterprise blockchain applications. Key factors such as consensus mechanisms, smart contract capabilities, privacy features, and scalability are evaluated to provide readers with a holistic understanding of these frameworks and their implications for blockchain adoption in various industries.

In the second part, the study takes a practical turn as a Hyperledger Fabric testnet is deployed. Within this testnet, a smart contract undergoes rigorous testing, and its performance is evaluated. The assessment goes beyond Fabric, extending to an Ethereum Virtual Machine, allowing for a comprehensive evaluation of the smart contract's functionality and performance across different blockchain environments. With the help of Hyperledger Caliper, a performance benchmarking of Hyperledger Fabric blockchain and an Ethereum blockchain will conducted .Caliper is a blockchain performance benchmark framework, which allows users to test different blockchain solutions with predefined use cases, and get a set of performance test results. The chapter will delve into the setup and configuration of Caliper for Fabric and Ethereum networks, providing insights into the test scenarios and parameters used to benchmark these platforms. Through rigorous testing and analysis, readers will gain valuable insights into the comparative performance of Fabric and Ethereum, helping them make informed decisions when selecting a blockchain framework for their enterprise applications.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Distributed Ledger Technology
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: blockchain, Hyperledger Fabric, DLT, benchmark

Contents

Figures and Tables

Figure 7.9 - Round 4 metrics

Figure 7.10 - Round 4 Success Rate

Acronyms

BaaS	Blockchain As A Service
BFT	Byzantine Fault Tolerance
CAs	Certificate Authorities
CFT	Crash Fault-Tolerant
dApps	Decentralized Applications
DLT	Distributed Ledger Technology
EVM	Ethereum Virtual Machine
IAM	Identity And Access Management
IBFT	Istanbul Byzantine Fault Tolerance
MSP	Membership Service Provider
MVCC	Multi-Version Concurrency Control
PBFT	Practical Byzantine Fault Tolerance
PoA	Proof Of Authority
PoET	Proof Of Elapsed Time
PoS	Proof Of Stake
RA	Registration Authority
SDK	Software Development Kit
TEEs	Trusted Execution Environments
YAC	Yet Another Consensus Algorithm

Chapter 1 - The Hyperledger Foundation

Before getting into what Hyperledger Fabric is, one needs to understand what Hyperledger is specifically. Hyperledger stands as a pioneering collection of open-source projects dedicated to advancing the development of blockchain-based distributed ledgers. The overarching goal is to establish essential frameworks, standards, tools, and libraries that empower the creation of robust blockchains and associated applications. This chapter delves into the formation and structure of Hyperledger, emphasizing its pivotal role in supporting and hosting these projects through the Hyperledger Foundation.

1.1 Overview

While Hyperledger isn't an organization in the traditional sense, a cryptocurrency network, or a standalone blockchain system, its significance lies in its role as a collaborative hub[1]. Unlike cryptocurrency-focused networks such as Bitcoin, Hyperledger doesn't support its own cryptocurrency. Instead, it serves as a foundational infrastructure and standards provider for the development of diverse blockchain-based systems and applications tailored for industrial use. Think of Hyperledger as an inclusive ecosystem, fostering collaboration among diverse projects. Under its umbrella, these projects adhere to a shared design philosophy, ensuring interoperability and adherence to defined standards. The absence of a native cryptocurrency aligns with its enterprise-focused mission, emphasizing the development of practical solutions for industrial applications.

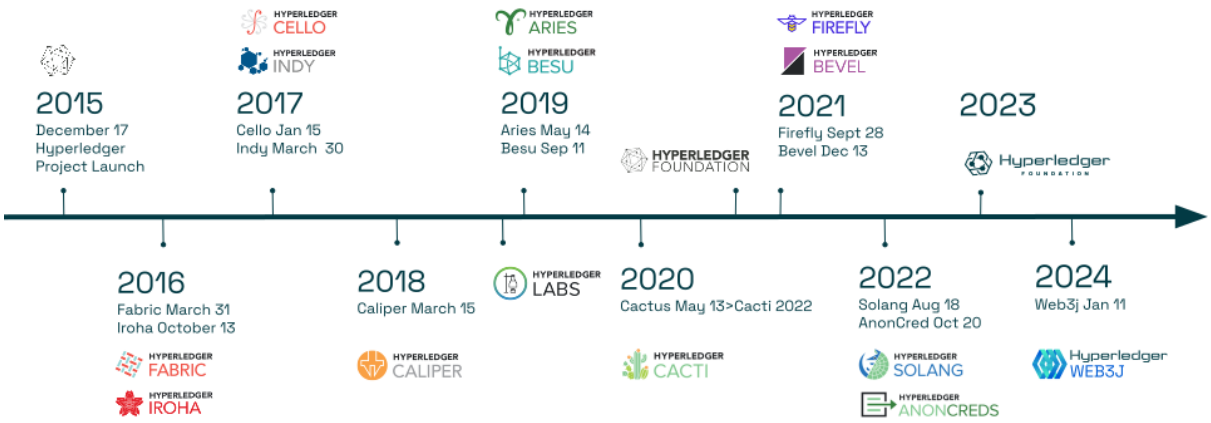

Figure 1.1 - Hyperledger Foundation Business Development Call - Feb 2024[2]

1.2 Hyperledger Technology Layers

In terms of the architecture, Hyperledger uses the following key business components:

Consensus Layer: At the core of Hyperledger's architecture is the consensus layer, a critical component responsible for establishing agreement on the order and validating the correctness of transactions within a block. This layer ensures that all participants in the network are aligned on the sequence of transactions, fostering a unanimous and secure ledger.

Smart Contract Layer: The smart contract layer is dedicated to processing transaction requests and authorizing only those deemed valid. This layer plays a pivotal role in automating and executing predefined contractual agreements, enhancing the efficiency and reliability of the transaction processing mechanism.

Communication Layer: Facilitating peer-to-peer message transport, the communication layer ensures seamless interaction between network participants. This layer is vital for maintaining

connectivity and enabling the swift and secure exchange of information across the distributed ledger.

Identity Management Services: In the realm of Hyperledger, identity management services are integral for maintaining and validating the identities of users and systems. Establishing trust on the blockchain is contingent upon the accurate and secure identification of participants, making this layer foundational for the overall security and integrity of the system.

API (Application Programming Interface): The API layer acts as a bridge, enabling external applications and clients to interface seamlessly with the blockchain. This layer facilitates interoperability, allowing for the integration of diverse applications, systems, and clients, thereby expanding the scope and utility of the Hyperledger framework.

1.3 Hyperledger Projects

Hyperledger employs an umbrella strategy by fostering distributed ledger frameworks, libraries, smart contract engines, and different corporate blockchain technologies. The project places a strong emphasis on distributed ledger component innovation and reusability.A few of them are listed below[2].

Hyperledger Besu: Besu emerges as a cornerstone within the Hyperledger ecosystem, presenting an open-source enterprise blockchain platform. Leveraging the Ethereum Virtual Machine (EVM), it not only supports Ethereum-based applications but goes beyond, incorporating features tailored to meet the stringent requirements of enterprise environments. This platform exemplifies adaptability and flexibility in constructing permissioned networks.

Hyperledger Indy: Diving into decentralized identity management, Hyperledger Indy offers a specialized distributed ledger. It equips developers with libraries, reusable components, and tools essential for creating robust digital identities on the blockchain. Within the expansive

Hyperledger umbrella, Indy plays a pivotal role in advancing secure and decentralized identity solutions.

Hyperledger Fabric: Standing out as a resilient, scalable, and flexible distributed ledger platform, Hyperledger Fabric takes center stage. With a modular architecture, it showcases pluggable implementations of diverse components, providing a solution tailored to the intricate needs of various economic ecosystems. This exploration delves into the unique attributes that make Hyperledger Fabric a standout player within the Hyperledger project.

Hyperledger Sawtooth: Addressing the critical balance between smart contract safety and decentralization, Hyperledger Sawtooth takes its place as an enterprise blockchain platform. Its innovative approach isolates the core system from the application domain, granting developers the freedom to articulate business rules in their preferred programming language. This chapter uncovers the features that set Hyperledger Sawtooth apart in the realm of distributed ledger technologies.

Hyperledger Iroha: A straightforward distributed ledger technology, draws inspiration from the Japanese Kaizen principle of eliminating excessiveness (muri). It offers fundamental functionalities for managing assets, information, and identities, while also serving as a reliable and efficient crash fault-tolerant tool for enterprise needs.

Figure 1.2 - Hyperledger projects - Feb 2024[3]

1.4 Hyperledger's Robust Ecosystem

Hyperledger has garnered substantial support from a diverse array of organizations, exceeding 250 and continually expanding[4]. The roster of supporters encompasses major players in the tech industry, including Airbus, Daimler, IBM, SAP, Huawei, Fujitsu, Nokia, Samsung, and financial giants like American Express and JP Morgan. Their endorsement highlights the broad applicability of Hyperledger across diverse sectors and industries. In addition to established enterprises, blockchain-focused startups such as Blockstream and ConsenSys contribute to Hyperledger's ecosystem. Their involvement underscores the collaborative nature of Hyperledger, where both seasoned industry leaders and pioneering startups find common ground for advancing blockchain technology.

The assertion regarding the absence of a Hyperledger coin[5] emphasizes the strategic direction of Hyperledger. By steering clear of cryptocurrency pursuits, Hyperledger mitigates

political challenges associated with maintaining a globally consistent currency. This decision fortifies the foundation of the Hyperledger Project, emphasizing its commitment to industrial applications of blockchain technology rather than speculative currency-driven schemes. The deliberate choice to abstain from promoting a cryptocurrency sets Hyperledger apart, aligning its goals with the development of practical and industrial blockchain applications. This strategic move shields Hyperledger from the volatility and controversies often associated with currency-backed blockchains, consolidating its focus on fostering innovative solutions for real-world challenges.

Chapter 2 - Fabric

This chapter unravels the complexities of Hyperledger Fabric, positioning it as the blockchain framework of choice for enterprises and medium-sized companies. The journey begins with an exploration of Fabric's modular structure, serving as a foundational platform for the development of diverse blockchain-based solutions within private enterprises.

Hyperledger Fabric is a plug-and-play blockchain architecture designed to be used in private organizations. It serves as a basis for the development of blockchain-based solutions and applications. Originally developed by Digital Asset and IBM, Hyperledger Fabric is now a cooperative cross-industry project that is housed at the Linux Foundation. Fabric was the first Hyperledger project to come out of the "greenhouse" and be released in July 2017.

Networks, applications, and other blockchain-based projects can be developed on top of the framework. The purpose of Fabric is to create private blockchains that can be utilized by a single business or a collection of related organizations that connect to other blockchain implementations. Fabric's architecture is broken down to showcase its salient characteristics, each designed to meet the particular requirements of private blockchain deployments. The focus of the conversation is Fabric's dedication to privacy, which is upheld by a Membership Service Provider (MSP) via a permissioned membership structure. The importance of privacy cannot be overstated, especially in sectors where maintaining data secrecy is critical.

2.1 Features

Identity and Access Management (IAM) is of utmost importance in a Fabric network. Fabric requires all computers within its network to be identified; potential members of a network supported by Fabric must sign up and provide their identity through an MSP. A "permissioned" membership is what this is. For many sectors, maintaining data privacy is crucial, and for this reason alone, Fabric seems like a good choice. It is crucial to remember that Fabric does not mandate permissions for every component of a blockchain; instead, the decision to demand permissions rests with the network designer.

Several other distinctive features set Fabric apart, three of the more pivotal being channels, scalability, and modularity. Fabric introduces the concept of channels, allowing the partitioning of ledgers to facilitate the creation of segregated transactions for enhanced data privacy. Members of the network may create a separate set of transactions that are not visible to the larger network. This allows for more sensitive data to be segregated from nodes who do not require access. Scalability emerges as another standout feature, enabling the network to efficiently process large amounts of data with a minimal set of resources. Similar to other implementations, the system can process vast amounts of data with fewer resources even though the number of nodes participating in the network can increase quickly. This makes it possible to take the best of both worlds. A limited number of nodes can be used to start a blockchain, and it can grow as needed. Actually Fabric is one of the better performing platforms available today both in terms of transaction processing and transaction confirmation latency. According to a paper published in March of 2019, Fabric can scale up to 20000 TPS [6].

Lastly, the modular architecture of Fabric is made to support the addition and implementation of distinct components at various points in time. Many of the elements are optional and can be added later or entirely skipped without impairing functioning. The purpose of this feature is to empower an organization to decide what needs to be implemented and what can wait. The process of reaching consensus, membership services for identity, the ledger store itself, certain access APIs, and chaincode integration are a few of the modular, or "plug-and-play," components.

2.2 Key Components

After discussing the essential features of Hyperledger Fabric, it’s time to examine its architecture and corresponding key components.Hyperledger Fabric 2.0's architecture aims to offer scalability, flexibility, and modularity. It is made up of multiple parts that come together to offer an enterprise blockchain application development platform with distributed ledger capabilities.

2.2.1 Nodes

In Hyperledger Fabric, a "node" is any computing device that is a component of a certain network. They are arranged in groups and communicate with one another according to rules established by logical entities. They can be mainly divided into three categories: Client nodes, Peer nodes, and Orderer nodes.

Client Node: This node submits the transaction invocation to the endorsers on behalf of the end user. These nodes converse with ordering service nodes as well as the peer nodes. Transactional activities are started by clients who create them. A transaction is started by the client sending a PROPOSE statement to a group of peers who approve it.

Peer Node: Hyperledger Fabric is known to facilitate peer-to-peer networks, and Peer nodes are the core components of a Fabric network. They fulfill critical roles such as ledger maintenance, smart contract execution, transaction validation, and transaction endorsement before committing them to the ledger. These peers come in two distinct types: *endorsing peers* and *committing peers*. Endorsing peers engage in transaction simulation and subsequent endorsement, while committing peers execute the vital task of committing endorsed transactions to the ledger.

Orderer Node: Orderer nodes play a crucial role in managing the consensus mechanism within the Hyperledger Fabric network. These nodes are responsible for receiving endorsed transactions from peer nodes, packaging them into blocks, and establishing a total order of transactions across the network. Fabric supports various consensus algorithms, such as Kafka, Raft, and others, providing flexibility in consensus mechanisms. The primary functions of orderer nodes include receiving transaction proposals from clients, ordering them into blocks, and distributing these blocks to the endorsing and committing peers in the network. By doing so, orderer nodes establish a consistent transaction order and ensure that all peers maintain a synchronized view of the blockchain ledger. This orchestration of transaction ordering and distribution is essential for maintaining the integrity and reliability of the blockchain network[7].

2.2.2 Channels

Channels are foundational components within Hyperledger Fabric, integral for network establishment and operation[8]. Every network must include at least one channel, typically initiated with the creation of a system channel during network bootstrapping. This system channel serves as a conduit for disseminating network configuration details and membership information. Moreover, it hosts system chaincodes essential for ongoing network functionality and governance.

Beyond the system channel, developers engage with application channels, which are pivotal from a development standpoint. These channels are versatile and can be customized to suit specific application requirements. Unlike system channels, there is no limit to the number of application channels that can be created within a network. Each application channel can accommodate multiple deployed chaincodes, offering developers flexibility in structuring their decentralized applications. When creating a new channel, developers define the access permissions for its members, tailoring access levels to suit distinct roles and responsibilities. For instance, certain clients may be restricted to ledger querying capabilities, ideal for audit purposes or regulatory compliance. Overall, channels in Hyperledger Fabric provide a mechanism for organizing network participants and governing access to shared resources, contributing to the platform's robustness and adaptability in diverse enterprise environments.

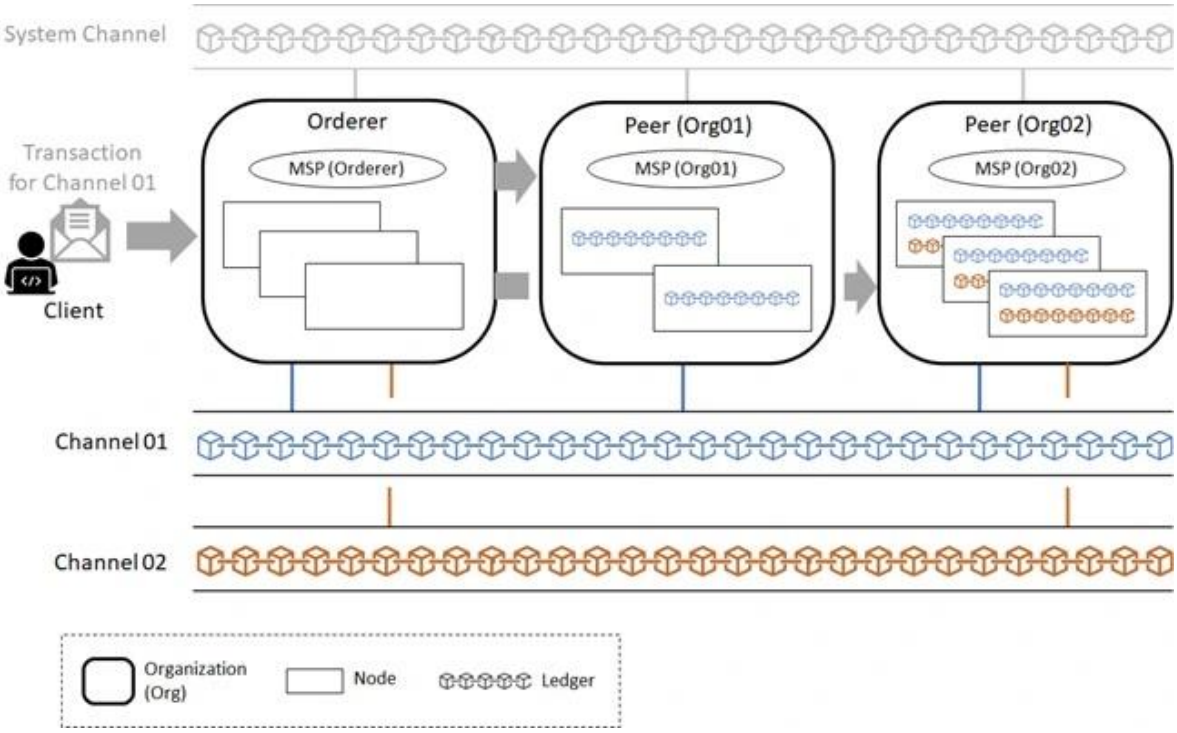

Figure 2.1 - Channels in Fabric network [9]

2.2.3 Chaincode

In the realm of blockchain technology, the term chaincode primarily finds its application within the Hyperledger Fabric platform, denoting the smart contracts responsible for defining a blockchain network's business logic and regulations. These code segments hold paramount significance in executing predetermined instructions and automating processes within the blockchain. Chaincode serves as a cornerstone in ensuring the efficient, secure, and tailored operation of blockchain networks, aligning precisely with the requirements of a given application.[10]

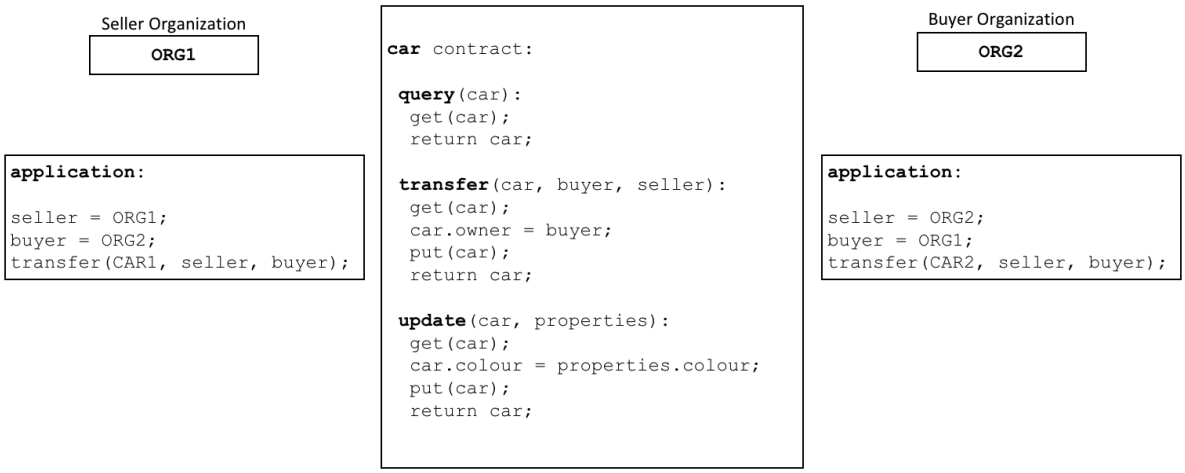

Figure 2.2 - Chaincode example [11]

Chaincode essentially embodies Hyperledger Fabric's rendition of smart contracts, serving as the vehicle to encapsulate business logic. It governs the access and modification of ledger data, facilitating the automation of intricate processes. Chaincode offers versatility by supporting various programming languages such as Go, Node.js, or Java and is deployed to endorser nodes and network channels, thereby furnishing specific functionalities for diverse blockchain applications. Its adaptable nature empowers it to cater to an array of industry requirements and use cases. Chaincode serves as the backbone for decentralized applications (dApps) on Hyperledger Fabric, it is responsible for managing the application's state and ensuring the consistent execution of agreed-upon terms. It operates as the transaction processor, implementing the rules collectively endorsed by transaction-involved parties. Chaincode life cycle includes approval, activation, and installation. This guarantees that, prior to execution, chaincode is appropriately deployed, examined, and approved by the relevant network participants. These essential Fabric components enable the safe and decentralized execution of business operations by defining the guidelines for transaction processing, data validation, and state modifications.

2.2.4 Membership Services Provider (MSP)

Figure 2.3 - Fabric MSP as a real world scenario [12]

The Membership Service Provider is a pivotal component responsible for managing identity and access control policies for network participants within the Hyperledger Fabric framework. It plays a crucial role in authenticating and authorizing participants, thereby ensuring secure interactions within the network. MSPs facilitate diverse membership models, including Certificate Authorities (CAs) and token-based systems, thereby enabling flexible management of network memberships. As a module integrated within Hyperledger Fabric, MSP oversees the management of identities, certificates, and cryptographic materials for network entities, encompassing peers, orderers, clients, and administrators.

MSP handles the creation, issuance, and revocation of digital certificates that serve as the identity credentials for network participants. These certificates are used to prove the authenticity and authority of an entity to access the network and interact with other peers. It relies on CAs to issue certificates to network participants, employing two distinct types of CAs[13]:

Registration Authority (RA): The RA is responsible for authenticating users and clients, as well as registering their identities with the CA. It validates the identity of entities requesting certificates, ensuring the integrity of the certificate issuance process.

Intermediate Certificate Authority (Intermediate CA): Intermediate CAs function to issue certificates on behalf of the root CA. They are often deployed to establish a hierarchical

structure for certificate issuance, enhancing both security and manageability within the network.

Moreover, Hyperledger Fabric supports the creation of multiple MSPs within a single network, with each MSP representing a distinct organization. This capability enables the establishment of multi-organization consortium networks, facilitating separate identity management for each participating organization.

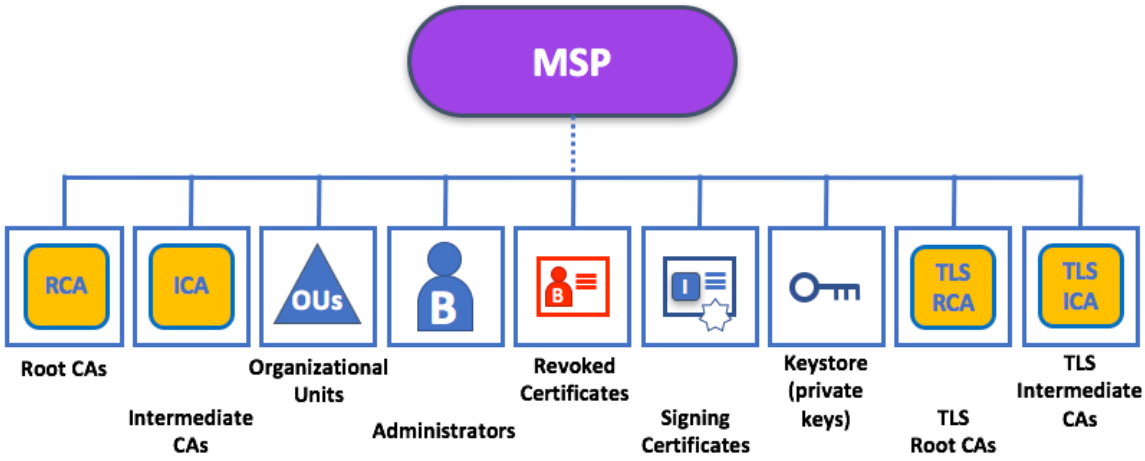

Figure 2.4 - Fabric MSP diagramm [13]

2.2.5 Ledger

In Hyperledger Fabric, the distributed ledger serves as a tamper-proof repository that records all network transactions. It comprises two essential components: the blockchain (‘chain’) and the state database. The state database captures the current state of assets, reflecting their latest values and configurations. On the other hand, the chain guarantees the integrity and transparency of the ledger by providing an immutable record of all transactions executed within the network. Together, these components form a robust foundation for maintaining an accurate and trustworthy ledger in Hyperledger Fabric[14].

The chain serves as a transaction log organized into hash-linked blocks. Each block contains a sequence of transactions, and its header includes a hash of the block's transactions and a hash of the preceding block's header. This structure ensures that all transactions are

chronologically ordered and cryptographically linked together. As a result, the integrity of the ledger is maintained, as any attempt to tamper with the data would require breaking these hash links. The hash of the latest block encompasses all preceding transactions, ensuring that all peers have access to a consistent and trustworthy ledger state.The chain is stored on the peer's file system, which can be either local or attached storage. This storage mechanism efficiently supports the append-only nature of blockchain workloads, allowing new transactions to be added to the chain while preserving the historical integrity of the ledger.

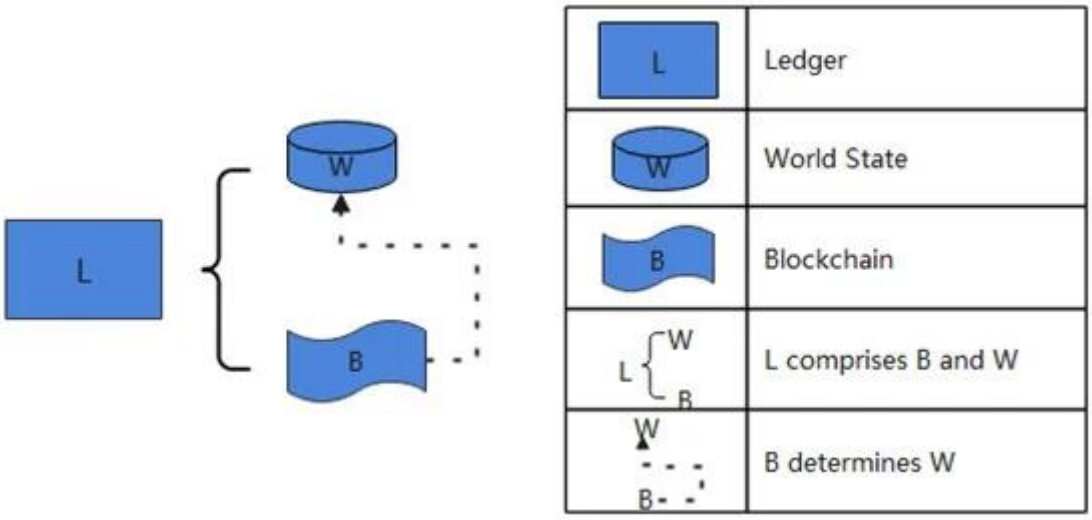

Figure 2.5 - Fabric Ledger explained [14]

The ledger's current state data, also known as World State, represents the latest values for all keys ever included in the chain transaction log. This current state is pivotal for efficient chaincode invocations, as transactions are executed against this data. To ensure the efficiency of chaincode interactions, the latest values of all keys are stored in a state database. This state database acts as an indexed view into the chain's transaction log and can be regenerated from the chain whenever necessary. Upon peer startup, the state database automatically recovers or generates the latest data before accepting transactions. Hyperledger Fabric provides two options for the state database: LevelDB and CouchDB. LevelDB, embedded in the peer process by default, stores chaincode data as key-value pairs. Alternatively, CouchDB serves as an optional external state database, offering additional query support, particularly beneficial

when chaincode data is modeled as JSON. This enables rich queries of the JSON content, enhancing flexibility and usability[14].

2.2.6 Consensus

In distributed ledger technology, the concept of consensus has often been narrowly associated with a specific algorithm focused solely on agreeing upon the order of transactions. However, Hyperledger Fabric emphasizes a broader understanding of consensus, recognizing its fundamental role in the entire transaction flow. Consensus in Hyperledger Fabric encompasses more than just transaction ordering[15]; it extends to the complete verification of a set of transactions, from proposal and endorsement to ordering, validation, and commitment. In essence, consensus in Fabric represents the comprehensive validation of the accuracy of a block composed of transactions.

While many distributed blockchains like Ethereum and Bitcoin operate on a permissionless basis, allowing any node to participate in the consensus process, Hyperledger Fabric adopts a distinct approach. In Fabric, transaction ordering is handled by the specialized node known as the orderer, which, along with other orderer nodes, constitutes an ordering service. Unlike the probabilistic consensus algorithms utilized in permissionless blockchains, Fabric employs deterministic consensus algorithms. This means that any block validated by a peer is assured to be both final and accurate. Consequently, the ledger in Hyperledger Fabric cannot experience forks, as is common in other distributed and permissionless blockchain networks.

As mentioned, consensus in Fabric extends beyond simply agreeing on the order of transactions within a block. It involves a series of checks and balances throughout the transaction lifecycle to ensure the integrity and security of the ledger. The consensus in the Hyperledger Fabric network is a result of the cooperation of different nodes, the whole transaction flow and differs radically from the public blockchain probabilistic consensus mechanisms. There are basically three mechanisms that take part in reaching the consensus. *Endorsement policies* dictate which specific members must endorse a transaction, ensuring that it meets the required criteria before it can be committed to the ledger. System chaincodes enforce these policies, verifying that sufficient endorsements are present and

derived from the appropriate entities. Additionally, a *versioning check* ensures agreement on the current state of the ledger before appending new blocks, guarding against double spend operations and other threats to data integrity. *Identity verifications* are ongoing throughout the transaction flow, with access control lists implemented at various hierarchical layers of the network. Payloads are repeatedly signed, verified, and authenticated as transaction proposals traverse different architectural components. The ordering service is not responsible for the content of transactions but for the order of transactions. It also forms the blockchain - a single, final source of truth in the network.

2.2.6.1 Endorsement policies

Endorsement policies in Hyperledger Fabric determine the criteria for endorsing transactions before they can be committed to the ledger. These policies mandate that each transaction must receive an endorsement from a specified number of endorsing peers in accordance with the endorsement policy[16]. The endorsement policies can be configured based on either the number of signatures required or the specific identities of the endorsing peers. This flexibility allows organizations to tailor endorsement policies according to their specific security and governance requirements. Example endorsement policies might be:

- *Peers A, B, C, and F must all endorse transactions of type T*
- *A majority of peers in the channel must endorse transactions of type U*
- *At least 3 peers of A, B, C, D, E, F, G must endorse transactions of type V*

Each chaincode has an endorsement policy that dictates what organizations need to execute the smart contracts. The results are signed by the peers. If the endorsement policy is satisfied and all peers return the same value, the content of the transaction is considered valid.

2.2.6.2 Multi-version concurrency control (MVCC)

In the context of Hyperledger Fabric, concurrency control mechanisms detect and handle concurrent reads and writes to the same blocks so that ledger consistency is maintained. If multiple transactions attempt to modify the same data simultaneously, potential conflicts may arise. This mechanism ensures that the versions of keys read during the endorsement phase of a transaction remain unchanged during the validation phase. By doing so, it prevents scenarios where old values are read after they have been modified by another concurrent transaction. If a transaction attempts to validate while another transaction has already updated the versions of the keys listed in its read set, the MVCC validation will fail.

In the case of a read-write conflict, where two clients attempt to update and read the same key simultaneously, MVCC validation plays a crucial role. For instance, if user1 submits TX1 and user2 submits TX2 concurrently, both transactions may read and update the same value (Value1) associated with Key1. After endorsement and ordering, TX1 modifies the value and version of Key1 to value1 and version1. However, during MVCC validation, TX2 fails because the version of Key1 in its read set does not match the version in the ledger. As a result, the MVCC validation identifies the inconsistency between the ledger and the endorsement result, leading to an MVCC error[17].

2.2.6.3 Ordering mechanism

The ordering phase comes after the Endorsement phase; it agrees to the sequential order of the execution of the Hyperledger decided earlier. The ordering mechanism in Hyperledger Fabric 2.0 comprises three distinct implementations: Raft, Kafka, and SOLO. The Kafka and SOLO implementations are deprecated in v2.x[18]

Raft: Raft is a crash fault-tolerant (CFT) ordering service using the Raft protocol within etcd[19]. Operating on a "leader and follower" model, Raft elects a leader node per channel, whose decisions are then replicated by follower nodes[20]. Unlike Kafka-based ordering services, Raft offers simpler setup and management, making it more accessible for users. Additionally, Raft's design facilitates contributions from different organizations[21], allowing

for the establishment of a distributed ordering service comprising nodes from multiple entities.

Raft has emerged as the consensus mechanism of choice for numerous Hyperledger Fabric deployments, owing to several compelling advantages. Firstly, its simplicity sets it apart, offering a straightforward understanding, implementation, and debugging process in comparison to more complex consensus algorithms like Paxos. This simplicity accelerates the deployment process and reduces the likelihood of errors during configuration and operation. Moreover, Raft boasts impressive scalability capabilities, capable of accommodating a large number of nodes within the network. This scalability factor is pivotal for ensuring the resilience and efficiency of large-scale deployments, making it an attractive option for enterprise-grade blockchain networks. Raft excels in facilitating quick leader elections[22] swiftly replacing failed leaders to minimize disruptions and maintain network continuity. It prioritizes safety, providing robust guarantees for system consistency even in the event of network failures or node crashes. This resilience ensures that critical blockchain operations proceed smoothly and reliably, bolstering confidence in the integrity and stability of the Hyperledger Fabric network. Collectively, these advantages position Raft as a highly desirable consensus mechanism for Hyperledger Fabric deployments, offering simplicity, scalability, rapid leader election, and robust safety measures.

Kafka: Similar to Raft-based ordering, Apache Kafka is a crash fault-tolerant implementation that adopts a "leader and follower" node setup. Kafka relies on a ZooKeeper ensemble for management tasks. While Kafka-based ordering services have been accessible since Fabric v1.0, some users might perceive the added administrative burden of managing a Kafka cluster as daunting or undesirable.

SOLO: In the SOLO implementation, a single ordering node orchestrates the transaction flow, ensuring transactions are arranged chronologically and executed in an orderly manner. The Solo implementation of the ordering service is intended for testing only.

2.3 Transaction flow

Enterprises opt for Hyperledger Fabric due to its dependable and swift transaction processing. Unlike public blockchains, which typically employ the *order-execute-validate* method, Hyperledger Fabric adopts the *execute-order-validate* approach for transaction execution. In this method, transactions are initially executed, often in parallel as requested. Once executed, these transactions are then sent to an ordering service for sequencing[23]. Below is a breakdown of the entire transaction process.

When a transaction is initiated, the end user utilizes the client application to submit the request to the Software Development Kit (SDK). The SDK translates the transaction request into a transaction proposal, which is then signed by the end user's distinctive cryptographic signature. Subsequently, the SDK transmits the transaction proposal to the designated peers responsible for endorsement, adhering to the specified endorsement policy. Upon receiving the transaction proposal, the endorsing peers first verify its validity. This validation process ensures that:

- The transaction proposal is properly structured.
- The transaction proposal has not been previously submitted.
- The cryptographic signature of the transaction proposal is authentic.
- The end user possesses the necessary authorization to execute the requested action on the channel or network[54].

If the transaction proposal passes these validation checks, the endorsing peers proceed to invoke the specified chaincode function. The input provided in the proposal is passed as arguments to this function, triggering its execution. However, at this stage, no changes are applied to the ledger. The proposal responses collected from all the endorsing peers are matched, and the application verifies their signatures. The transaction isn't submitted to the ordering service if the end user only queried some data.

Following the execution, the chaincode function generates an output, comprising a response value, a write set, and a read set. This output, along with the cryptographic signatures of the endorsing peers, is then dispatched back to the SDK. The SDK subsequently parses the

transaction response, making it accessible to the application. The transactions then are forwarded to the ordering service, which aggregates them into blocks. The ordering service establishes a total order of transactions across the network, ensuring consistency and determinism. Once a sufficient number of peers agree on the results of a transaction, it is added to the ledger and distributed to all peers in the network. This step marks the first time that transactions are given an ordering. Until transactions are added to the ledger, there is no concept of one transaction happening before or after another. The addition of transactions to the ledger ensures that the entire network has a consistent and agreed-upon record of all transactions, establishing a definitive order and history of events within the blockchain.

Once ordered into blocks, the transactions undergo validation by the endorsing peers. Each transaction is validated independently by a subset of peers according to the endorsement policy. The peers verify the correctness of the transaction results and ensure that they comply with the predefined rules. Validated transactions are then committed to the ledger by the committing peers. Once committed, these transactions become a permanent part of the immutable ledger and are visible to all participants in the network. This ensures transparency and trust in the system, as all network participants have access to the same verified and unalterable record of transactions.

By employing the execute-order-validate method, Hyperledger Fabric ensures a robust and efficient transaction flow, meeting the performance and reliability requirements of enterprise applications.

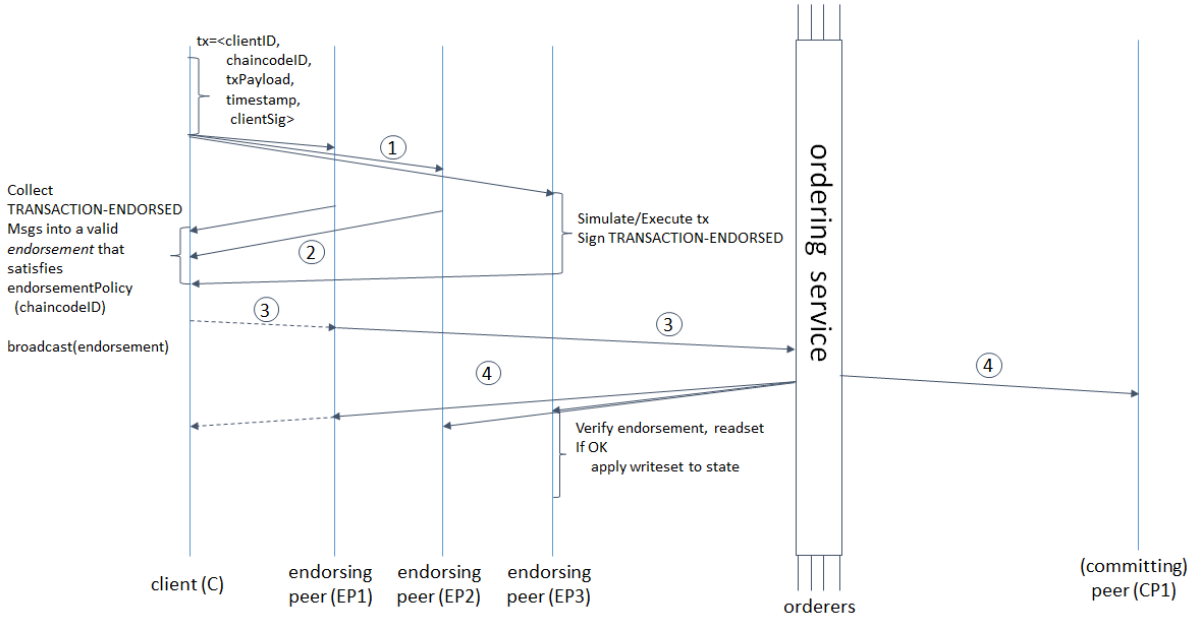

Figure 2.6 - The transaction flow in Fabric network [24]

2.3.1 Strong Data consistency and finality

Strong consistency is a fundamental property in distributed systems that guarantees all nodes within the system observe the same data concurrently, irrespective of the node accessed. Put simply, any write operation is immediately reflected in subsequent read operations across all nodes, ensuring consistency across the entire system. This ensures a linear ordering of operations, effectively rendering the system as a unified and coherent entity, despite its distributed nature[25]. Public permissionless blockchains typically use a variety of methods (such Proof of Work or Proof of Stake) to make sure that all of the blockchain nodes concur on transactions. The probabilistic finality of the state is the result of those probabilistic algorithms. It indicates that a block does not become final once it is appended to the chain; rather, it becomes final with a high probability and is reversible in the event that a network forks. In contrast, a transaction in Hyperledger Fabric has absolut finality once it is committed to the chain. Since the ordering service, which maintains the consensus, is centralized and deterministic, there are no forks. Blockchain is immutable and this absolute finality of Hyperledger Fabric blockchain is very important for enterprise applications.

Chapter 3 - Iroha

Hyperledger Iroha, another blockchain framework under The Linux Foundation's Hyperledger umbrella[26], offers a straightforward and adaptable solution for infrastructure projects seeking distributed ledger technology. Initially contributed by Soramitsu, Hitachi, NTT Data, and Colu, Iroha prioritizes simplicity and ease of integration. It targets industries like identity and finance, providing a domain-driven C++ programming language for mobile application development tailored to specific business requirements. Notably, Iroha features a modular design, emphasizing client application development and introducing the crash fault-tolerant consensus algorithm YAC (Yet Another Consensus Algorithm). This algorithm, like others, orchestrates a step-by-step process to address challenges and execute a sequence of instructions effectively[27].

3.1 Overview

Hyperledger Iroha offers a versatile set of features for blockchain development. Iroha supports cross-platform application development, allowing developers to build applications for various platforms, including mainframe and mobile. It is compatible with popular programming languages such as JS, Java, iOS, and Python. The framework seamlessly operates across multiple operating systems, including Linux, Windows, and macOS, ensuring accessibility and flexibility. It offers support for multiple keys or multi-signature functionalities, facilitating transaction settlements requiring numerous signatures. Its modular and plug-in design simplifies adoption for developers, while a comprehensive range of code libraries enhances ease of maintenance and deployment. With its modular and plug-in design, Iroha offers an intuitive and developer-friendly environment. It provides extensive code libraries for easy maintenance, deployment, and hassle-free application development. Additionally, Iroha facilitates streamlined asset management, secure control over user activities and roles, and features a modular design architecture that fosters the blockchain ecosystem.

In contrast to public blockchains like Ethereum and Bitcoin, Iroha operates as a permissioned network, restricting access to authorized participants. It does not rely on a proof-of-work mechanism for transaction verification, eliminating issues of slowness or latency associated with high transaction volumes. Although Iroha does not have a native cryptocurrency, eligible participants can generate cryptocurrencies for enterprise use within the network.

3.2 Architecture

Hyperledger Iroha comprises several key components that collectively facilitate its functionality:

Model classes represent essential system entities within the framework. Torii, acting as a gateway, serves as the primary input and output interface for clients. It operates as a single gRPC server, enabling clients to interact with peers across the network. Torii operates asynchronously, allowing for non-blocking RPC calls. It handles both commands (transactions) and queries (read access) from clients. The Network component governs the interaction among peers within the network while consensus oversees the agreement among peers regarding the content of the blockchain. Iroha utilizes the YAC mechanism, a practical Byzantine fault-tolerant(PBFT) algorithm based on voting for block hash[28].

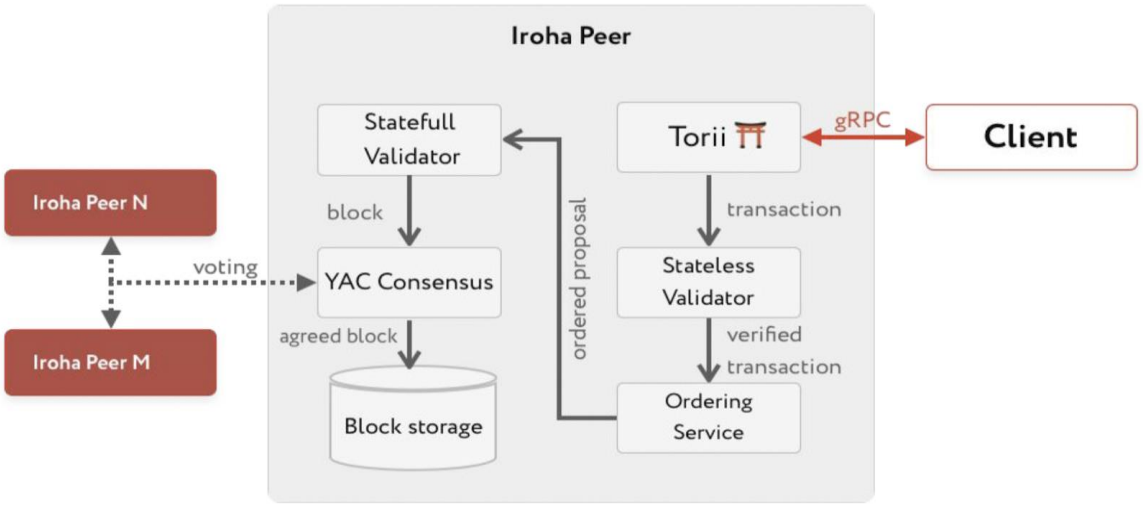

Figure 3.1 - Iroha architecture components [29]

On the business logic level, the simulator generates a temporary snapshot of storage to validate transactions by executing them against this snapshot. It verifies the validity of transactions, forming a verified proposal comprising only valid transactions, Validator classes ensure the adherence to business rules and validate the format of transactions or queries. These validators perform two distinct types of validation: *stateless* validation and *stateful* validation. Stateless validation involves quick schema and signature checks of transactions, while stateful validation involves verifying permissions and assessing the current world state view to ensure compliance with desired business rules and policies. Synchronizer facilitates the synchronization of new peers into the system or reconnects temporarily disconnected peers. Lastly, Ametsuchi serves as the ledger block storage, encompassing a block index (currently Redis), block store (currently flat files), and a world state view component (currently PostgreSQL).

In the Iroha network, three key participants play integral roles:
Clients interact with the network, accessing permitted data and executing state-changing actions known as transactions. These transactions comprise atomic operations called commands. For example, a transaction might involve transferring funds to multiple recipients through distinct commands. If the initiator lacks sufficient funds to cover all transactions within a single transaction, the entire transaction will be rejected. Peers maintain the network's current state and possess individual copies of the shared ledger. Each peer operates as a distinct entity with its own address, identity, and trust level. Iroha supports both single-peer instances and clustered setups, where different computers fulfill various functions such as ledger storage, indices management, validation, and peer-to-peer communication. The ordering service organizes transactions into a predetermined sequence. It employs various algorithms for this task, with Kafka being a favored choice. It's essential to cluster distributed solutions like Kafka to prevent a single point of failure.

3.3 Command-driven Architecture

As Iroha is a special purpose framework for asset, identity & supply chain management, it allows users to perform common functions with some prebuilt commands and queries which negates the need to write cumbersome and hard to test smart contracts, enabling developers to complete simple tasks faster and reliably with less risk.

Iroha simplifies the concept of smart contracts by transforming them into fixed commands, streamlining the development process and ensuring consistency in transaction execution. Despite this approach, Iroha remains an open-source project with an extensible design, allowing developers to enhance its functionality by adding new features. However, customizing Iroha requires a deep understanding of its codebase, as developers must navigate the framework's architecture to implement tailored solutions that meet specific use cases and requirements. Commands are utilized to initiate transactions that modify the ledger's state, representing actions like asset transfers or account creation. These commands are submitted by clients and executed by network nodes to enact changes to the ledger's data. In contrast, queries provide read-only access to the ledger, allowing clients to retrieve information such as account balances or transaction history without modifying the ledger's state. While smart contracts can enforce complex business logic on a blockchain, commands and queries in Iroha focus on managing transactions and accessing ledger data, serving as essential tools for clients to interact with the network.

Scope	Commands
Account	CreateAccount AddSignatory RemoveSignatory SetAccountQuorum SetAccountDetail
Assets	CreateAsset AddAssetQuantity SubtractAssetQuantity TransferAsset

Domains	CreateDomain
Peer	AddPeer
Permissions	CreateRole AppendRole DetachRole GrantPermission RevokePermission

Table 1 - Prebuilt commands in Iroha

Scope	Queries
Account	GetAccount GetAccountAssets GetAccountDetail GetSignatories
Transactions	GetTransactions GetAccountTransactions GetAccountAssetTransactions GetPendingTransactions
Assets	GetAssetInfo
Peer	AddPeer
Permissions	GetRoles GetRolePermissions

Table 2 - Prebuilt queries in Iroha

3.4 YAC

YAC [28], is founded on a process of voting for block hash. Once validated, blocks are subject to collaboration with other blocks to determine commit decisions and propagate them across peers. YAC's core functions comprise ordering and consensus.

Ordering orchestrates the sequential arrangement of transactions, assembling them into proposals, and disseminating them across the network to peers. The ordering service acts as the hub for establishing the transaction sequence and broadcasting proposals. It's important to note that while ordering sets the transaction order, it doesn't handle the stateful validation of transactions. However, it's crucial to acknowledge that the current implementation of the ordering service poses a single point of failure, rendering Hyperledger Iroha neither crash fault-tolerant nor Byzantine fault-tolerant. On the other hand, consensus ensures agreement on blocks based on identical proposals, facilitating the synchronized progression of the ledger.

Despite the outlined steps for achieving consensus, scenarios exist where consensus may fail. One such scenario involves a broken leader, where the leader exhibits unfair behavior in vote collection or delays in responding with a commit message. Peers mitigate this risk by setting a time limit for receiving commit messages from the leader; if the timer expires, the next peer in the order list assumes leadership. Another potential failure scenario arises from the ordering service forwarding transactions that fail stateless validation. In response, peers must remove these transactions from the proposal and recalculate the hash based on the remaining valid transactions to rectify the situation.

3.5 Transaction Flow

In the transaction flow of Hyperledger Iroha[30], a client starts by creating and sending a transaction to the Torii gate, which then routes it to a peer responsible for stateless validation. After this initial validation step, the transaction proceeds to the ordering gate, which determines the optimal strategy for connecting to the ordering service. The ordering service organizes transactions into proposals, which are essentially unsigned blocks containing ordered transactions. These proposals are then forwarded to peers in the consensus network.

Notably, proposals are dispatched only when a sufficient number of transactions have accumulated or a specific time threshold has been reached, ensuring that empty proposals are not transmitted.

Following this, each peer undergoes stateful validation in the Simulator, where the proposal's contents are verified. Subsequently, a block comprising only validated transactions is constructed by each peer. These validated blocks are then forwarded to the consensus gate, where the YAC consensus logic is applied. Based on this logic, a leader is elected, and an ordered list of peers is established. Peers participate in the consensus process by signing and sending their proposed blocks to the leader. If the leader receives a sufficient number of signed proposed blocks, it initiates a commit message, signaling that the proposed block should be applied to the chain of every participating peer. Upon receipt of the commit message, the proposed block becomes the next block in the chain of each peer, facilitated by the synchronizer component.

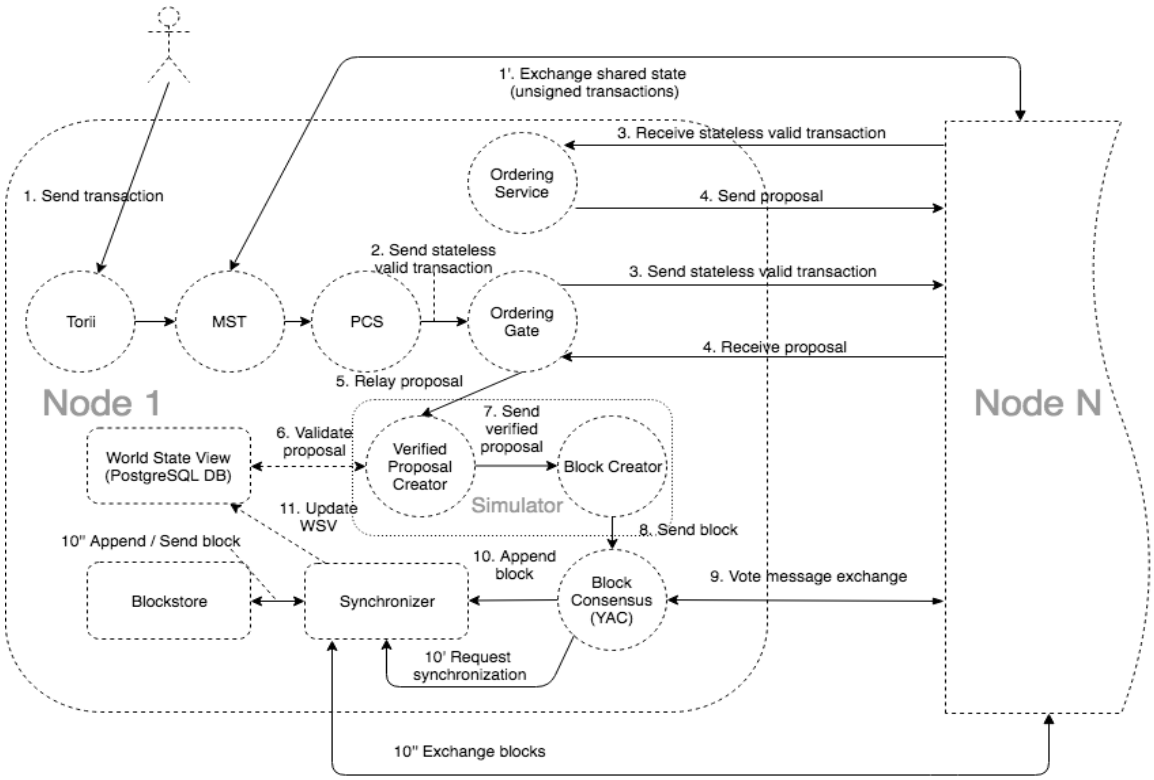

Figure 3.2 - The transaction flow in Iroha network [31]

Chapter 4 - Sawtooth

Overview

Hyperledger Sawtooth emerges as a robust corporate blockchain platform tailored to develop distributed ledger networks and applications, particularly geared towards enterprise use cases. Its design philosophy revolves around ensuring the integrity of distributed ledgers and fortifying smart contracts, catering specifically to the demands of enterprise-grade applications. Sawtooth adopts a Blockchain as a Service (BaaS) approach, offering a comprehensive suite of tools and functionalities to facilitate seamless deployment and management of blockchain networks.

The hallmark of Sawtooth lies in its modularity, which empowers companies and consortiums to tailor policies according to their domain-specific requirements. This modular architecture provides the flexibility for applications to select transactional, permissioning, and consensus algorithms that align best with their unique business needs. Unlike many existing blockchain platforms, where core functionalities and applications coexist on the same infrastructure, Sawtooth's design decouples these components, potentially enhancing both security and performance aspects of the system. Core architecture is built separately from the application domain, which allows developers to write business rules in preferred languages without knowing the internal structure of the system. This chapter delves deeper into the intricacies of Hyperledger Sawtooth, exploring its modular design, customizable features, and its significance in driving enterprise blockchain adoption.

4.1 Features

Hyperledger Sawtooth stands as a modular platform designed to construct, deploy, and operate distributed ledgers, also known as blockchains. At its heart lies the "Proof of Elapsed Time" (PoET) consensus algorithm , leveraging trusted execution environments (TEEs) to ensure fair and efficient consensus[32]. Offering a modular framework, Sawtooth accommodates pluggable consensus algorithms, catering to both permissioned and

permissionless networks. The cornerstone of Sawtooth is its distributed ledger, meticulously logging all transactions and smart contract executions. Replicated across all network nodes, this ledger facilitates parallel transaction processing for enhanced performance. Additionally, Sawtooth features a smart contract engine, simplifying the deployment and execution of smart contracts. Complementing these functionalities is a user-friendly RESTful API for seamless ledger interaction and transaction submission. The system was designed in a way that lets developers specify their own policy, rules, permissions, and consensus algorithms. Thus, you can create applications with native business logic or build smart contract-based virtual machines, and both types can run on the same blockchain.

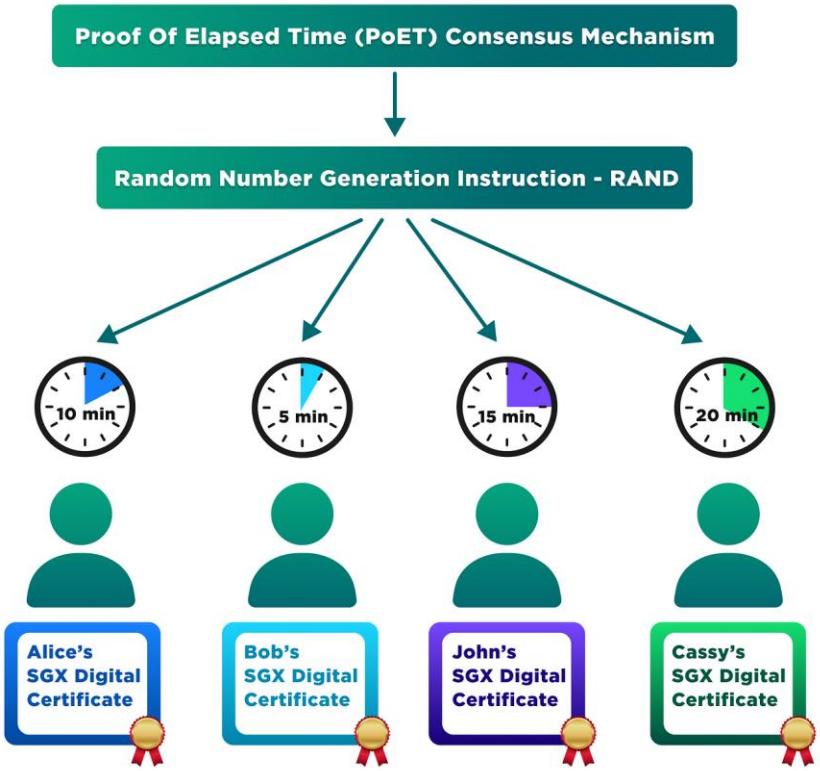

Figure 4.1 - Proof of Elapsed Time Mechanism [33]

Sawtooth is engineered for scalability, capable of supporting networks comprising thousands of nodes and processing millions of transactions per second. Its adaptability makes it suitable for a wide array of applications, including supply chain management, digital asset tracking, and voting systems[34]. By segregating the core ledger system from application-specific environments, Sawtooth streamlines application development while upholding system

security. Developers can define application-specific business rules without delving into the intricacies of the underlying ledger architecture, enabling the creation of applications in their preferred programming languages. During the development stage, the core system remains insulated from any impact stemming from resource sharing. Hyperledger Sawtooth boasts compatibility with a diverse array of programming languages such as Go, C++, Python, JavaScript, Rust, among others. This broad language support provides developers with a multitude of exciting avenues for implementing blockchain solutions and applications. This design fosters the development of applications that can be hosted, managed, and utilized independently of the core blockchain network.

The challenges of permissioned networks are addressed by enabling the deployment of node clusters with independent permissioning, eliminating the need for centralized services that might compromise data privacy. Permissions, including roles and identities, are stored on the blockchain, ensuring accessibility for all network users. Additionally, Sawtooth implements a parallel scheduler that enables transactions to be executed concurrently. It achieves transaction isolation by ensuring that each transaction's execution is independent of others, while still preserving contextual changes based on the state locations accessed by the transaction. Despite the potential for conflicts when modifying the same state multiple times, Sawtooth enables transactions to be executed in parallel, mitigating the risk of double-spending. This parallel scheduling mechanism[35] provides a substantial speed boost compared to serial execution, enhancing the overall performance of the system. Moreover, Sawtooth's compatibility with Ethereum is enhanced through the Seth[36] integration project, allowing for the deployment of EVM smart contracts on the Sawtooth platform. This interoperability expands Sawtooth's capabilities, enabling developers to leverage Ethereum's ecosystem while benefiting from Sawtooth's performance and scalability features.

4.2 Consensus algorithms and dynamic consensus

Hyperledger Sawtooth offers a flexible architecture with support for various pluggable consensus algorithms to meet diverse application needs. The default consensus algorithm, PoET, leverages TEEs to ensure fair and efficient consensus. PoET[37] employs random leader

election and wait time mechanisms for leader selection, providing equal opportunities to all participants.Sawtooth also supports the Practical Byzantine Fault Tolerance algorithm, well-suited for permissioned networks with known participants, as it employs a replica voting process to achieve consensus. Raft, another supported algorithm, is designed for distributed systems with dynamic changes, utilizing a leader-based approach for efficient network partition recovery. For testing and development purposes, Sawtooth offers the Devmode consensus mechanism, enabling parallel transaction processing without requiring consensus. Other advanced algorithms like Proof of Authority (PoA) and Proof of Stake (PoS) can be integrated by installing corresponding consensus engines.

Sawtooth's architecture effectively decouples consensus from transaction semantics, abstracting the core concepts underlying the consensus process. By providing a consensus interface through which consensus engines communicate with the validator via the consensus API, Sawtooth enables the integration of diverse consensus implementations. This modular approach allows for the seamless plugging in of various consensus mechanisms, empowering users to tailor the consensus protocol to their specific requirements and preferences.

4.3 Architecture

Sawtooth operates on an asynchronous client/server pattern, where clients send requests to the server and receive 0 or more replies in response. The server can process multiple requests and replies simultaneously, providing a flexible interaction model. The platform also offers a RESTish API for convenient interaction with the validator, supporting common JSON/HTTP standards. This API serves as a separate process, allowing clients to submit transactions and retrieve block data through a language-neutral interface. Errors in transaction processing are communicated back to clients via a JSON envelope, containing standardized error codes and messages. Additionally, query parameters are supported to customize request formation, and endpoints provide references to ledger resources such as blocks and transactions.

Transaction processors handle business logic and validate incoming transactions, ensuring their compliance with predefined rules before adding them to the ledger. These processors

consist of a Processor class provided by the SDK and application-specific Handler classes responsible for transaction validation and execution. The architecture supports customization through additional transaction handlers, which can be invoked using the apply method or metadata method. Moreover, Sawtooth's consensus API has been revamped to operate as a separate process termed the 'consensus engine.' This modular design enhances flexibility and language independence for integrating diverse consensus algorithms. The consensus engine comprises three processors: BlockPublisher, BlockVerifier, and ForkResolver, each responsible for distinct consensus-related tasks. Validators in Hyperledger Sawtooth validate blocks and batches of transactions using predefined rules and permissions. The validation process includes verifying on-chain transaction permissions and applying on-chain block validation rules. Validators ensure network integrity and communication between nodes, managing transactions, blocks, and supporting consensus engines. Sawtooth supports both serial and parallel(default) scheduling of transactions, enabling efficient handling and execution of transactions while preventing issues such as double spending. The validator process consists of two primary components: the chain controller, which manages the blockchain's state and determines chain head updates, and the block manager and publisher, responsible for creating new candidate blocks with valid transactions.

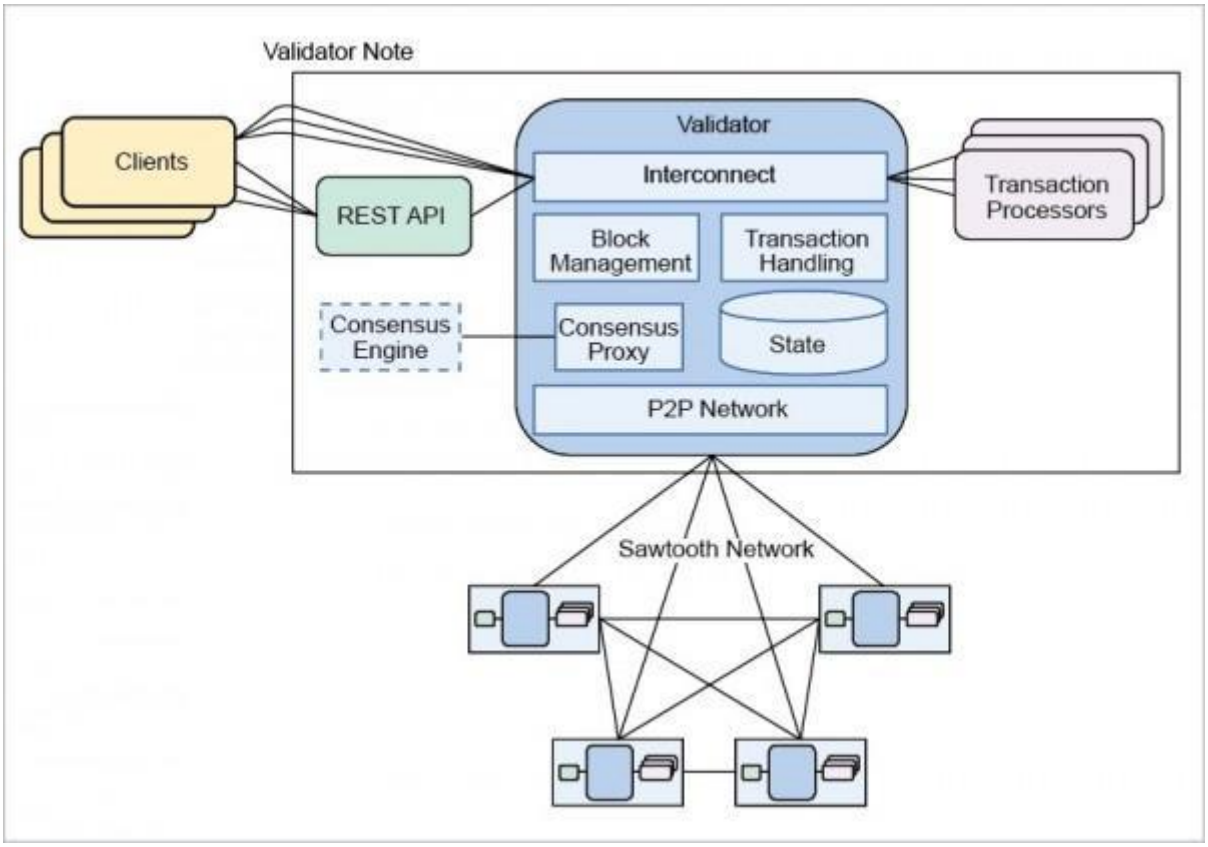

Figure 4.2 - Architecture of Sawtooth network [38]

4.4 Transaction Families

In Hyperledger Sawtooth, transaction updates within the blockchain framework are facilitated through the use of transaction families, which serve as a key component for capturing state changes and implementing transaction logic. These transaction families are essentially groups of operations or transaction types that are permitted on the shared ledger. By organizing transactions into families, Sawtooth offers a flexible approach to managing the level of versatility and risk within a network.

Transaction families act as "safer" smart contracts, providing a predefined set of acceptable smart contract templates, rather than requiring developers to create smart contracts from

scratch. This abstraction allows for greater consistency and reliability in transaction execution while reducing the potential for errors or vulnerabilities. Moreover, Sawtooth's transaction families support a variety of programming languages, including Javascript, Java, C++, Python, and Go, offering developers the flexibility to leverage their preferred language for implementing transaction logic. One of the key features of Hyperledger Sawtooth is its support for specifying the address or namespace of data within the ledger. This capability provides developers with the flexibility to define, share, and reuse data across different transaction families, enhancing interoperability and data management within the blockchain network. By allowing businesses to bring their own transaction families and adapt them to their specific needs, Sawtooth empowers organizations to tailor their blockchain solutions to suit their unique requirements and use cases.

Sawtooth offers a diverse array of transaction families, each serving a specific purpose within the blockchain framework. The *BlockInfo* transaction family, for instance, enables the storage of various information related to blocks, providing users with a means to document and track block-related data effectively. Meanwhile, the *Smallbank* transaction family is designed for testing and analyzing business quality, offering a valuable tool for evaluating the performance and reliability of blockchain-based business processes. Another essential transaction family in Sawtooth is the *Settings* family, which facilitates the storage of on-chain configurations and serves as a reference model for managing system settings efficiently. The *Validator Registry* transaction family plays a crucial role in adding validators to the system, ensuring the integrity and security of the network by managing validator nodes effectively.

The *IntegerKey* transaction family is particularly noteworthy for its ability to test deployed ledgers comprehensively without requiring additional resources, offering developers a streamlined approach to ledger testing and validation. On a lighter note, the XO transaction family injects a touch of entertainment into the network with a game of tic-tac-toe, showcasing Sawtooth's versatility in accommodating diverse use cases. The *Identity* transaction family focuses on preserving permissioned data for validators, storing crucial information such as public keys to maintain the integrity and security of the network. Lastly, the Seth transaction family in Sawtooth provides a valuable feature by enabling the integration of Ethereum-based applications within the network. With Seth, users gain the

ability to utilize Solidity-based smart contracts and other applications developed for the Ethereum platform directly on the Sawtooth blockchain. This interoperability expands the range of use cases and applications that can be implemented on the Sawtooth network, offering developers more options and flexibility in building decentralized applications. Additionally, leveraging Ethereum-compatible smart contracts opens up opportunities for collaboration and integration with existing Ethereum ecosystems, further enriching the Sawtooth ecosystem with a diverse range of decentralized applications and services.

4.5 Transaction Flow

Initiating a transaction requires a private key generated by the SDK's signing module, serving as proof of identity. Each transaction consists of a signature and a binary encoded payload, with the encoding defined by the transaction processor. Dependencies can be specified within transactions, dictating prerequisite operations that must be fulfilled before the current one can proceed, enabling complex transaction sequences.

Once constructed, transactions are bundled into batches and forwarded to the validator, which requests processing by the transaction processor. Upon validation, batches are deemed atomic units of change within the Sawtooth network. If any operation within a batch fails, the entire batch is rejected. The validator employs a BlockCache to manage a working set of blocks and monitor processing status, while the BlockStore retains only valid blocks, and the BlockCache handles new and invalid blocks, facilitating parallel execution.

Subsequently, transaction batches reach the Journal, accommodating pluggable consensus algorithms like Proof of Work, Proof of Elapsed Time, and Practical BFT algorithms. The Journal routes batches and blocks for ledger appending. The completer within the Journal ensures all dependencies are met before batches proceed to the BlockPublisher for validation and inclusion in a block. Upon block completion, the completer confirms fulfillment of dependencies before routing blocks to the chain controller for validation and fork resolution. Finally, validated blocks undergo ChainUpdate via the BlockStore, which maintains a chronological ledger of all blocks, tracing back to the Genesis blocks.

Chapter 5 - Comparison

In the realm of enterprise blockchain frameworks, Hyperledger Fabric, Sawtooth, and Iroha stand out as prominent solutions, each offering unique features and design philosophies tailored to diverse business requirements. A comparative analysis of these frameworks illuminates their respective strengths and trade-offs, providing valuable insights for organizations seeking to deploy blockchain solutions. Fabric, with its modular architecture and support for pluggable consensus algorithms, prioritizes flexibility and scalability, making it suitable for a wide range of enterprise applications. Conversely, Sawtooth's emphasis on modularity and ease of integration appeals to organizations seeking a customizable and adaptable blockchain platform. Meanwhile, Hyperledger Iroha's simplicity and focus on mobile application development make it particularly well-suited for identity and finance-related use cases.

Hyperledger Fabric, as a leading enterprise blockchain framework, encompasses several key components, including channels, smart contracts (chaincode), and a modular consensus mechanism. Channels enable the creation of isolated communication pathways within the network, facilitating data privacy and access control. Fabric's support for pluggable consensus algorithms, such as Raft, offers flexibility in tailoring the network's performance and fault tolerance to specific requirements. Transaction execution in Fabric follows a multi-step process, involving endorsement, ordering, and validation, ensuring consistency and immutability of ledger data across the network. Moreover, Fabric's permissioned architecture allows for granular control over access rights and governance, making it well-suited for consortiums and regulated industries.

In contrast, Hyperledger Sawtooth distinguishes itself with its modular design, which enables developers to choose consensus algorithms and transaction semantics that best suit their use cases. Sawtooth's support for parallel transaction execution enhances scalability and throughput, making it suitable for high-performance applications. The framework's smart contract engine, Sawtooth Lake, simplifies the deployment and execution of smart contracts, while its RESTful API facilitates interaction with the ledger. Additionally, Sawtooth's

compatibility with Ethereum smart contracts through the Seth integration expands its applicability to a broader developer community.

Iroha distinguishes itself with a unique architecture that emphasizes modularity and extensibility, making it a compelling choice for various enterprise blockchain applications. Operating within a permissioned network model, Iroha leverages the YAC algorithm to achieve high throughput and low latency, ensuring efficient transaction processing. This architectural approach enables seamless integration of custom smart contract logic, empowering developers to customize the platform according to specific requirements. Additionally, Hyperledger Iroha provides user-friendly client libraries and software development kits, streamlining the development process for decentralized applications and enhancing accessibility for developers. Iroha emerges as a simple and easy-to-incorporate blockchain framework, focusing on use cases in identity and finance industries. Despite their differences, all three frameworks share a common goal of enabling enterprises to leverage blockchain technology for enhanced transparency, security, and efficiency in various industries. Through a comprehensive examination of Fabric, Sawtooth, and Iroha, this chapter aims to provide valuable insights into the considerations and challenges involved in selecting the most suitable framework for enterprise blockchain deployments.

Factor per platform	**Fabric**	**Iroha**	**Sawtooth**
Differentiators	Extendable deployment architecture, channels	Universal peer role, SQL state, linearly scalable consensus	Transaction processors, pluggable components
Regional awareness	China and the rest of the world	Asia,especially Japan	USA

Permission Level	Permissioned	Permissioned	Permissioned and Permissionless
BFT Support	Only on v3.0.0-beta March 14, 2024	+	+
Transaction Processing	Endorsing Peers and Ordering Services	Universal role peers	Validators + Parallel transactions
Consensus Algorithm	Raft	YAC	PoET
EVM Support	No	No	Yes
Smart Contract Technology	Chaincode	Commands and queries	Transaction Families

Table 3 - Platform Comparison

5.1 Private and Public Network

Sawtooth stands out with its provision of two blockchain variants, allowing users to seamlessly switch between permissioned and permissionless access based on their specific needs. This dual offering provides unparalleled flexibility, enabling businesses to tailor their blockchain solutions to diverse scenarios. In contrast, Iroha is primarily a permissioned ledger platform, emphasizing secure and controlled access for enterprise applications. Fabric, on the other hand, is known for its privacy features, making it ideal for private networks where access rights are paramount. Fabric's architecture facilitates swift transactions and data partitioning, ensuring efficient performance even with a reduced number of network nodes.

5.2 Privacy and IAM

All three frameworks approach privacy, identity, and access management with distinct methodologies tailored to the diverse needs of enterprise applications. Fabric, known for its sophisticated privacy features, offers a fine-grained access control model that enables organizations to define granular permissions for network participants through the use of channels. Fabric channels serve as isolated communication pathways within the network, allowing organizations to segregate transactions and data into distinct groups. Each channel operates with its own set of participants, ensuring that transactions are visible only to the parties involved. Fabric leverages identity management through Membership Service Providers within each channel, allowing each member to have a unique digital identity and ensuring that transactions are executed only by authorized parties. Additionally, Fabric integrates private data collections, enabling selective disclosure of sensitive information to specific network participants while maintaining data confidentiality across the network.

In contrast, Hyperledger Iroha adopts a permissioned network model with a focus on simplicity and ease of integration. While Iroha provides identity management capabilities through digital signatures and cryptographic keys, its primary emphasis is on streamlined access control mechanisms suitable for straightforward use cases. Iroha's permissioned ledger architecture restricts access to predefined network participants, ensuring that only authorized entities can transact on the network. However, Iroha's approach to privacy and identity is less nuanced compared to Fabric, making it more suitable for applications requiring basic access controls and uncomplicated identity management. Sawtooth, as mentioned before, stands out with its modular design and support for both permissioned and permissionless blockchain variants. Sawtooth offers flexibility in identity management through its pluggable consensus algorithms, enabling organizations to choose the authentication mechanisms that best align with their security requirements. Additionally, Sawtooth's support for multiple permissioning mechanisms allows for fine-tuned access control policies, facilitating secure interaction among network participants. However, Sawtooth's approach to privacy and identity management may require more customization compared to Fabric's comprehensive privacy features, making it better suited for applications where adaptability and extensibility are paramount.

5.3 Consensus

Fabric offers a pluggable consensus model, allowing organizations to choose the most suitable algorithm for their network requirements. Among the supported algorithms is Raft, a leader-based protocol designed for fault tolerance and quick leader election. Raft is considered a CFT consensus algorithm. It is designed to handle crash faults, where nodes fail by stopping to respond or crash but do not exhibit Byzantine behavior. Raft focuses on ensuring the availability and consistency of the system in the presence of these types of failures, making it suitable for scenarios where Byzantine faults are less of a concern. However, it does not provide Byzantine fault tolerance and may not be suitable for environments where Byzantine faults are more likely to occur. Fabric also supports Practical Byzantine Fault Tolerance, a consensus algorithm known for its resilience against Byzantine faults, making it ideal for permissioned networks where trust among participants is essential.

Hyperledger Iroha implementation of the YAC algorithm, is based on voting for block hash. YAC focuses on simplicity and efficiency, making it suitable for applications requiring high throughput and low latency. While it may not offer the same level of fault tolerance as PBFT, YAC provides a balance between performance and reliability. Sawtooth, on the other hand, offers a modular consensus architecture that supports multiple algorithms, including Practical Byzantine Fault Tolerance and Crash Fault Tolerance. PBFT ensures Byzantine fault tolerance by requiring a two-thirds majority agreement among nodes, while CFT ensures network resilience by tolerating node failures without compromising data consistency.

Each of these consensus mechanisms has its strengths and weaknesses, making them suitable for different use cases. Fabric's pluggable consensus model offers flexibility and customization options, while Iroha's YAC algorithm prioritizes simplicity and efficiency. Sawtooth's modular architecture allows for experimentation with various consensus algorithms, making it suitable for diverse network requirements. Ultimately, the choice of consensus mechanism depends on factors such as network scalability, security requirements, and performance considerations.

5.4 Smart Contracts

Smart contracts play a pivotal role in blockchain networks, automating the execution of predefined agreements and transactions without the need for intermediaries. In Hyperledger Fabric, smart contracts, known as chaincodes, are written in familiar programming languages such as Go, JavaScript, or Java. Fabric supports both stateless and stateful smart contracts, allowing developers to define business logic that interacts with the ledger's state. Chaincodes in Fabric are executed within a secure and isolated environment called the "execution environment," ensuring the integrity and confidentiality of transactions. Fabric's modular architecture enables the deployment of multiple chaincodes on different channels, facilitating the segregation of business logic and data across distinct organizational entities.

Iroha simplifies smart contract development with its domain-driven programming model and support for a wide range of programming languages, including C++, JavaScript, and Java. Iroha's smart contracts, referred to as commands and queries, focus on modularity and extensibility, enabling developers to create custom logic tailored to specific use cases. Unlike Fabric and Sawtooth, Iroha's smart contracts are fixed commands rather than programmable scripts, offering a simplified approach to decentralized application development. However, this fixed design provides a level of predictability and security, making it suitable for applications requiring high throughput and low latency, such as identity management and financial services.

Hyperledger Sawtooth employs a flexible approach to smart contracts, supporting both Ethereum-compatible smart contracts through the Seth integration and custom transaction processors for more specialized use cases. Additionally, Sawtooth utilizes transaction families, which are akin to smart contracts in other blockchain platforms, allowing developers to define custom transaction types and associated business logic. These transaction families encapsulate specific sets of operations or domain-specific functionality within the blockchain network. Sawtooth's smart contracts and transaction families are written in languages like Python or Rust and run within transaction processors, providing developers with the flexibility to implement complex business logic tailored to their unique use cases.

5.5 Transaction Execution

Another critical aspect for comparison between the three frameworks is transaction execution efficiency and how each of them approaches the optimization of performance while maintaining integrity and security. In Fabric, transactions are executed within chaincodes, ensuring security and flexibility in state transitions. However, the endorsement policy mechanism, where transactions must be endorsed by a predefined number of peers, can introduce overhead and potentially impact efficiency, particularly in networks with stringent endorsement requirements. On the other hand, Hyperledger Iroha prioritizes efficiency with its streamlined transaction execution model. Atomic commands executed directly by nodes, eliminating the need for complex smart contracts. This approach results in high throughput and low latency, making Iroha ideal for applications requiring rapid transaction processing and scalability. By simplifying transaction execution, Iroha minimizes computational overhead and resource consumption, enhancing overall network efficiency.

Similarly, Hyperledger Sawtooth emphasizes efficiency through its modular architecture and flexible transaction processing models. Sawtooth allows developers to choose between parallel and sequential transaction execution, enabling optimization for specific use cases. Additionally, Sawtooth's support for pluggable consensus algorithms enhances efficiency by providing options to tailor consensus mechanisms to the network's requirements. By offering a balance between flexibility and performance, Sawtooth ensures efficient transaction execution while accommodating diverse application needs.

Overall, Fabric, Iroha, and Sawtooth demonstrate different approaches to transaction execution efficiency, each suited to specific use cases and requirements. While Fabric emphasizes security and flexibility, Iroha prioritizes simplicity and performance, and Sawtooth offers a versatile solution with customizable transaction processing models. By optimizing transaction execution efficiency, these frameworks enable the development of scalable and robust blockchain applications across various industries and domains.

5.6 Conclusion

In conclusion, Hyperledger Fabric, Iroha, and Sawtooth represent three distinct blockchain frameworks within the Hyperledger ecosystem, each offering unique features and capabilities tailored to different use cases and requirements. Fabric stands out for its sophisticated privacy features, fine-grained access control, and support for complex multi-party workflows. Iroha emphasizes simplicity, modularity, and high throughput, making it well-suited for applications in finance, identity management, and mobile development. Sawtooth distinguishes itself with its flexible architecture, support for Ethereum-compatible smart contracts, and modular consensus algorithms, providing developers with the flexibility to build custom blockchain solutions that meet their specific needs.

Despite their differences, all three frameworks share common characteristics such as robust security, scalability, and support for smart contracts. They enable developers to build decentralized applications with tailored business logic while ensuring the integrity and security of transactions on the blockchain. Moreover, their modular architectures and support for pluggable components empower developers to customize and extend the frameworks to address diverse use cases and industry requirements. As blockchain technology continues to evolve Fabric, Iroha, and Sawtooth will play crucial roles in driving innovation and adoption across various industries. Whether it's implementing supply chain solutions, improving identity management, or enhancing financial services, these frameworks provide the foundation for building scalable, secure, and interoperable blockchain networks that empower organizations to transform their business processes and drive value in the digital economy.

Chapter 6 - Deployment and benchmarks

In this chapter, a performance benchmarking exercise is conducted to compare two blockchain networks utilizing different consensus mechanisms. The first network is based on Hyperledger Fabric using the Raft consensus algorithm, while the second network is built on Hyperledger Besu, an EVM-compatible platform, employing the Quorum Byzantine Fault Tolerance (QBFT) consensus mechanism.The primary objective is to deploy a Fabric test network and a Besu test network and deploy identical chaincode and smart contracts across both networks for a fair comparison. Leveraging Hyperledger Caliper, a benchmarking tool, performance evaluations will be conducted under these networks. Subsequently, a second Fabric test network will be deployed based on Fabric's 3.0 beta version. Fabric v3.0 is the first release(March 2024) to provide a BFT ordering service based on the SmartBFT consensus library. The benchmarks are re-run to juxtapose the results and ascertain the comparative efficacy of the two consensus approaches. Through this iterative process, the aim is to glean insights into the performance disparities and nuances between Raft and BFT in Fabric, elucidating their implications for blockchain network deployments. The metrics on the benchmarks include Send Rate—the rate at which Caliper issued the transactions, Latency (max/min/avg)—statistics relating to the time taken in seconds between issuing a transaction and receiving a response, and Throughput—the average number of transactions processed per second.

To provide a comprehensive comparison, the following table outlines the basic characteristics of each network selected for this benchmarking analysis. These networks have been chosen due to their distinctive consensus mechanisms and architectural designs, which are pivotal in assessing their performance across various transaction workloads:

Network	Consensus Mechanism	Deployment model	Key Characteristics

Hyperledger Fabric(Raft)	Raft	Docker container	Simplified single-node Raft consensus, no TLS CA, certificates from root CAs, container isolation
Hyperledger Fabric(BFT)	Practical Byzantine Fault Tolerance (BFT)	Docker container	Enhanced fault tolerance, BFT consensus for Byzantine fault resilience, similar architecture to Raft
Hyperledger Besu	IBFT	Docker container	Ethereum client, IBFT for high fault tolerance, supports public and private networks

Table 4 - Selected networks key characteristics

Reasons for Selection:

1. Hyperledger Fabric with Raft: Chosen for its straightforward and efficient consensus mechanism suitable for typical enterprise blockchain applications. Its single-node setup makes it an ideal candidate for testing and educational purposes.
2. Hyperledger Fabric with BFT: Included to evaluate the performance of a Byzantine Fault Tolerant consensus in Fabric. BFT provides higher fault tolerance, making it relevant for more secure and resilient applications.
3. Hyperledger Besu: Selected to compare an Ethereum-based blockchain with IBFT consensus, highlighting differences in transaction handling and network performance relative to Fabric's solutions. Besu’s compatibility with both public and private networks offers insights into versatile blockchain deployment scenarios.

These networks provide a balanced spectrum of blockchain solutions, each with unique features and consensus mechanisms that make them suitable for different use cases. The forthcoming sections will delve into the detailed performance analysis of these networks,

including both query and transaction tests. The results are presented in the following tables, providing a comparative evaluation of their capabilities.

6.1 Caliper

Hyperledger Caliper[39], originating in 2017, has evolved into a useful tool for assessing blockchain platform performance. Within the Hyperledger project, it serves as an open-source solution tailored for standardized performance evaluations across diverse blockchain systems. Its modular architecture allows for easy extension and customization, adapting to evolving requirements. Addressing the critical need for objective performance evaluation in blockchain, Caliper provides standardized benchmarks and metrics. It empowers application developers and system designers by enabling evaluation of transaction success rates, throughput, latency, and resource utilization. Hyperledger Caliper's significance lies in filling the gap for a universally accepted tool to compare blockchain platform performances. By furnishing diverse performance metrics, it aids clients in selecting the most suitable blockchain implementation for their specific needs.

Hyperledger Caliper supports various blockchain platforms, including Hyperledger Fabric, Sawtooth, and Ethereum, ensuring comprehensive performance evaluations. It offers versatile benchmark tests, encompassing smart contract execution, consensus mechanisms, and transaction processing, facilitating thorough comparisons among different blockchain platforms. Compatible with multiple programming languages such as JavaScript, TypeScript, and Python, Caliper ensures ease of use and integration within diverse development environments. The tool generates detailed performance reports, complete with comprehensive metrics and graphical representations, facilitating clear interpretation and analysis of benchmark results. Seamless integration into existing blockchain development workflows ensures minimal disruption and efficient incorporation into established processes.

6.2 Fabric test network

The provided test network within the fabric-samples[40] repository serves as an educational tool for developers to gain familiarity with Hyperledger Fabric. Designed for local machine deployment, it facilitates experimentation with smart contracts and applications. However, it's essential to recognize that this network isn't intended as a blueprint for production setups. Modifying the provided scripts is discouraged, as it could lead to network instability. The network architecture comprises two peer organizations and an ordering organization. It simplifies the ordering service by employing a single node Raft consensus mechanism. Notably, a TLS Certificate Authority (CA) isn't deployed, and all certificates originate from root CAs for streamlined configuration. Deployed via Docker Compose, the network isolates nodes within a container network, limiting direct connections to external Fabric nodes. This approach reduces complexity but isn't suitable for inter-network communication.

In a Fabric network, each participant, whether a node or user, must belong to an organization to engage with the network. The test network in question comprises two peer organizations, Org1 and Org2, along with a single orderer organization responsible for managing the network's ordering service. Peers serve as the backbone of Fabric networks, responsible for storing the blockchain ledger and validating transactions before they are committed. These peers execute smart contracts containing the logic governing asset management within the ledger. In the test network setup, each organization operates a single peer, labeled as peer0.org1.example.com and peer0.org2.example.com, respectively.

Within a Fabric network, an ordering service plays a pivotal role in establishing transaction order, essential for maintaining consistency across distributed peers. While peers validate and add transactions to the ledger, they do not determine the transaction order. This responsibility falls on the ordering service, ensuring agreement on transaction sequencing among geographically dispersed peers. The ordering service streamlines transaction validation and commitment for peers. Upon receiving endorsed transactions, ordering nodes reach consensus on transaction order and package them into blocks. These blocks are then disseminated to peer nodes, where they are appended to the blockchain ledger.

In the test network, a single node Raft ordering service is utilized, operated by the orderer organization under the identifier orderer.example.com. However, production networks typically feature multiple ordering nodes managed by one or more organizations. These nodes collaborate using the Raft consensus algorithm to finalize transaction order across the network, ensuring robustness and fault tolerance. With the peer and orderer nodes operational, we're ready to create a Fabric channel to facilitate transactions between Org1 and Org2. Channels serve as private communication layers exclusively accessible to designated network participants, remaining concealed from other network members. Each channel operates with its distinct blockchain ledger.

To initiate the channel creation process, organizations extend invitations to their respective peers to join the channel. Invited peers then "join" the channel, enabling them to store the channel ledger and validate transactions occurring within it. Creating a Fabric channel involves several steps. First, each organization extends invitations to its peers to join the channel. Invited peers accept these invitations, gaining access to the channel's ledger. Configuration parameters, such as policies and access controls, are established for the channel. Peers validate transactions within the channel based on the defined policies. Lastly, participating peers store and update the channel's ledger to reflect validated transactions.By establishing this dedicated communication channel, Org1 and Org2 can securely transact with each other, leveraging blockchain technology's immutability and transparency.

6.3 Besu test network

For the EVM network, the chosen solution is the quorum-dev-quickstart[41] Besu network. Hyperledger Besu is an Ethereum client tailored to cater to enterprise needs across both public and private permissioned network scenarios, boasting an adaptable EVM implementation. Its versatility extends to compatibility with test networks like Sepolia and Görli. Noteworthy is Besu's inclusion of diverse consensus algorithms, encompassing Proof of Stake, Proof of Work, and Proof of Authority (such as IBFT 2.0, QBFT, and Clique). Moreover, its intricate permissioning schemes are meticulously crafted to suit consortium environments, ensuring robust governance and access control within such settings.This quickstart package offers developers a streamlined approach to setting up a local development environment

tailored specifically for Besu blockchain development. Leveraging Docker containers, the quickstart provides a pre-configured Besu network with essential components such as transaction nodes, block-making nodes, and consensus mechanisms.

The quorum-dev-quickstart simplifies the setup process, enabling developers to focus on building and testing their applications rather than dealing with the intricacies of network configuration. Alongside the core Besu network, the package includes a suite of development tools like Truffle and Remix, facilitating smart contract development and application testing. These tools enhance the development workflow, allowing for rapid iteration and refinement of decentralized applications.

In the Besu network provided, the architecture is structured around several key components, each playing a vital role in the operation and consensus of the network. Each quickstart setup consists of 4 validators and one RPC node.These nodes communicate with each other via the RPC interface, facilitating the exchange of data and coordination of network activities.

One of the defining features of the Besu network is its consensus mechanism, which governs how transactions are validated and added to the blockchain. The Besu based Quorum variant uses the Istanbul BFT 2.0 consensus mechanism,a proof of authority (PoA) consensus protocol. In IBFT 2.0 networks, approved accounts, known as validators, validate transactions and blocks. Validators take turns to create the next block. Before inserting the block onto the chain, a super-majority (greater than or equal to 2/3) of validators must first sign the block.

6.4 Benchmark configuration

To run a benchmark with Hyperledger Caliper, one needs to establish the foundation of the blockchain network by setting up the network configuration files, smart contracts, and the benchmark configuration file. The process begins with crafting the network configuration files, which serve as blueprints defining the topology of the blockchain network. These files typically outline the structure of the network, including details such as the number of nodes, organizations, consensus algorithms, and any other relevant parameters. For instance, in the

case of Hyperledger Fabric, a `test-network.yaml` file is created to specify the peers, orderers, channels, and other essential network details. If working with other blockchain platforms like Sawtooth or Ethereum, corresponding network configuration files tailored to their specific requirements must be crafted. Figures a and b show the configurations used to conduct this benchmark.

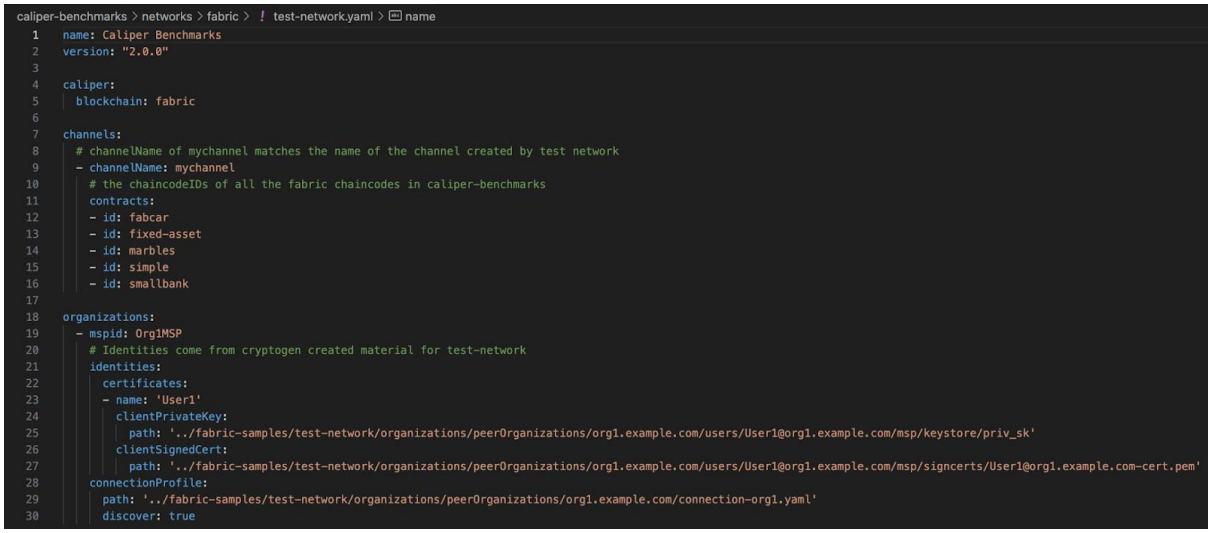

Figure 6.1 - Test-network.yml for fabric

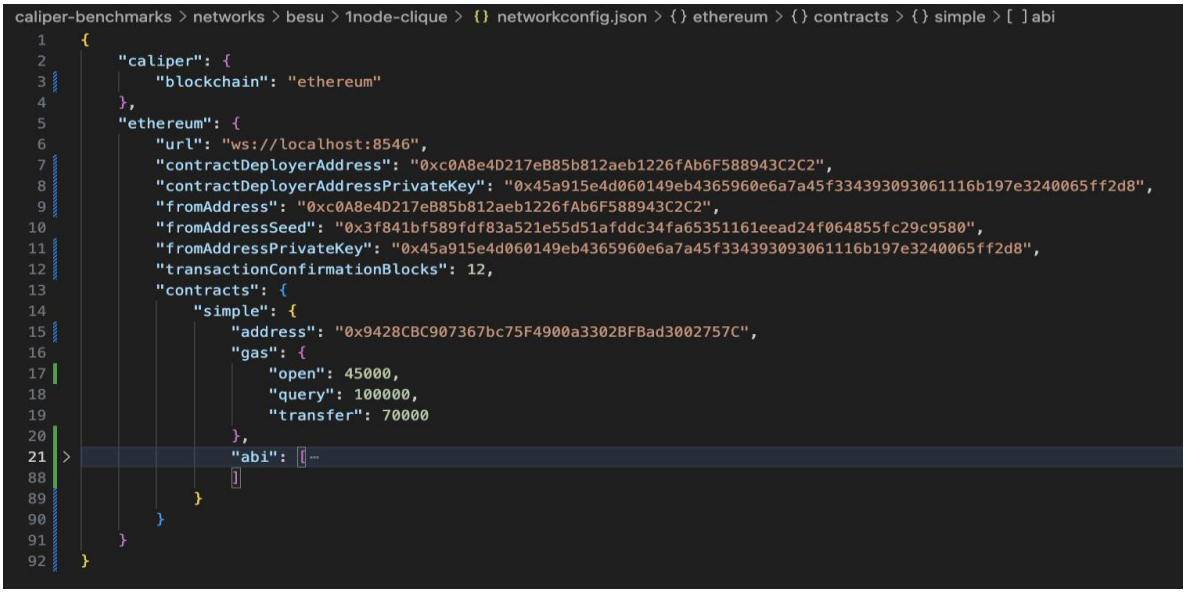

Figure 6.2 - Network-config.json for Besu

Next, the user must develop or obtain the smart contracts that will be deployed and executed on the blockchain network. Smart contracts contain the business logic governing transactions and interactions within the network. For Hyperledger Fabric, these contracts are typically written in Go, JavaScript, or Java, while for Ethereum-based networks, they're coded in Solidity. It is important to ensure that the smart contracts align with the chosen programming language and version of the blockchain platform. Solidity contracts should be compiled into bytecode for deployment on the Ethereum network, while Fabric smart contracts need to be packaged for installation on the Fabric network.Simple.sol and Simple.go shown in Figures 7.3 and 7.4 were used on this benchmark. Both implementations serve similar purposes of facilitating basic financial transactions on a blockchain network. They allow for account opening, balance querying, and fund transfers between accounts. While one is implemented using Hyperledger Fabric's chaincode in Go, the other is a smart contract written in Solidity for execution on the Ethereum Virtual Machine (EVM). Despite the differences in the underlying blockchain platforms and programming languages, the core functionalities provided by both implementations remain consistent.

```
pragma solidity >=0.4.22 <0.6.0;

contract simple {
    mapping(string => int) private accounts;

    function open(string memory acc_id, int amount) public {
        accounts[acc_id] = amount;
    }

    function query(string memory acc_id) public view returns (int amount) {
        amount = accounts[acc_id];
    }

    function transfer(string memory acc_from, string memory acc_to, int amount) public {
        accounts[acc_from] -= amount;
        accounts[acc_to] += amount;
    }
}
```

Figure 6.3 - Simple.sol

Once the network and smart contracts are in place, the user can proceed to craft the benchmark configuration file, which serves as the blueprint for the benchmarking process. This file outlines various parameters such as the workload to be executed, the number of transactions to be generated, the rate control mechanism, and any additional arguments required by the workload modules or rate control mechanism. The user should specify the

workload modules for each benchmarking round, configure the rate control mechanism to determine the transaction rate, and define any other parameters necessary for the benchmarking process. Figure 7.5 shows the configuration used on this test. This configuration file defines multiple test rounds, each with a specific workload to be executed against the deployed smart contract. The simpleArgs section defines arguments used by the workloads, such as initial account balances and the number of accounts. The test section specifies the name and description of the benchmark, as well as the number of workers to be used. Each round within the rounds section describes a specific test scenario, including the number of transactions, the transaction rate, and the workload module to be executed. Lastly, the monitors section configures resource monitoring for Docker containers running the network components, including peers and orderers, to capture resource utilization metrics during the benchmark execution.

```go
// query current money of the account,should be [query accout]
func (t *SimpleChaincode) Query(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 1 {
        return shim.Error(ERROR_WRONG_FORMAT)
    }

    money, err := stub.GetState(args[0])
    if err != nil {
        s := fmt.Sprintf(ERROR_SYSTEM, err.Error())
        return shim.Error(s)
    }

    if money == nil {
        return shim.Error(ERROR_ACCOUNT_ABNORMAL)
    }

    return shim.Success(money)
}

// transfer money from account1 to account2, should be [transfer account1 account2 money]
func (t *SimpleChaincode) Transfer(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 3 {
        return shim.Error(ERROR_WRONG_FORMAT)
    }
    money, err := strconv.Atoi(args[2])
    if err != nil {
        return shim.Error(ERROR_WRONG_FORMAT)
    }

    moneyBytes1, err1 := stub.GetState(args[0])
    if err1 != nil {
        s := fmt.Sprintf(ERROR_SYSTEM, err1.Error())
        return shim.Error(s)
    }
    moneyBytes2, err2 := stub.GetState(args[1])
    if err2 != nil {
        s := fmt.Sprintf(ERROR_SYSTEM, err2.Error())
        return shim.Error(s)
    }
    if moneyBytes1 == nil || moneyBytes2 == nil {
        return shim.Error(ERROR_ACCOUNT_ABNORMAL)
    }

    money1, _ := strconv.Atoi(string(moneyBytes1))
    money2, _ := strconv.Atoi(string(moneyBytes2))
    if money1 < money {
        return shim.Error(ERROR_MONEY_NOT_ENOUGH)
    }

    money1 -= money
    money2 += money

    err = stub.PutState(args[0], []byte(strconv.Itoa(money1)))
    if err != nil {
        s := fmt.Sprintf(ERROR_SYSTEM, err.Error())
        return shim.Error(s)
    }

    err = stub.PutState(args[1], []byte(strconv.Itoa(money2)))
    if err != nil {
        s := fmt.Sprintf(ERROR_SYSTEM, err.Error())
        return shim.Error(s)
    }
```

Figure 6.4 - Query and transfer functions on simple.go chaincode

In the "Query Round - 5000 transactions at 100 TPS," the goal is to assess query performance by executing 5000 transactions at a fixed rate of 100 TPS. This round utilizes the query.js workload module to simulate queries against the smart contract, enabling analysis of response time and throughput. Similarly, the "Query Round - 5000 transactions at 200 TPS" evaluates query performance but at a higher transaction rate of 200 TPS. This round employs the same query.js workload module to execute queries, facilitating a comparison of performance metrics across different transaction rates.

Shifting focus to transfer functionality, the "Transfer Round - 1000 transactions at 5 TPS" tests transfers between accounts at a fixed rate of 5 TPS. Using the transfer.js workload module, this round aims to assess transaction throughput and latency under a lower TPS scenario.The subsequent "Transfer Round - 1000 transactions at 10 TPS" and "Transfer Round - 1000 transactions at 20 TPS" increase the transaction rates to 10 TPS and 20 TPS, respectively. These rounds aim to evaluate scalability and performance under higher transaction rates, utilizing the same transfer.js workload module for comparison.

Overall, these rounds offer a comprehensive evaluation of the smart contract's performance across different transaction types and rates. The comparative analysis provides insights into scalability, efficiency, and responsiveness under varying workloads, facilitating informed decision-making for optimizing the smart contract's performance.

```yaml
test:
  name: simple
  description: >-
    This is an example benchmark for Caliper, to test the backend DLT's
    performance with simple account opening & querying transactions.
  workers:
    number: 1
  rounds:
    - label: query-5000tx-100tps
      description: query performance of the deployed contract.
      txNumber: 5000
      rateControl:
        type: fixed-rate
        opts:
          tps: 100
      workload:
        module: benchmarks/scenario/simple/query.js
        arguments: *simple-args
    - label: query-5000tx-200tps
      description: query performance of the deployed contract.
      txNumber: 5000
      rateControl:
        type: fixed-rate
        opts:
          tps: 200
      workload:
        module: benchmarks/scenario/simple/query.js
        arguments: *simple-args
    - label: transfer-1000tx-5tps
      description: transfering money between accounts.
      txNumber: 1000
      rateControl:
        type: fixed-rate
        opts:
          tps: 5
      workload:
        module: benchmarks/scenario/simple/transfer.js
        arguments:
          << : *simple-args
          money: 100
    - label: transfer-1000tx-10tps
      description: transfering money between accounts.
      txNumber: 1000
      rateControl:
        type: fixed-rate
        opts:
          tps: 10
      workload:
        module: benchmarks/scenario/simple/transfer.js
        arguments:
          << : *simple-args
          money: 100
    - label: transfer-1000tx-20tps
      description: transfering money between accounts.
      txNumber: 1000
      rateControl:
        type: fixed-rate
        opts:
          tps: 20
      workload:
        module: benchmarks/scenario/simple/transfer.js
        arguments:
          << : *simple-args
          money: 100
```

Figure 6.5 - Config.yaml defines workloads and benchmark rounds

Finally, the benchmark can be executed using the Caliper CLI. The user should utilize the `npx caliper launch manager` command, providing the path to the benchmark configuration file and the path to the network configuration as arguments. Throughout the benchmarking process, performance metrics such as transaction throughput, latency, and resource utilization should be monitored. Once the benchmark concludes, Caliper will generate a

comprehensive report containing detailed performance statistics and analysis, enabling the user to evaluate the efficacy of the blockchain network under various conditions.

6.4.1 Fabric network with BFT

In March 2024, Hyperledger Fabric introduced the beta version 3.0, which includes the Byzantine Fault Tolerance consensus mechanism as a notable feature. This update enables users to retest the network using the BFT consensus algorithm alongside the existing Raft consensus mechanism. However, it's important to note that this BFT functionality is currently available only in the beta version 3.0 of Hyperledger Fabric. Conducting benchmark tests with both consensus mechanisms offers valuable insights into their respective performance characteristics and can inform decision-making regarding network deployment and optimization strategies.

6.5 Comparison and takeaways

By focusing on key performance metrics such as transaction throughput, latency, and success rates, the evaluation seeks to provide comprehensive insights into the networks' ability to handle both read and write-intensive workloads. By analyzing those metrics, this thesis will try to shed light on the comparative strengths and weaknesses of each network's query and transaction functionality, offering valuable insights for blockchain developers and network architects. The results of the benchmark analysis are presented in the following tables.

Round	Success	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
query-5000tx-100tps	5000	0	100	0.04	0	0	100
query-5000tx-200tps	5000	0	200.1	0.02	0	0	200
transfer-1000tx-5tps	1000	0	5	5.09	0.18	2.61	4.9
transfer-1000tx-10tp	1000	0	10	5.21	0.17	2.67	9.8
transfer-1000tx-20tp	1000	0	20	5.32	0.13	2.73	19.3

Table 5 - Test results for Besu

Round	Success	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
query-5000tx-100tps	5000	0	100	0.04	0	0.01	100
query-5000tx-200tps	5000	0	200	0.03	0	0.01	200
transfer-1000tx-5tps	985	15	5	2.07	0.06	0.99	5
transfer-1000tx-10tp	984	16	10	1	0.06	0.53	10
transfer-1000tx-20tp	979	21	20	2.09	0.05	0.32	19.3

Table 6 - Test results for Fabric using Raft

Round	Success	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
query-5000tx-100tps	5000	0	100	0.03	0	0.01	100
query-5000tx-200tps	5000	0	200.1	0.03	0	0.01	200
transfer-1000tx-5tps	976	24	5	2.14	0.04	1.62	5
transfer-1000tx-10tp	928	72	10	2.14	0.03	1.52	9.8
transfer-1000tx-20tp	900	100	20	2.13	0.04	1.56	19.2

Table 7 - Test results for Fabric using BFT

The query tests reveal intriguing similarities in performance among the three networks, suggesting a level of parity in their query handling capabilities. Across different transaction rates, all three networks consistently maintain high success rates, indicating robustness in query execution. Similarly, latency metrics exhibit comparable trends, with minor fluctuations observed between the networks. Hyperledger Besu, Fabric with BFT consensus, and Fabric with Raft consensus all demonstrate commendable response times for query transactions, underscoring their efficiency in processing read requests. These findings suggest a degree of uniformity in query performance across the evaluated blockchain networks, hinting at the resilience and reliability of their query functionalities.

Moreover, the consistency in performance metrics across the three networks implies a level playing field in terms of query processing efficiency. While each network may employ distinct consensus mechanisms and architectural designs, the query tests indicate that they all excel in providing timely and accurate responses to query transactions. This uniformity in performance underscores the maturity and robustness of blockchain technology, wherein different networks can achieve comparable levels of efficiency in handling read-intensive workloads. As such, developers and enterprises evaluating blockchain solutions for query-intensive applications can draw confidence from the consistent performance exhibited by

Hyperledger Besu, Fabric with BFT consensus, and Fabric with Raft consensus in this benchmark analysis.

Upon closer examination of the transfer transactions, it becomes evident that both Hyperledger Fabric with Raft and BFT consensus encountered varying degrees of failures, particularly attributed to MVCC errors during transaction execution. These errors resulted in a higher number of failures compared to Hyperledger Besu, which experienced no such issues. MVCC errors signify challenges in managing concurrent transactions and accessing shared data, potentially leading to inconsistencies in transaction outcomes and network performance.

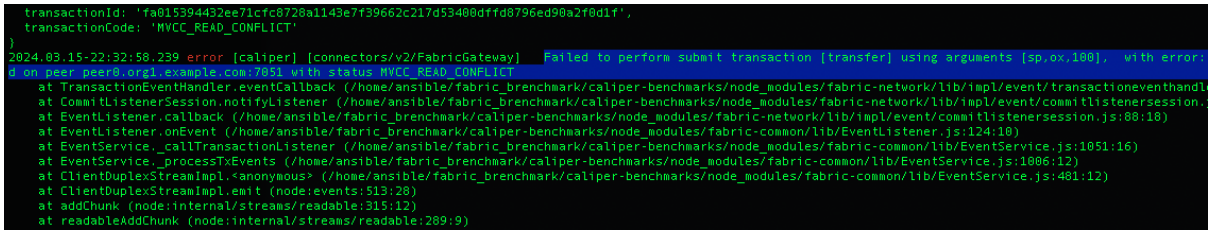

Figure 6.6 - MVCC Error while executing transaction in Fabric

While both Fabric networks experienced these errors, there are notable differences in their performance metrics. Hyperledger Fabric with Raft demonstrated faster transaction processing speeds and lower latencies compared to its BFT counterpart. Despite experiencing MVCC errors, Fabric with Raft maintained a higher success rate and exhibited superior performance in handling transfer transactions, showcasing its resilience and efficiency. Conversely, Fabric with BFT consensus exhibited slightly higher latencies and lower throughputs, indicating potential scalability challenges under similar transaction loads.This comparison underscores the strengths and weaknesses of each consensus mechanism. Raft's simplicity and fast leader election process contribute to its robustness and efficiency in managing transaction concurrency and ensuring data consistency. On the other hand, BFT consensus, while offering strong fault tolerance and Byzantine fault resistance, may introduce overhead and complexity that impact performance, especially in scenarios with high transaction rates.Hyperledger Besu demonstrates robustness in transaction handling with

consistently high success rates and minimal errors, it is worth noting that it exhibits lower transaction throughput compared to Hyperledger Fabric. Despite its reliability, Besu's slower transaction processing speed may pose challenges for applications requiring high transaction throughput or low-latency transaction execution.

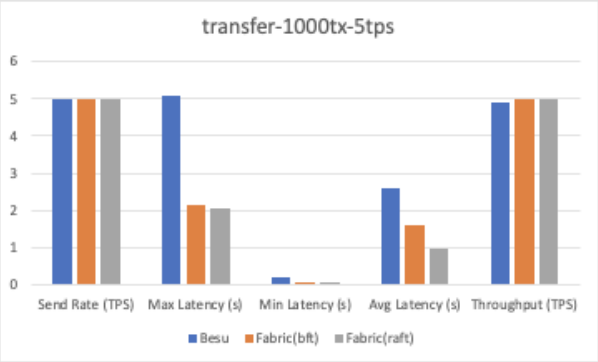

Figure 6.7 - Round 3 metrics

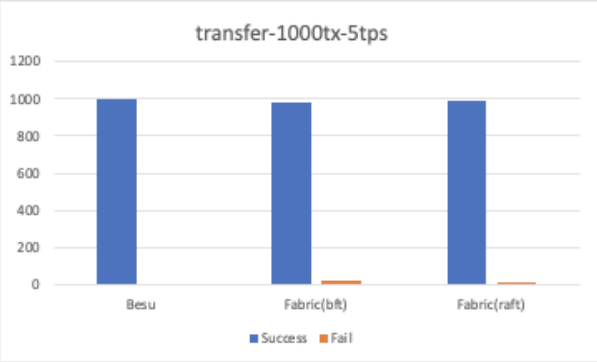

Figure 6.8 - Round 3 Success Rate

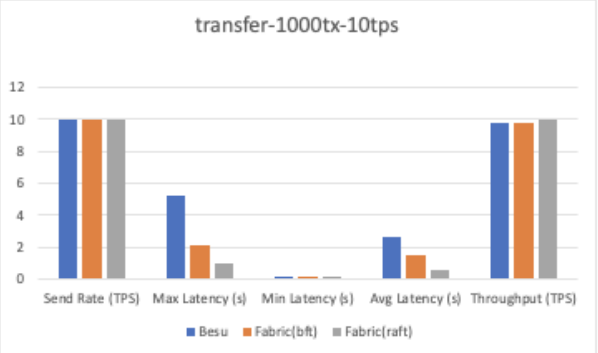

Figure 6.9 - Round 4 metrics

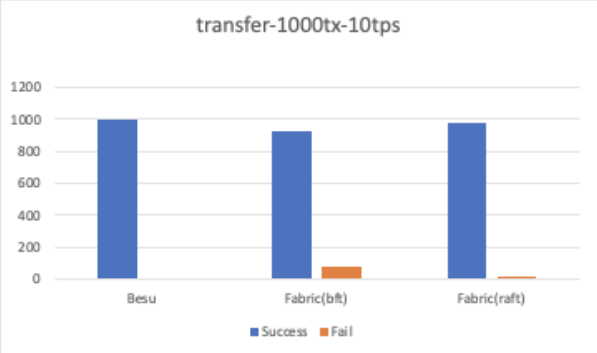

Figure 6.10 - Round 4 Success Rate

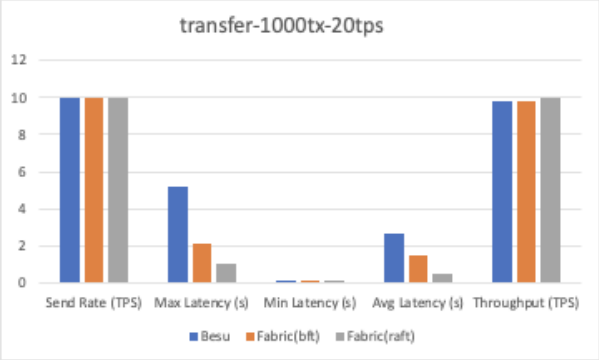

Figure 6.11 - Round 5 metrics

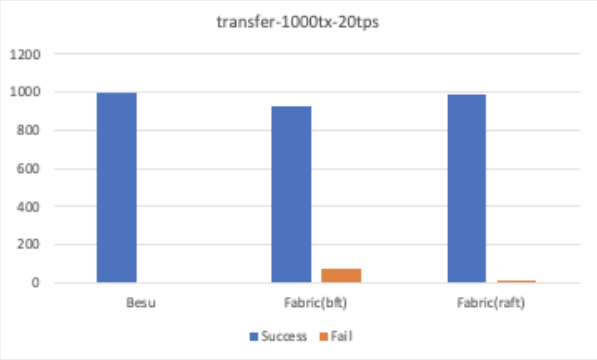

Figure 6.12 - Round 5 Success Rate

Overall, this comparative analysis highlights the nuanced trade-offs between consensus mechanisms and their impact on network performance. Raft's simplicity and efficiency make it well-suited for transaction-heavy applications where fast processing and low latencies are paramount. In contrast, BFT consensus, while offering strong fault tolerance, may introduce complexities that affect performance, especially under high transaction loads. Hyperledger Besu, while reliable, may face challenges in handling high transaction throughput due to its slower processing speed. Ultimately, blockchain developers and enterprises must carefully evaluate their performance requirements and consider the strengths and weaknesses of each network to make informed decisions when selecting a blockchain solution for their specific use case. Moving forward, continued research, experimentation, and optimization efforts will be essential to unlock the full potential of these blockchain frameworks and enable their seamless integration into diverse enterprise ecosystems.

References

[1] Hyperledger: Open-Source Blockchain Framework and Standards
- https://www.investopedia.com/terms/h/hyperledger.asp
[2] https://www.hyperledger.org/about
[3] https://www.hyperledger.org/projects
[4] https://www.hyperledger.org/members
[5] https://www.eweek.com/cloud/hyperledger-blockchain-project-is-not-about-bitcoin/
[6] FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second
- https://ieeexplore.ieee.org/document/8751452
[7] What are the different types of Hyperledger Fabric nodes? -
https://www.linkedin.com/pulse/what-different-types-hyperledger-fabric-nodes-charge-maria-lopez
[8]Channel capabilities - https://hyperledger-
fabric.readthedocs.io/en/latest/capabilities_concept.html
[9] https://tsmatz.wordpress.com/2020/02/17/hyperledger-fabric-tutorial-on-azure/
[10] Introducing Chaincode: Hyperledger Fabric Smart Contracts -
https://www.kaleido.io/blockchain-blog/introducing-chaincode-what-you-need-to-know
[11] https://hyperledger-fabric.readthedocs.io/en/release-2.5/smartcontract/smartcontract.html
[12] Membership Service Provider (MSP) - https://hyperledger-
fabric.readthedocs.io/en/latest/membership/membership.html
[13] The Architecture of Hyperledger Fabric: An In-Depth Guide -
https://www.spydra.app/blog/architecture-of-hyperledger-fabric-an-in-depth-guide
[14] Ledger - https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger.html
[15]Exploring Consensus Mechanisms in Hyperledger Fabric: Raft, Solo, and Kafka -
https://www.linkedin.com/pulse/exploring-consensus-mechanisms-hyperledger-fabric-raft-solo-
kafka
[16] Understanding Hyperledger Fabric — Endorsing Transactions -
https://medium.com/kokster/hyperledger-fabric-endorsing-transactions-3c1b7251a709
[17] Read-Write set semantics - https://hyperledger-fabric.readthedocs.io/en/release-
2.2/readwrite.html
[18] Ordering service implementations¶ -https://hyperledger-fabric.readthedocs.io/en/release-
2.2/orderer/ordering_service.html
[19] https://etcd.io/
[20] Raft Understandable Distributed Consensus - https://thesecretlivesofdata.com/raft/

[21] Hyperledger Fabric — The Taste of Raft - https://medium.com/coinmonks/hyperledger-fabric-the-taste-of-raft-4f9f0df20b
[22] Raft Understandable Distributed Consensus -https://thesecretlivesofdata.com/raft/
[23]How do transactions work in Hyperledger Fabric - https://www.educative.io/answers/how-do-transactions-work-in-hyperledger-fabric
[24] Transaction flow - https://hyperledger-fabric.readthedocs.io/en/release-2.5/txflow.html
[25] Strong Consistency vs Eventual Consistency - https://medium.com/@abhirup.acharya009/strong-consistency-vs-eventual-consistency-19ce6f87c112
[26] Iroha - https://www.hyperledger.org/projects/iroha
[27] Hyperledger Iroha: What It Is, How It Works - https://www.investopedia.com/terms/h/hyperledger-iroha.asp
[28] YAC: BFT Consensus Algorithm for Blockchain - https://arxiv.org/pdf/1809.00554.pdf
[29] https://wiki.hyperledger.org/download/attachments/6423987/HL-HK-Bootcamp-Iroha-basic-slides.pdf?api=v2&modificationDate=1551945944000&version=1
[30] Introduction to Hyperledger Iroha (Alexandra & Arianna Groetsema) - https://github.com/hyperledger-archives/education/blob/master/LFS171x/docs/introduction-to-hyperledger-iroha.md
[31] What's inside Iroha? - https://iroha.readthedocs.io/en/main/concepts_architecture/architecture.html
[32] Hyperledger Sawtooth: A gentle introduction - https://blog.logrocket.com/hyperledger-sawtooth-introduction/
[33] What is Proof of Elapsed Time (PoET)? Unlocking the Secrets of Proof of Elapsed Time (PoET) - https://medium.com/unicorn-ultra/what-is-proof-of-elapsed-time-poet-unlocking-the-secrets-of-proof-of-elapsed-time-poet-0bb7dd1b614e
[34] What Is Hyperledger Sawtooth And Its Exclusive Benefits? - https://blockchainshiksha.com/what-is-hyperledger-sawtooth/
[35] Hyperledger Sawtooth Overview: Looking at Permissioned Networks from a Different Perspective -https://openledger.info/insights/hyperledger-sawtooth-overview/
[36] Hyperledger Sawtooth - Seth - https://github.com/hyperledger-archives/sawtooth-seth
[37]Understanding Hyperledger Sawtooth — Proof of Elapsed Time - https://medium.com/kokster/understanding-hyperledger-sawtooth-proof-of-elapsed-time-e0c303577ec1

[38]https://www.cryptopolitan.com/how-to-explore-the-capabilities-of-hyperledger-fabric-and-sawtooth-blockchains/

[39] https://hyperledger.github.io/caliper/

[40] https://github.com/hyperledger/fabric-samples

[41] https://github.com/Consensys/quorum-dev-quickstart