universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Benchmarking consensus techniques in permissioned blockchain systems

verfasst von | submitted by

Yernar Kumashev B.Sc.

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von | Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas, for his clear guidance, continuous support, and remarkable patience throughout the course of this thesis. His expertise and insightful feedback were instrumental in shaping both the direction and the quality of this work. I am especially thankful for the autonomy he granted me, which allowed me to develop a deeper sense of ownership and responsibility for my research.

I would also like to extend my heartfelt thanks to my colleagues, whose support, encouragement, and constructive discussions greatly enriched my academic journey. Their companionship helped me navigate challenges more confidently and made the experience all the more rewarding.

Furthermore, I am deeply grateful to my family for their unwavering belief in me and for providing the emotional foundation that carried me through every phase of this endeavor. Without their constant support, this achievement would not have been possible.

To everyone who played a role in this journey—whether through mentorship, collaboration, or moral support—I offer my sincerest appreciation.

Abstract

Permissioned blockchain systems have gained significant traction among corporations due to features such as access control, scalability, performance, and privacy—features that are notably absent in the most widely adopted permissionless blockchain, Bitcoin. These attributes are particularly critical in industries such as healthcare, finance, supply chain, and others. However, there are many different types of permissioned blockchain systems, each employing its own consensus mechanism. This poses a challenge in selecting an appropriate system, as no universal solution currently meets all requirements.

This thesis introduces a novel framework capable of creating networks across three supported permissioned blockchain systems: Ethereum Clique, Quorum, and Tendermint. Following network stabilisation, tests are conducted to capture key metrics, including Throughput, Latency, CPU usage, and Memory consumption. These metrics are then visualised in graphical form for analysis.

The first section of the thesis provides a comprehensive review of permissioned blockchain systems. The second section outlines the design and implementation of the framework, along with detailed steps to extend its functionality to support additional blockchain systems. Finally, the third section offers an in-depth analysis of the captured metrics, evaluating the strengths and weaknesses of the three blockchain systems supported.

Kurzfassung

Erlaubnisbasierte Blockchain-Systeme haben insbesondere aufgrund ihrer Eigenschaften wie Zugriffskontrolle, Skalierbarkeit, Leistung und Datenschutz erheblich an Bedeutung für Unternehmen gewonnen – Merkmale, die in der am weitesten verbreiteten erlaubnisfreien Blockchain, Bitcoin, weitgehend fehlen. Diese Eigenschaften sind insbesondere in Sektoren wie dem Gesundheitswesen, dem Finanzwesen, der Lieferkette und weiteren Branchen von entscheidender Bedeutung.

Gleichzeitig existiert eine Vielzahl unterschiedlicher erlaubnisbasierter Blockchain-Systeme, von denen jedes ein eigenes Konsensverfahren implementiert. Diese Diversität stellt eine Herausforderung bei der Auswahl eines geeigneten Systems dar, da derzeit keine universelle Lösung existiert, die sämtliche Anforderungen erfüllt.

Im Rahmen dieser Arbeit wird ein neuartiges Framework vorgestellt, das die Erstellung von Netzwerken auf drei unterstützten erlaubnisbasierten Blockchain-Systemen ermöglicht: Ethereum Clique, Quorum und Tendermint. Nach der Stabilisierung des Netzwerks werden Tests durchgeführt, um zentrale Leistungskennzahlen zu erfassen, darunter Durchsatz, Latenz, CPU-Auslastung und Speicherverbrauch. Diese Kennzahlen werden anschließend grafisch aufbereitet und analysiert.

Der erste Teil der Arbeit bietet einen umfassenden Überblick über erlaubnisbasierte Blockchain-Systeme. Im zweiten Teil werden das Design und die Implementierung des Frameworks erläutert sowie die erforderlichen Schritte zur Erweiterung der Funktionalität auf zusätzliche Blockchain-Systeme detailliert beschrieben. Abschließend folgt im dritten Teil eine eingehende Analyse der erfassten Leistungskennzahlen, wobei die Stärken und Schwächen der drei unterstützten Blockchain-Systeme kritisch bewertet werden.

Contents

List of Tables

List of Figures

Listings

1. Introduction

Blockchain technology first gained widespread recognition with the emergence of Bitcoin, introduced in Satoshi Nakamoto's 2008 whitepaper [Nak08]. This work proposed a tamper-proof ledger secured by cryptography and consensus mechanisms, solving the double-spending problem through a chain of proof-of-work blocks. In 2015, Ethereum extended these ideas by enabling smart contracts and decentralised applications on the Internet [B+13].

As blockchain technology grew in popularity, industry leaders and researchers recognised its potential beyond cryptocurrency. Today, blockchain adoption spans diverse sectors, including healthcare [AEVL16] and supply chain management [KHD17], where it enhances transparency and traceability. The technology's appeal lies in its core features: immutability, transparency, and decentralisation.

However, public blockchains like Bitcoin and Ethereum do not fully align with corporate requirements for privacy and regulatory compliance. This gap led to the development of new blockchain types—namely, permissioned (private) and consortium systems. Blockchains are broadly categorised into three types: permissionless (public), permissioned (private), and consortium. Their key distinction lies in governance, particularly in determining which entities can control and participate in the network.

Permissionless blockchains operate without a central authority, allowing anyone to join and contribute to consensus (e.g., Bitcoin, Ethereum). In contrast, permissioned blockchains restrict access to authorised users, with a single organisation or group controlling read and write permissions. Consortium blockchains represent a middle ground, operating as semi-decentralised systems typically managed by a consortium of entities rather than a single party.

Governance in permissioned and consortium blockchains is more centralised, controlled by known stakeholders, unlike the fully decentralised governance of public blockchains. This difference introduces design trade-offs, shaping their typical use cases:

- **Permissionless blockchains** prioritise decentralisation, relying on open consensus protocols like Proof-of-Work or Proof-of-Stake. However, this results in lower efficiency. For instance, Bitcoin processes only ∼7 transactions per second, with confirmation latencies of ∼10 minutes [CDE+16]. In contrast, global payment technology company Visa achieves finality in seconds and handles up to 56,000 transactions per second [CDE+16].

- **Permissioned blockchains** prioritise authorised participation, privacy, and performance. They employ faster consensus protocols, enabling higher throughput and quicker finality, making them suitable for enterprise contexts (e.g., banking transactions or hospitals sharing patient records).

Since 2016, major permissioned blockchain projects have emerged. J.P. Morgan introduced Quorum, a private Ethereum-based ledger for financial markets [Cha16], while the Linux Foundation launched Hyperledger Fabric, the first open-source, extensible blockchain for distributed applications [ABB+18a]. These projects retain blockchain's core benefits while incorporating critical business features, including access control, transaction and data privacy, and improved performance.

Real-world applications demonstrate blockchain's impact. For example, pharmaceutical supply chains now utilise blockchain technology for the traceability and authenticity validation of drugs, combating counterfeits [She22]. In healthcare, blockchain-based solutions have been explored to enhance data security to combat the rising number of cyberattacks and data breaches. Permissioned blockchains offer the potential to replace traditional centralised databases or manual record-keeping systems that require trust in a central authority.

These successes have driven further adoption and development, resulting in the creation of numerous consensus mechanisms utilised by permissioned blockchains. This Master's Thesis paper aims to identify the most widely used consensus mechanisms and build a benchmarking tool capable of evaluating their performance across multiple blockchain systems, providing clear guidelines on how to update existing versions of blockchains and add additional blockchains to the system.

1.1. Motivation

Permissioned blockchains are widely adopted, yet research in this domain remains limited. For enterprises, selecting an appropriate permissioned blockchain platform is a critical decision, yet the lack of reliable benchmarks complicates this process. The primary challenge stems from the extensive variety of available platforms. For instance, a healthcare consortium implementing a shared patient record system must evaluate options such as Hyperledger Fabric, Quorum, or Tendermint/Cosmos-based solutions. While each platform claims theoretical advantages, empirical performance evaluations are necessary to validate these assertions.

A benchmarking tool is needed to provide decision-makers with actionable data. Current benchmarking frameworks—such as BlockBench [DWC+17], BCTMark [SLM20], and BlockCompas [RHF23]—lack the flexibility or documentation to support newer blockchain versions, rendering them obsolete over time. A review of related work underscores this gap: no existing benchmarking tool can comprehensively evaluate a diverse and growing set of blockchain systems.

This Master's Thesis paper addresses this gap by developing an extensible benchmarking tool capable of conducting thorough performance evaluations of popular blockchain systems. The tool is designed for adaptability, enabling users to integrate and assess additional systems with minimal effort. By providing empirical comparisons, this work highlights the strengths and weaknesses of popular blockchain systems, answering important questions such as "Which blockchain system delivers the lowest latency in a 32-node network?" and "What is the CPU overhead of Tendermint compared to Quorum?"

Several factors drive the need for such a tool:

- **No one-size-fits-all solution:** No single platform excels in all scenarios, requiring organisations to identify the optimal solution for their specific needs. The proposed tool clarifies the trade-offs between Ethereum Clique, Quorum, and Tendermint, with extensibility enabling comparisons with other systems (as detailed in Chapter 6).
- **Rapid innovation in blockchain technology:** New consensus mechanisms and frameworks emerge frequently, necessitating a benchmarking tool that can adapt swiftly. This motivates the focus on extensibility, ensuring minimal effort is required to integrate additional blockchain systems.
- **Performance bottlenecks in production environments:** Enterprises must assess whether a blockchain system can handle heavy workloads. By measuring Throughput, the tool validates the suitability of these systems for large-scale, real-world applications.
- **Resource efficiency and operational costs:** Infrastructure requirements significantly influence platform selection. For instance, a 10% Throughput gain at double the CPU cost may not justify the trade-off. Metrics such as CPU and memory consumption highlight resource efficiency, directly impacting operational costs and sustainability.

1.2. Objectives

To address the identified needs, this Master's Thesis paper establishes the following objectives:

- **Comprehensive analysis of permissioned blockchains:** Conduct a systematic review of permissioned blockchain systems, with particular emphasis on consensus mechanisms and their impact on security, scalability, and decentralisation. The analysis will examine key use cases across different industries.
- **Identification of key performance metrics:** Establish and justify the most critical metrics for blockchain performance evaluation, including Throughput, Latency, CPU/Memory utilisation, and Scalability. The relevance of each metric to real-world operational requirements will be explicitly demonstrated.
- **Evaluation of existing benchmarking tools:** Conduct a critical survey of current blockchain benchmarking frameworks, analysing their methodologies and limitations. This assessment will identify gaps in existing solutions and justify the need for a new approach.
- **Development of a novel benchmarking framework:** Design and implement an extensible benchmarking framework that supports out-of-the-box evaluation of three major blockchain systems (Ethereum Clique, Quorum, and Tendermint).

The framework's architecture will emphasise modularity and extensibility, enabling straightforward integration of additional blockchain systems.

- **Simulation of realistic workloads:** Implement workload patterns that accurately reflect production environments, including sustained operation over extended periods to evaluate system stability and performance degradation.
- **Comparative performance analysis:** Conduct rigorous empirical evaluation of the selected blockchain systems under controlled conditions. The study will correlate performance characteristics with underlying consensus mechanisms, providing actionable insights for system selection.

This research is motivated by real challenges faced in the enterprise adoption of blockchain technology. The resulting framework and findings will provide practical guidance for selecting appropriate permissioned blockchain systems. By developing a rigorous yet adaptable performance evaluation methodology, this work contributes to both academic research and industrial practice, facilitating more informed decisions regarding the adoption of new technologies.

The Master's Thesis paper is organised as follows: Chapter 2 presents comprehensive background research on permissioned blockchains and benchmarking methodologies. Chapter 3 presents comparative studies in permissioned blockchain systems and reviews existing frameworks developed for benchmarking such systems. Chapter 4 details the framework's architecture and implementation. Chapter 5 analyses experimental results and performance metrics. Finally, Chapter 6 provides complete guidelines for extending the framework with additional blockchain systems.

1.3. Research Questions

This Master's Thesis paper addresses the following key research questions, with answers provided in the indicated chapters:

- **RQ1:** What are the interesting criteria for benchmarking various consensus techniques for a given set of blockchain systems? (Chapter 2)
- **RQ2:** What are the advantages or disadvantages of each consensus technique in all the systems (of our selection) concerning each criterion? (Chapter 2)
- **RQ3:** How can the framework be easily extended to support additional blockchain systems? (Chapter 6)

2. Background

2.1. Blockchain technology

A blockchain is a type of distributed ledger that enables peer-to-peer transactions without requiring a central authority. In a blockchain, data is stored in a sequence of blocks that are cryptographically linked (each block contains a hash of the previous block), forming an immutable chain. This design makes it computationally infeasible to tamper with records, since altering a block would break the link to all following blocks and be immediately evident to the network participants [NBF+16]. The concept was first introduced with Bitcoin in 2008 by Satoshi Nakamoto [Nak08], and has since evolved far beyond cryptocurrencies.

Key properties of blockchain systems include:

- **Decentralisation:** Control of the ledger is distributed among network participants rather than controlled by a single authority. This means no single entity can maliciously alter the ledger, increasing robustness against single points of failure.
- **Transparency:** In blockchain systems, all transactions can be verified by any authorised participant. Every node can hold a copy of the ledger, enabling easy audit.
- **Integrity:** Consensus protocols and cryptographic algorithms guarantee the immutability of executed transactions. Once a transaction is confirmed and added to a block, it becomes challenging to change or remove it without detection, ensuring strong data integrity.
- **Availability:** The distributed nature of the network ensures continuous operation. Even if some nodes go offline or malfunction, the network as a whole remains available and can continue processing transactions.

The blockchain network is maintained by miners or validators who are responsible for validating transactions and producing new blocks. The method by which these network participants reach agreement on the next block is known as the consensus mechanism. Consensus varies between blockchains and is a central aspect of blockchain design. The first-generation consensus mechanism was Proof-of-Work (PoW), as used by Bitcoin, which involves all miners competing to solve a cryptographic puzzle; the first to solve it appends the next block and receives a reward. PoW is effective at securing open networks, but it is highly energy-consuming and relatively slow, thus often criticised for its inefficiency and environmental impact [OM14]. These limitations have prompted

the development of many other consensus mechanisms that aim to be more efficient. For example, Proof-of-Stake (PoS) is now used by Ethereum and other blockchains to eliminate the energy cost of PoW by requiring validators to put cryptocurrency at stake instead of performing wasteful computations [GHW24].

Depending on the consensus mechanism and governance model, blockchain systems can be divided into three types:

- **Permissionless Blockchains:** Open networks that anyone can join and participate in without prior approval. These systems maximise decentralisation and trustlessness, often at the cost of performance. Bitcoin and the original Ethereum are prime examples, both employing resource-intensive consensus (Bitcoin uses Proof-of-Work [Nak08], Ethereum used Proof-of-Work until switching to Proof-of-Stake [GHW24]). Since anyone can participate in the network, a robust defence system powered by mining rewards is maintained to keep the network secure, which leads to lower throughput and higher latency.

- **Permissioned Blockchains:** Private networks controlled by a single entity or a closed group, where access is restricted to authorised participants. As the network operates in a secure and trustworthy environment, these systems can utilise a more efficient consensus mechanism that doesn't require high computational power. This results in significantly higher throughput and faster transaction finality. For example, the permissioned blockchain framework Hyperledger Fabric can achieve end-to-end Throughput of 3,500 transactions per second [ABB+18b]. Because participants are known, these systems can enforce privacy policies, making them an attractive option for enterprises, where data must remain confidential.

- **Consortium Blockchains:** Networks where control is distributed among a group of pre-approved organisations rather than a single entity. It combines traits of both permissionless and private systems: the network is not open to the public, but it is not under the sole control of one party, which provides a degree of decentralisation. Consortium blockchains are commonly used in industries where different organisations need to collaborate and share information without making it public. For instance, an international bank might form a consortium blockchain for inter-bank payments. This ensures shared trust and accountability among the stakeholders while maintaining strong privacy.

2.2. Permissioned Blockchains

Permissioned blockchains have gained popularity in enterprise settings by addressing some limitations of public blockchains, like a lack of privacy and low performance. Meanwhile, they are still maintaining blockchain's core benefits such as immutability, distributed consensus and traceability. Enterprises that want to utilise blockchain's features, but require keeping records private and access-controlled. They achieve this with permissioned networks where only verified identities can join. By restricting participation, permissioned

blockchains enhance privacy, enable compliance with regulations, and can hold participants accountable for their actions on the network.

Some key features of permissioned blockchains include:

- **Access control:** Only authorised participants are allowed to join and transact on the network. This controlled membership mitigates many security issues and ensures that sensitive information is only shared among trusted parties.
- **Scalability:** Because permissioned systems usually employ more lightweight consensus algorithms, they tend to scale better in terms of transaction throughput. With fewer nodes and a more efficient consensus, the network can handle a higher load.
- **Performance:** Permissioned blockchains prioritise speed and throughput. Transactions can reach finality much faster, as consensus can be reached through quick voting, for example, rather than the time-consuming process of mining.
- **Privacy and Confidentiality:** Permissioned networks can implement strict privacy controls. This is often a deciding factor for adopting a permissioned blockchain in industries like healthcare or finance. This feature is crucial in contexts where business or personal data is involved.

Overall, permissioned blockchains sacrifice open decentralisation in favour of controlled memberships and performance efficiency. Many modern enterprise blockchain platforms such as Hyperledger Fabric [ABB+18b], R3 Corda [HB16], Quorum [Cha16], and Hyperledger Sawtooth [Sawst] are permissioned or consortium in nature, offering various features for identity management, consensus, and smart contracts to suit business needs.

2.3. Overview of consensus mechanism in permissioned blockchains

A consensus mechanism is the protocol that network participants use to agree on the contents of the next block in the blockchain. In permissionless blockchains like Bitcoin, consensus mechanisms such as PoW (Proof-of-Work) are necessary to enable anonymous, trustless participants to reach a consensus. In permissioned blockchains, since participants are verified, we often use faster and less resource-intensive consensus algorithms. This Master's Thesis focuses only on permissioned blockchains.

Below is an overview of several popular consensus mechanisms relevant to permissioned blockchains:

2.3.1. Proof of Authority (PoA)

Proof of Authority [Cli17] is a consensus mechanism that relies on a small number of trusted validator nodes (authorities) who are granted the right to create new blocks.

Unlike PoW, there is no competition among miners; instead, validator nodes take turns proposing or validating blocks in a round-robin or scheduled manner. A block is considered valid when a majority (or a predetermined quorum) of the authorised validators sign off on it. Because the validators are known and reputable (their identities are often public or at least known to network members), PoA can achieve consensus with very low overhead, resulting in a significantly more efficient validation process compared to PoW.

Strengths: PoA is very fast and offers high throughput. Since no complex computation is required, block times can be short (e.g., 1–5 seconds), and the transaction confirmation is almost immediate after a validator creates a block and others approve it. The algorithm is straightforward to implement, and resource consumption is low. PoA works well in private networks where all validators are trusted to some degree by the network operator.

Trade-offs: The main concern with PoA is its impact on decentralisation. Because only a limited set of validators control the ledger, the system essentially trusts those authorities. If a majority of them misbehave, they could violate the ledger's history. Thus, PoA sacrifices decentralisation and it's only suitable when validators can be held accountable. In summary, PoA offers the typical benefits of blockchain, such as immutability, redundancy, and transparency among participants, but places trust in a few known entities to maintain those benefits.

Use cases: PoA is used in several private Ethereum variants and test networks. For example, Ethereum's Clique consensus algorithm is a PoA mechanism [Cli17]. Another PoA implementation, called Aura, is used by Parity Ethereum and substrates [Fou20]. These demonstrate PoA's practicality for networks where nodes are permissioned and efficiency is a priority.

2.3.2. Proof of Elapsed Time (PoET)

Proof of Elapsed Time (PoET) [Cor16] is an energy-efficient consensus algorithm introduced by Intel as part of the Hyperledger Sawtooth project [Sawst]. PoET uses secure hardware to achieve a fair leader election without heavy computation. The core idea is that each validator requests a wait time from a trusted hardware module (an Intel SGX enclave) and goes to sleep for that randomised interval. The node whose timer expires first wakes up and becomes the leader, eligible to produce the next block. It then broadcasts a proof of its wait time to the network. Other nodes can verify this proof using the same secure hardware technology to ensure that the leader indeed waited for the allotted time and was not just self-selecting.

Strengths: PoET is designed to be as fair and random as PoW in choosing a leader, but without requiring any work, meaning no energy wasted on mining. It leverages hardware-based trust (SGX) to enforce the random wait. This results in consensus efficiency similar to PoA/PoS — low latency and high Throughput — but with a larger validator set, as anyone with the required hardware can participate (in a permissioned

context). It's quite scalable and suitable for consortium environments where participants have compatible hardware.

Trade-offs: PoET's reliance on Intel SGX means trust is somewhat shifted to hardware and Intel's implementation of secure enclaves. If the hardware's security is compromised or faulty, the consensus can be misused. Also, PoET requires all validators to have the specific hardware, which may not always be feasible. It's not a fully decentralised solution, as it assumes a trusted execution environment; however, in permissioned settings, this is often an acceptable trade-off for efficiency.

Use cases: PoET was implemented in Hyperledger Sawtooth [Sawst], which demonstrated its viability. Sawtooth's PoET can achieve transaction finality in the order of seconds and scale to dozens or more validators doing minimal work beyond maintaining the SGX environment.

2.3.3. Practical Byzantine Fault Tolerance (PBFT)

Practical Byzantine Fault Tolerance is a consensus algorithm (proposed by Castro and Liskov in 1999 [CL+99]) that guarantees consensus even in the presence of malicious or "Byzantine" nodes, as long as a majority fraction (more than two-thirds) of nodes are honest. PBFT works through a series of voting rounds. In simplified terms, one node acts as a primary (leader) to propose a block, and all nodes exchange several rounds of messages: pre-prepare, prepare, and commit. During these phases, nodes cast votes on the proposal, eventually arriving at a final commit if a majority ($\geq 2/3$ of nodes) agrees. The result is that all honest nodes will have reached the same decision on the block, which is then appended to the chain; the process repeats for the next block, with a new leader often determined through a round-robin protocol.

Strengths: PBFT and its variants provide firm consistency guarantees and fast finality. Once a block is committed in PBFT, it is final, and no forks will occur. The safety holds as long as less than 1/3 of the nodes are faulty, which is a robust assumption for many consortium scenarios. PBFT's Latency is typically low in a local network, often resulting in transaction finalisation in maybe a few hundred milliseconds to a second, depending on network speed [CL+99]. It is well-suited for small networks where security against certain potential internal threats is required. Many modern permissioned blockchain consensus protocols are based on PBFT or incorporate its principles. For example, Hyperledger Fabric has an optional BFT ordering service based on PBFT for high-security environments [ABB+18a], and Quorum's Istanbul BFT [Cha16] is effectively a PBFT-style algorithm adapted for Ethereum transactions.

Trade-offs: The main drawback of PBFT is scalability. Its communication overhead is on the order of $O(n^2)$, as every node needs to communicate with every other node in each consensus round for voting. This means that as the number of validator nodes (n) grows, the number of commit messages grows quadratically, which can dramatically slow down

the consensus. In practice, pure PBFT is considered practical for smaller networks, as its performance degrades significantly beyond that point. Additionally, any network delays will slow the latency. Finally, the protocol is complex to implement correctly [CKR12], given the numerous corner cases that arise when handling faulty nodes; it requires careful engineering to avoid safety or liveness bugs.

Use cases: PBFT has influenced many enterprise blockchain designs, and solutions often combine PBFT with other consensus mechanisms. Zilliqa [Sec18] and other blockchain platforms use PBFT, combined with different techniques, to finalise blocks. Generally, whenever a permissioned blockchain network is small and needs tolerance of minority malicious insider nodes, a PBFT-based consensus is a strong candidate.

2.3.4. Raft

Raft is a well-known crash fault-tolerant consensus algorithm from the distributed systems field (developed by Ongaro and Ousterhout, 2014) [OO14], designed to be easy to understand and implement. Raft assumes no Byzantine behaviour, meaning nodes may fail or disconnect, but will not deliberately lie. It focuses on replicating a log of transactions across nodes. In Raft-based consensus, one node is the leader at any given time, and it is responsible for selecting transactions and appending them to the distributed log chain. The leader sends the new entry to follower nodes, which then acknowledge it. Once a majority of followers have accepted the entry, it is considered committed, and the leader can officially add it to the ledger. Raft includes an automatic leader election process. If a leader becomes unavailable or unresponsive, the other nodes hold a vote to elect a new leader.

Strengths: Raft is admired for its simplicity and strong consistency under the assumption of non-Byzantine failures. It provides immediate finality. Latency is low because there is only one round-trip from the leader to the followers, and only a majority ($\geq 1/2$ of the nodes) need to respond. Throughput can also be high, limited mostly by the leader's performance capacity and the size of the data. Because it doesn't involve complex cryptographic computations or multiple phases, CPU and network overhead in Raft is minimal. Raft also has the advantage of being a mature algorithm with many real-world implementations.

Trade-offs: The obvious limitation is that Raft does not tolerate Byzantine faults. If a malicious actor somehow becomes the leader, Raft has no mechanism to detect dishonest behaviour. In a permissioned blockchain scenario, this means Raft is best suited when there is a high level of trust among nodes. Additionally, Raft-based blockchains typically do not rotate leadership very frequently under normal conditions. A stable leader can remain in place for a long time until a fault occurs. This leads to concerns about decentralisation, though the leader is still constrained by needing followers' acknowledgements.

Use cases: Quorum (J.P. Morgan's Ethereum-based ledger) offered a Raft consensus

mode for single/consortium organisations, for fast, crash-tolerant consensus suitable for scenarios that require high throughput [Cha16]. Quorum's Raft mode gives rapid block propagation times, fast transaction finality and high Throughput, making it comparable to a traditional database in performance for moderate node counts. R3 Corda [1] uses a Raft-based cluster in its standard configuration to achieve consensus on transaction uniqueness among a set of known authorities. These examples demonstrate that Raft is a popular choice when an easy and fast consensus mechanism is sufficient and the network participants are presumed to be honest.

2.3.5. Tendermint

Tendermint is a Byzantine Fault Tolerant consensus algorithm that has gained popularity through its use in the Cosmos network [KB19]. Its design is often compared to PBFT but with some valuable simplifications and optimisations tailored for blockchain settings [Kwo14]. Tendermint consensus operates in rounds where a validator is selected as a leader in a rotating fashion, weighted by stake, in the Cosmos network, but for permissioned scenarios, it could be a simple round-robin among authorised nodes. The proposer suggests the next block, and then validators go through two voting phases, called pre-vote and pre-commit, similar to PBFT. If a supermajority ($\geq 2/3$) of validators pre-commit a block, it will be finalised. If a problem occurs and not enough votes were collected, the protocol moves to the next round with a new leader.

Strengths: Tendermint provides fast finality and high throughput. It has a simple and efficient commit logic that requires only two stages of votes per block. In practice, Tendermint consensus can achieve block times of around 1 second or a few seconds, with thousands of transactions per block. The Throughput can be in the hundreds or thousands of transactions per second on a well-connected network of dozens of validators. Tendermint is known for its efficiency and consistency. Once Tendermint commits the block, it will never be reverted. Tendermint is also relatively easier to integrate as a module, due to its well-developed Tendermint Core engine, which can be integrated into various application layers.

Trade-offs: Like any other BFT consensus, Tendermint's performance will degrade if the number of validators grows very large, impacting throughput and latency significantly. Also, Tendermint usually relies on a stable network with minimal latency. If the network is too unstable, Tendermint's performance might significantly slow down until connectivity is restored. Still, among BFT protocols, Tendermint is regarded as one of the more scalable and production-ready for blockchain needs.

Use cases: Cosmos Hub and many blockchain zones in the Cosmos ecosystem [KB19] use Tendermint as their consensus engine, illustrating its capability in a decentralised context. In private settings, Tendermint can be utilised whenever a consortium requires

[1] https://developer.r3.com/

robust Byzantine fault tolerance. Companies have explored Tendermint for financial solutions, supply chain networks, and other use cases that require fast settlement and security against a minority of corrupt insider nodes.

2.3.6. HotStuff BFT (as used in Libra/Diem)

HotStuff is a newer Byzantine fault-tolerant consensus algorithm that gained popularity for its clean, modular design and was adopted by Facebook's Libra (Diem) blockchain project [Tas19]. The HotStuff BFT protocol, introduced by Yin et al. (2019) [YMR+18], simplifies BFT consensus by having a unified three-phase commit for every proposed block and chaining these commits to pipeline consensus decisions. In HotStuff BFT, a leader proposes a block, and the validators undergo three rounds of voting (prepare, pre-commit, and commit). If the process completes successfully with a sufficient number of votes, that block becomes committed. Importantly, when one block is committed, the next block can start the phases concurrently, which improves throughput. New blocks can be proposed without waiting for the previous block to reach finality.

Strengths: HotStuff's main advantages are simplicity and adaptability. The algorithm is leader-based. It has a linear communication complexity after the leader's proposal. Followers send votes to the leader, who then sends combined certificates back, avoiding the PBFT's all-to-all communication in each phase. This means HotStuff scales better in terms of throughput. LibraBFT [BCC+19], which is built based on HotStuff BFT, demonstrated around one thousand transactions per second in testing [ZGW+19], and demonstrated high scalability scores.

Trade-offs: HotStuff still inherits the logic of BFT consensus (requires $\geq 3f + 1$ validators to tolerate f faulty nodes), so the integrity is similar to PBFT/Tendermint. While it improves communication complexity, it doesn't eliminate the fact that every validator must still process every block and vote, so a large number of validators will introduce a downgrade in latency. In leader-based BFT protocols, performance can degrade if the leader is slow; however, HotStuff mitigates this issue by enabling leader rotation. Another challenge is that HotStuff is relatively new and has not yet gained widespread adoption globally. In short, HotStuff is a modified and enhanced version of BFT consensus, and similarly, it must balance the same trade-offs as any distributed consensus.

Use cases: The most popular use of HotStuff BFT was in the Diem (Libra) blockchain project, which implemented a variant called LibraBFT. It was planned to use in a permissioned payment network [BCC+19]. Even though Diem was never launched publicly, the consensus design was made public and stands as a benchmark for modern BFT protocols. Despite this, HotStuff can be used by banks or large-scale consortia to manage digital currency platforms where members are known, but strong security is required.

2.3.7. Other Notable Consensus Mechanisms

In addition to the above, there are several other consensus algorithms and variants that have been proposed or used in permissioned settings:

- **Istanbul BFT (IBFT):** Originating from the Quorum platform, IBFT is a Byzantine Fault Tolerant, similar to PBFT. In IBFT, a known set of validators take turns proposing blocks and using a voting protocol to finalise them. IBFT is specifically adapted to block creation in an Ethereum context. In practice, IBFT provides immediate finality and good performance for Ethereum consortium networks, albeit at the cost of the usual BFT limit, which scales beyond dozens of validators. Both Quorum [Cha16] and Hyperledger Besu [FLKM22] support IBFT for private networks.

- **HoneyBadger BFT:** HoneyBadgerBFT [MXC+16] is an asynchronous BFT protocol. HoneyBadger can tolerate arbitrary network delays and still make progress. It achieves this by using atomic broadcast through threshold encryption. Transactions are encrypted and spread to nodes, then collectively decrypted only when enough shares are received. The protocol can reach high throughput, but the trade-off is higher latency and complexity in implementation. HoneyBadger's fully asynchronous nature makes it resilient, but slow to reach transaction finality. It's the most suitable in a setting where network instability often occurs.

- **Others:** There are many niche emerging algorithms. For instance, Redundant Byzantine Fault Tolerance (RBFT) [AMQ13] is a variant aiming to improve performance by running multiple parallel instances of BFT and replacing the primary one if it becomes slow. Algorand's [CGT19] consensus (permissionless) introduces cryptographic sortition to randomly select small consensus committees from a large pool each round – a concept that could inspire permissioned systems wanting to involve many nodes but not have all vote on everything. New research continues to introduce improvements targeting better speed, security, or scalability.

Challenges and Trade-offs: It's important to note that no single consensus mechanism is optimal for all scenarios. PoW-based systems maximise decentralisation and security but suffer from throughput and latency. BFT-style systems (such as PBFT, Tendermint, and HotStuff) provide strong consistency and security guarantees, but typically require a relatively small, fixed validator set to perform optimally. PoA and Raft sacrifice decentralisation to achieve the highest performance. This variety of consensus mechanisms in permissioned blockchains motivates a thorough performance evaluation. Understanding how each mechanism affects throughput, latency, resource usage, and scalability helps determine which blockchain system best fits the user's needs.

2.4. Benchmarking metrics

Given the many blockchain platforms and consensus algorithms available, benchmarking is crucial for consistently comparing their performance and characteristics. By systematically measuring such factors, stakeholders can make informed decisions about which technology to adopt or how to configure it. Some of the key performance metrics include:

- **Throughout:** The rate at which the blockchain can process transactions, typically measured in transactions per second (TPS). Throughput represents the system's capacity to handle transaction load. High throughput is necessary for scalability. The more transactions a blockchain can handle per second, the more users it can onboard to the network. For example, for a trading company employing a blockchain system, a higher TPS means that more trades can be processed per second, and hopefully, more profit can be generated. The benchmarking tool BlockBench [DWC+17] measures throughput to compare two systems: Ethereum and Hyperledger Fabric. The results demonstrated that Hyperledger Fabric has more throughput capacity than Ethereum, but struggles to scale beyond 16 nodes [DWC+17]. Many recent blockchain systems aim to maximise TPS and report thousands of TPS in a permissioned setting, but benchmarks are needed to validate these numbers.

- **Latency:** The time it takes for a transaction to be confirmed and finalised, such that it becomes irreversible. Low latency is essential for a good user experience and for use cases that require fast settlement, for example, payment processing. It is often measured by subtracting the timestamp of when the transaction was submitted from the timestamp of when it was confirmed. Studies like Polge et al. (2021) highlight latency as a key criterion alongside throughput and note that high throughput per second (TPS) doesn't always mean low latency [PRLT21].

- **CPU consumption:** The percentage of CPU resources used by blockchain nodes during operation. This metric is related to the computational efficiency of the blockchain system. High CPU usage might indicate heavy cryptographic operations or inefficient code execution. Benchmarking CPU utilisation helps ensure the network can run on the intended hardware. It is also related to the Energy consumption metric. BCTMark benchmarking tool measured power usage to compare the environmental footprint of different blockchains [SLM20].

- **Memory usage:** The amount of memory (RAM) that is consumed by the blockchain while operating. A node that uses memory inefficiently could suffer slowdowns due to constant garbage collection or even node failure if it runs out of memory. Tracking memory usage is essential for the stability and longevity of the blockchain. In permissioned blockchains, memory can be used to queue pending transactions, store recent blocks, maintain network state and caches, and so on. Effective memory management is crucial for the network's ability to maintain performance over time.

- **Error Rate:** The frequency of failed transactions or communication errors in the network. This might include transactions that are rejected or timed out. A robust blockchain platform should have minimal errors, regardless of the network's size and load. High error rates could indicate problems such as network overload, malicious nodes, or other issues. In a benchmark, measuring error rate helps to demonstrate the effectiveness of the throughput. If the system achieves high TPS but half the transactions fail, its effectiveness diminishes.

- **Success Rate:** This is essentially the complement of error rate, measured as the percentage of submitted transactions that are successfully confirmed. A high success rate at high throughput indicates the system is handling the load effectively. A dropping success rate as load increases would be a good indicator that the network capacity limit has been reached.

- **Scalability:** This refers to how well the blockchain's performance metrics hold up as the scale of the system increases. Scaling is measured by increasing the number of validator nodes in the context of this Master's Thesis. Ideally, a scalable consensus will handle the increase in network density gracefully with a slight performance degradation. The scalability metric is essential for estimating whether a particular blockchain can meet future growth requirements.

- **Fault Tolerance:** In consensus benchmarking, one can test the system's resilience by intentionally introducing node failures or crashes. For example, in a leader-based blockchain network, if the leader node crashes, it's essential to know how quickly the new leader can be elected. Measuring fault tolerance is more of a qualitative test and can be crucial for understanding the reliability of a blockchain in the event of faults.

- **Network Utilisation:** If accessible, metrics like network bandwidth usage and packet loss can be helpful. The metric can indicate where the consensus algorithm is flooding the network with messages, which might not be clear from throughput and latency numbers. Packet loss or delay can be artificially introduced in some testbeds to see how the consensus reacts and recovers from it.

- **Qualitative Factors:** Qualitative metrics are also considered essential aspects of blockchain systems, such as ease of use, community support, and adoption of the technology. These factors may influence the final decision in selecting the right blockchain system. For example, a platform with a strong community and ongoing development might be favourable despite slightly lower performance. Polge et al. (2021) in their comparative study of five permissioned blockchain platforms explicitly included "adoption" (which encompassed factors like community size, maturity, and industry usage) as one of the benchmarking criteria [PRLT21]. In their analysis, Hyperledger Fabric scored the highest overall, due to strong performance in several categories and widespread adoption. Nevertheless, they noted that Hyperledger Fabric struggled with scalability.

3. Related Work

This chapter reviews prior comparative studies of permissioned blockchain systems and existing benchmarking frameworks. We highlight key findings and identify remaining research gaps.

3.1. Comparative Studies of Permissioned Blockchain Systems

Researchers have employed both empirical performance testing and theoretical analysis to evaluate permissioned blockchains:

- **ConsenSys Quorum Case Study:** Mazzoni et al. [MCDN22] conducted an in-depth evaluation of ConsenSys Quorum's three consensus mechanisms: Raft, Istanbul BFT (IBFT), and Clique (Proof-of-Authority). Their experiments revealed that Raft achieved the highest throughput due to its lightweight leader-based design. However, for financial applications, the authors recommended IBFT despite its lower throughput, citing its immediate finality and Byzantine fault tolerance as critical advantages. This finding demonstrates that throughput alone is an insufficient metric for selecting a platform.

- **Framework for Blockchain Consensus Protocols:** Leonardos et al. [LRP20] proposed PREStO, a theoretical framework for comparing consensus protocols across five key dimensions: Persistence (ledger reliability and transaction finality), Efficiency (resource utilization), Robustness (resilience to failures and attacks), Stability (performance under network fluctuations), and Optimality (network capability utilization). Applied to Bitcoin and Quorum, this framework enables systematic analysis of protocol trade-offs.

- **Operational Benchmarking Factors:** Rasolroveicy et al. [RHF23] extended performance evaluation to include operational metrics through their BlockCompass framework. They measured installation time, configuration complexity, error rates, and documentation quality—factors critical for practical deployment. This approach recognises that blockchain effectiveness depends on both performance characteristics and deployment feasibility.

- **Ethereum Client Comparisons:** Samuel et al. [SGVGO21] evaluated three Ethereum clients (Geth, Parity, and Besu) in private network configurations. Their experiments revealed that Geth's Clique implementation suffered from more frequent

forking under stress, whereas Besu and Parity had other limitations, like higher memory usage or slower block processing. Despite temporary forks produced in Geth, the study found that Geth overall provided the best performance and stability balance among the three clients.

- **A Framework for Analysing Private Blockchains:** Dinh et al. [DWC+17] conducted the first comprehensive benchmarking of permissioned systems (Ethereum Geth, Parity, and Hyperledger Fabric) using their BlockBench framework. BlockBench introduced a suite of workloads, encompassing both micro-benchmarks (to stress specific components such as the CPU or I/O) and application-level macro-benchmarks (based on realistic smart contracts). For example, they implemented a simple pyramid scheme contract called Doubler, where participants send funds to a contract, and earlier participants are paid out when new ones join. Results demonstrated that none of the systems could surpass traditional databases in transaction processing, with each exhibiting distinct bottlenecks.

- **Empirical Evaluation of Hyperledger Fabric and Ethereum:** Dabbagh et al. [DKTA20] conducted an empirical evaluation of Hyperledger Fabric vs. Ethereum using Hyperledger Caliper[1] as a benchmarking tool. They measured success rate, average latency, throughput, and resource consumption for a simple workload. Their small-scale experiments showed that Fabric outperformed Ethereum in terms of success rate, throughput, latency, and resource efficiency—advantages attributed to Fabric's BFT consensus and parallel execution architecture. However, the authors caution that these results may not scale to larger workloads.

In summary, our research analysis reveals that consensus mechanisms have a significant impact on both throughput and consistency. These technical trade-offs can be further complemented by qualitative metrics that affect system usability. The subsequent sections examine dedicated benchmarking frameworks designed to facilitate more comprehensive and systematic comparisons.

3.2. Established Benchmarking Frameworks

Several frameworks have been developed to benchmark blockchain systems, each employing distinct architectures and methodologies for performance comparison. Below, we review the most prominent frameworks.

3.2.1. BlockBench (2017)

BlockBench, developed by Dinh et al. [DWC+17], was among the first benchmarking frameworks designed for permissioned blockchains. Its architecture introduced micro- and macro-benchmarks for the experiments. **Micro-benchmarks** evaluated low-level operational metrics, including ledger database I/O, memory usage, and CPU consumption.

[1] https://github.com/hyperledger/caliper

These were measured by executing computationally intensive smart contracts to identify performance bottlenecks across platforms. **Macro-benchmarks** assessed system performance under realistic workloads, implemented via specialised smart contracts modelling real-world applications: cryptocurrency transactions, banking operations, an auction marketplace, identity management, and the Doubler pyramid scheme. These workloads were deployed on Ethereum, Parity, and Hyperledger Fabric, with throughput and latency measured under varying node counts to evaluate scalability.

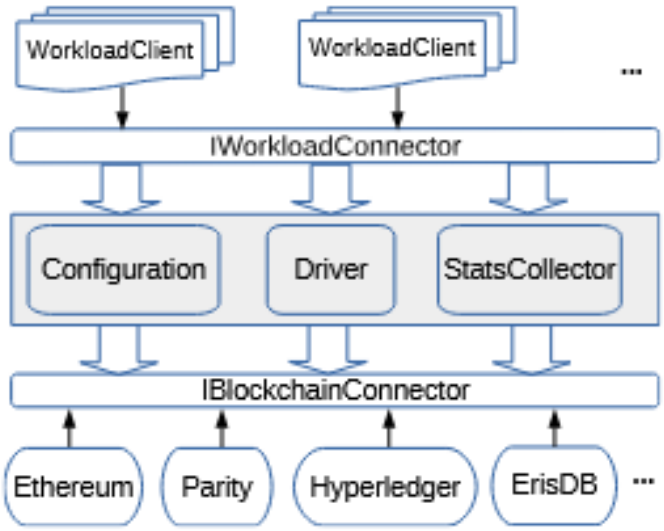

Figure 3.1.: BlockBench software stack. Source: [DWC+17, p.5]

BlockBench automates the deployment of target blockchain networks and employs a load generator to submit transactions according to predefined workloads. The generator records timestamps and outcomes to compute performance metrics, while a monitoring tool collects system statistics (e.g., CPU usage, network traffic). Figure 3.1 illustrates BlockBench's software stack. A key challenge was accommodating different platform architectures, such as Ethereum and Parity's account-based model with Proof-of-Work (PoW) versus Fabric's execute-order-validate paradigm with endorsement policies. BlockBench abstracts these differences through a unified interface, concealing consensus, storage, and endorsement mechanisms behind adapter modules and deployment scripts.

At its inception, BlockBench addressed the absence of a systematic performance evaluation framework for comparing blockchain systems under standardised conditions. Before BlockBench, comparisons between platforms like Ethereum and Fabric lacked consistent metrics or testing methodologies. By providing uniform workloads and evaluation criteria, BlockBench highlighted how design choices and decentralisation levels impact practical performance. However, its scope was limited to three hardcoded systems, and extending support for newer blockchains or versions required substantial effort without clear

guidelines. This limitation motivated subsequent framework developments.

3.2.2. BCTMark (2020)

BCTMark, introduced by Saingré et al. [SLM20], prioritised full automation and portability in blockchain benchmarking (Figure 3.2). The framework minimises manual intervention by leveraging SSH-based scripts for node orchestration, dependency installation, and monitoring agent deployment. Its adaptable deployment layer supports environments ranging from large virtualised clusters to low-power Raspberry Pis. Once deployed, BCTMark's workload engine generates synthetic transaction streams, while continuous monitoring captures throughput, latency, I/O rates, CPU usage, and power consumption (in watts). The inclusion of performance-per-watt metrics facilitates sustainability assessments for permissioned blockchains.

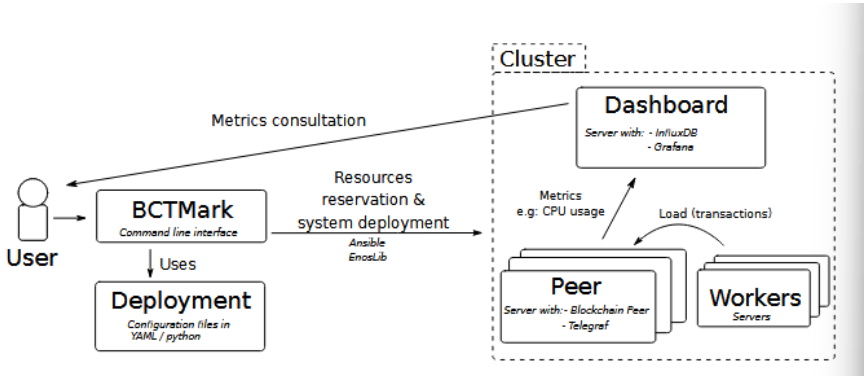

Figure 3.2.: BCTMark architecture. Source: [SLM20, p.6]

BCTMark is built with state-of-the-art industry tools. It uses Docker[2] containers to deploy nodes and Ansible[3] to orchestrate the setup. Telegraf[4] plugin to collect metrics and InfluxDB[5] to sotre collected resource metrics. Figure 3.2 depicts its architecture. Load generation is implemented in Python[6] and Go[7], sending transactions via the respective client SDKs or RPC calls of each blockchain. The framework supports Ethereum (with Ethash PoW and Clique PoA) and Hyperledger Fabric.

BCTMark tackled the problem of experiment reproducibility. Many earlier benchmarks were tied to a fixed lab environment. BCTMark allowed researchers to deploy the same experiment on a cloud cluster. It emphasised comparing different blockchain technologies under the same unified framework. Additionally, with the growing interest in permissioned

[2] https://www.docker.com/
[3] https://www.ansible.com/
[4] https://github.com/influxdata/telegraf
[5] https://github.com/influxdata/influxdb
[6] https://www.python.org/
[7] https://go.dev/

blockchains in IoT industries, where devices may be resource-limited, BCTMark's ability to run tests directly on Raspberry Pi addresses that need.

One limitation of BCTMark was that it was a generic approach. It primarily covered a narrow set of consensus technologies and generic, well-known blockchain systems. The authors themselves noted the need to extend to more protocols.

3.2.3. BlockCompass (2023)

BlockCompass, developed by Rasolroveicy et al. [RHF23], represents a comprehensive benchmarking solution for diverse blockchain systems and consensus algorithms. It supports Ethereum (Proof-of-Work and Proof-of-Authority), Hyperledger Fabric (Raft), and Hyperledger Sawtooth (Proof-of-Elapsed-Time, Practical Byzantine Fault Tolerance and Raft) out of the box. The framework employs an adapter-based architecture, where each blockchain integrates via a dedicated driver module, allowing for extensibility without requiring core modifications. A user-friendly interface provides real-time monitoring of metrics, including CPU/memory usage, network I/O, transactions per second (TPS), latency, and error rates, alongside automated report generation. BlockCompass emphasised user-friendly UI, as many earlier tools mainly relied on users running them via the command line. To validate the usability of the UI, the creators evaluated BlockCompass on various quality properties. They measured how easily new users could install and set up the BlockCompass. To assess this, a group of 33 students used BlockCompass and provided feedback on the documentation and user experience. The result of this feedback loop is a framework that not only works across many technologies but is also well-documented and designed to minimise user frustration.

Figure 3.3 illustrates BlockCompass's architecture, comprising a workload generator, blockchain clients, a resource monitor, and a backend-frontend pipeline—all containerised via Docker. The workload generator configures the target blockchain and initiates transactions, while the resource monitor aggregates node data for real-time visualisation (via the UI) and storage in MongoDB. The abstract client interface permits seamless integration of new blockchains by implementing a corresponding driver.

BlockCompass addresses longstanding usability challenges in blockchain benchmarking. By unifying diverse systems under an extensible architecture and simplifying comparative analysis through intuitive interfaces, it reduces the overhead of performance evaluation. Furthermore, its comprehensive metric collection and live monitoring capabilities offer more profound insights into system behaviour under load. In summary, BlockCompass enhances the usability of benchmarking tools while providing support across multiple blockchain platforms.

3.3. Limitations in Current Methodologies

Despite advancements in blockchain benchmarking frameworks, existing approaches exhibit notable gaps in platform coverage and consensus algorithm evaluation. The current literature predominantly focuses on mainstream systems, such as Ethereum

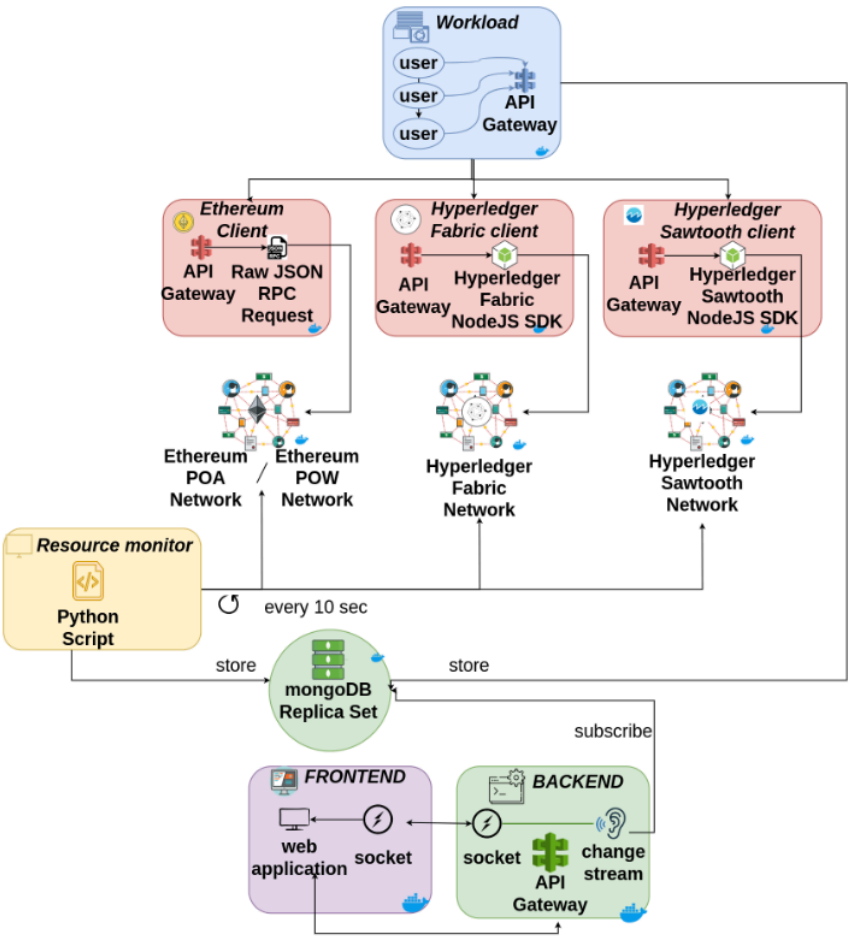

Figure 3.3.: BlockCompass architecture. Source: [RHF23, p.4]

and Hyperledger Fabric, while neglecting other prominent platforms. As Table 3.1 demonstrates, while BlockBench, BCTMark, and BlockCompass all evaluate Ethereum and Fabric, only BlockCompass includes Sawtooth, and only BlockBench incorporates Parity. No existing framework has assessed platforms such as Cosmos (Tendermint consensus), R3 Corda (Raft-based), or Diem (Libra's HotStuff BFT), despite their industry adoption. This limited scope leaves organisations without empirical data for comprehensive technology comparisons.

We identify three primary limitations in current benchmarking approaches:

- **Narrow scope:** Most benchmarks to date focus on Ethereum and Hyperledger-based systems, leaving out platforms like Tendermint/Cosmos, Corda, Quorum-IBFT, or Diem/Libra. This means essential technologies have not been rigorously compared under unified conditions.

BlockCompas	BlockBench	BCTMark
Hyperledger Fabric	Hyperledger Fabric	Hyperledger Fabric
Ethereum	Ethereum	Ethereum
Hyperledger Sawtooth	Parity	

Table 3.1.: Blockchains used for benchmarking by existing tools

- **Outdated Data:** Several comparative studies use old software versions that are no longer representative. As these platforms have evolved, earlier performance results may become obsolete, necessitating updated evaluations. For example, BlockBench's evaluation used Hyperledger Fabric v0.6, whereas the current version is v2.x, demonstrating significantly improved performance. Similarly, Ethereum clients have undergone substantial optimisations, rendering earlier performance data potentially obsolete.
- **Limited Extensibility:** Existing benchmarking frameworks often lack clear instructions on how to support new blockchain systems. Many are hard-coded for specific platforms or consensus types, making it non-trivial to plug in a different blockchain and thus slowing down comprehensive comparisons.

These limitations motivate the novel framework proposed in this Master's Thesis, which addresses these gaps through:

- Support for diverse permissioned blockchain platforms beyond traditionally studied systems
- Comprehensive extension guidelines for incorporating new blockchains
- Up-to-date performance comparisons

The subsequent chapters detail the design of our framework and its application in benchmarking Ethereum Clique, Quorum Raft, and Tendermint systems. This work aims to provide decision-makers with comprehensive, current data for informed selection of blockchain technology.

3.4. Importance of an Active Community

Effective blockchain benchmarking must consider factors beyond technical performance metrics. A platform's long-term viability heavily depends on the size and activity of its developer community. Vibrant communities drive continuous improvements, prompt bug fixes, and foster robust tool ecosystems. Community engagement metrics—such as GitHub stars and recent commits—serve as reliable indicators of a platform's long-term sustainability.

Table 3.2 presents current community activity metrics for major permissioned blockchain platforms from June 2025. While Ethereum leads in GitHub stars, this partly reflects its

Blockchain	Number of GitHub Stars	Last Commit
Ethereum	49.2k	2025-06-27
Diem	16.7k	2023-06-20
Hyperledger Fabric	16.2k	2025-06-27
Cosmos	6.6k	2025-06-23
Quorum	4.7k	2025-05-27
R3 Corda	4k	2025-06-27
Hyperledger Sawtooth	1.4k	2024-02-01

Table 3.2.: Developer community activity metrics for permissioned blockchain platforms (June 2025)

dual role as both a public and a permissioned blockchain. Among purely permissioned platforms, Hyperledger Fabric, Quorum, R3 Corda, and Cosmos demonstrate vigorous ongoing development activity. Notably, Diem (the discontinued Libra project) has shown no activity since June 2023, despite its historical popularity, while Hyperledger Sawtooth's development appears stagnant, with no commits since February 2024. These metrics provide valuable context for evaluating platform longevity in conjunction with its performance characteristics.

4. Framework Design And Implementation

This chapter presents the design of a modular, extensible benchmarking framework capable of evaluating diverse permissioned blockchain systems with minimal configuration changes. The architecture enables complete test cycle execution—from network deployment to workload execution and metrics collection—through a single command. The design combines automation, scalability, and extensibility to support new blockchain platforms with minimal adaptation.

4.1. Design Overview

The framework employs a layered architecture comprising three core components: a Backend Service, a Frontend Interface, and Blockchain Configuration files. This separation of concerns ensures distinct responsibilities and facilitates future enhancements.

As illustrated in Figure 4.1, user interaction occurs through the Frontend UI, which communicates with the Backend server. The backend orchestrates test execution on the blockchain network using parameters defined in configuration files. Key design objectives include:

Key design objectives include:

- **Extensibility:** Minimal code modifications required to add support for new blockchains
- **Full Automation:** End-to-end process automation from network setup to teardown ensures result reproducibility
- **User-Friendliness:** Single-command test initiation with results visualisation through an interactive dashboard

4.1.1. Backend Service Layer

The **Backend Service** serves as the central controller for benchmarking operations. Implemented as a Node.js application using the ExpressJS framework, it exposes RESTful endpoints to coordinate the test workflow. The backend architecture focuses on four critical functions:

- **Network Orchestration:** Manages blockchain network deployment and teardown using Docker containerisation. Each node runs in an isolated container, allowing for

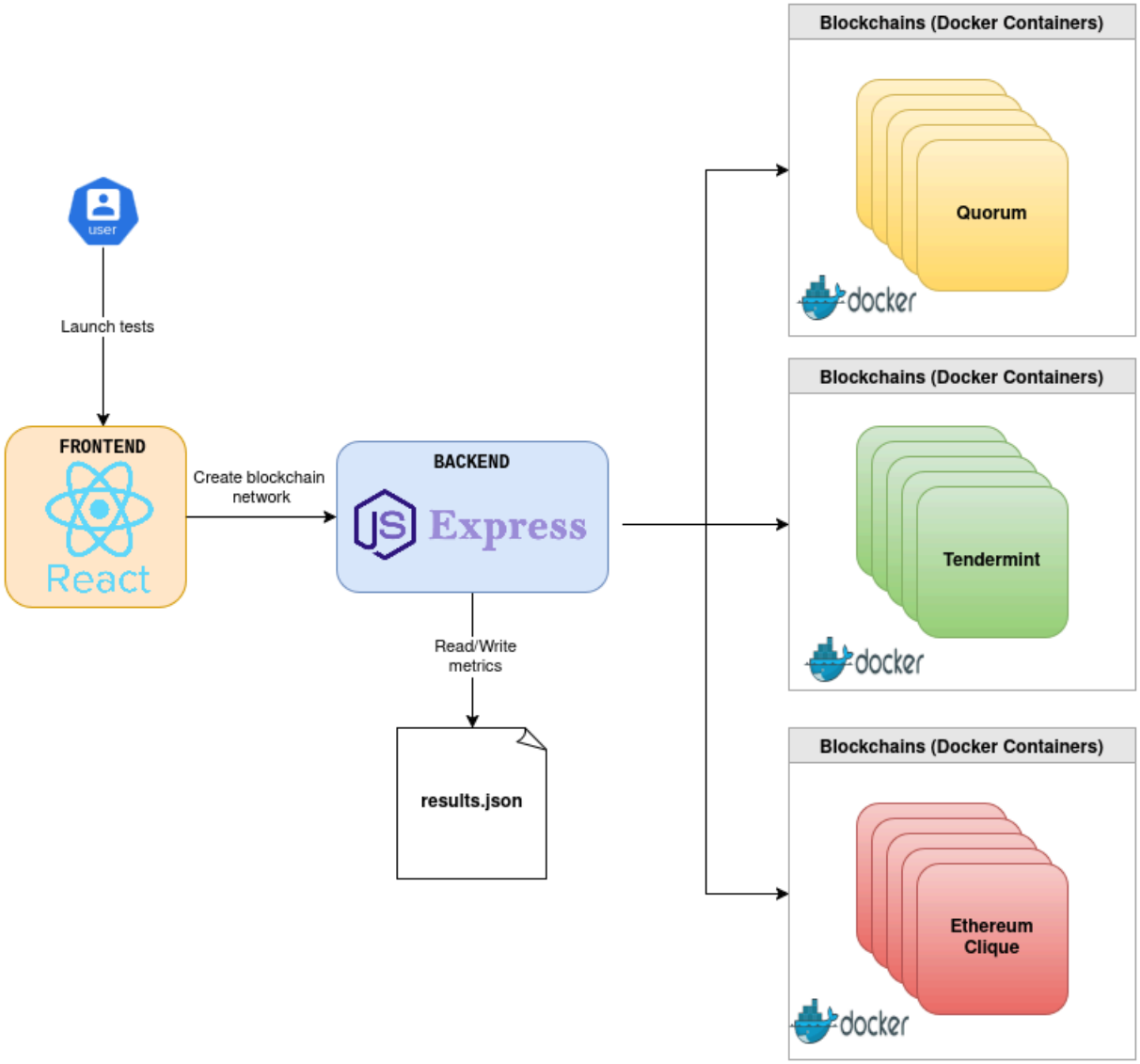

Figure 4.1.: High-level framework architecture

the emulation of various network sizes (e.g., 4, 8, 16, or 32 nodes) on a single host machine. Predefined blockchain-specific scripts ensure proper node initialisation and private network formation.

- **Workload Generation:** Simulates client activity by injecting transactions at configurable rates. The framework incorporates a load generator library capable of producing high transaction volumes (e.g., 10,000 transactions across 50 concurrent threads). This abstraction enables simple workload parameter adjustments without requiring framework modifications.

- **Metrics Collection:** Captures performance indicators during test execution. The framework focuses on transaction throughput, latency, as well as CPU and memory usage on each node. Throughput and latency are critical for understanding the capacity and responsiveness of a blockchain, while resource utilisation indicates efficiency. The design ensures that as soon as the workload execution finishes, the backend aggregates the metrics.

- **Data Persistence:** Stores test results for subsequent retrieval and analysis. Rather than implementing a complex database system, the design employs lightweight JSON file storage. This approach provides:
  - Simple implementation with fast read/write operations
  - Easy result portability across machines
  - Reproducibility of test conditions

  Sample test outputs from this research are provided in Appendix A.1.

To facilitate these operations, each supported blockchain maintains a configuration file specifying network initialisation parameters and transaction endpoints. This separation keeps the backend logic generic while accommodating platform-specific requirements.

For the backend, the Node.js/ExpressJS combination was selected for its:

- Rich ecosystem of maintained packages
- Rapid development cycle
- Established REST API implementation patterns

Docker provides several advantages for containerisation:

- Wide platform support and standardisation
- Dependency isolation through containerisation
- Simplified multi-node deployment via Docker Compose (used by comparable benchmarking frameworks)
- Native API support for container metric collection

For the backend, the Node.js/ExpressJS combination was selected for its rich ecosystem of maintained packages, rapid development cycle and well-established REST API implementation patterns. Docker is a widely supported containerisation platform that ensures each blockchain node has the required dependencies and correct configuration, thereby enabling its successful execution on any machine that supports it. Docker offers Docker Compose, which supports simplified multi-node deployment and native API support for container metric collection.

4.1.2. Frontend Service Layer

The **Frontend Service** is implemented as a React-based web interface designed to facilitate user interaction with the benchmarking framework. It provides three core functionalities:

- **User Controls:** Presents configurable test parameters through interactive UI elements (Figure 4.2). The blockchain selection and network size options (4, 8, 16, or 32 nodes) are dynamically populated from backend configuration files, automatically reflecting newly added blockchain systems to maintain extensibility.
- **Test Execution Monitoring:** Initiates benchmarking through backend API calls and provides test initiation status. The current implementation requires manual page refreshes for status updates; however, in the future, the design could be extended to include real-time updates.
- **Result Visualisation:** Displays performance metrics through an optimised dashboard interface that incorporates six colourful bar charts(Figure 4.3). The visualisation system presents six key metrics:
  - Throughput (requests/second)
  - Latency (milliseconds)
  - CPU usage (average and peak percentage)
  - Memory consumption (average and peak MB)

  Initial experiments with 3D chart visualisations (Figure 4.4) were abandoned in favour of 2D bar charts due to superior readability and comparative analysis capabilities. The UI is kept minimalistic, showing essential metrics side by side.

A web-based UI maximises accessibility, so that anyone with a Docker setup and a browser can use the tool. React was chosen due to its popularity and rich ecosystem, including numerous charting libraries for data visualisation. A React-based dashboard can efficiently render updates to enable overlaying multiple results on charts, allowing users to compare specific blockchain systems against each other.

The frontend architecture abstracts all backend complexity through REST API interactions, making the tool accessible to users without Docker or blockchain expertise. The React framework for UI was selected for its:

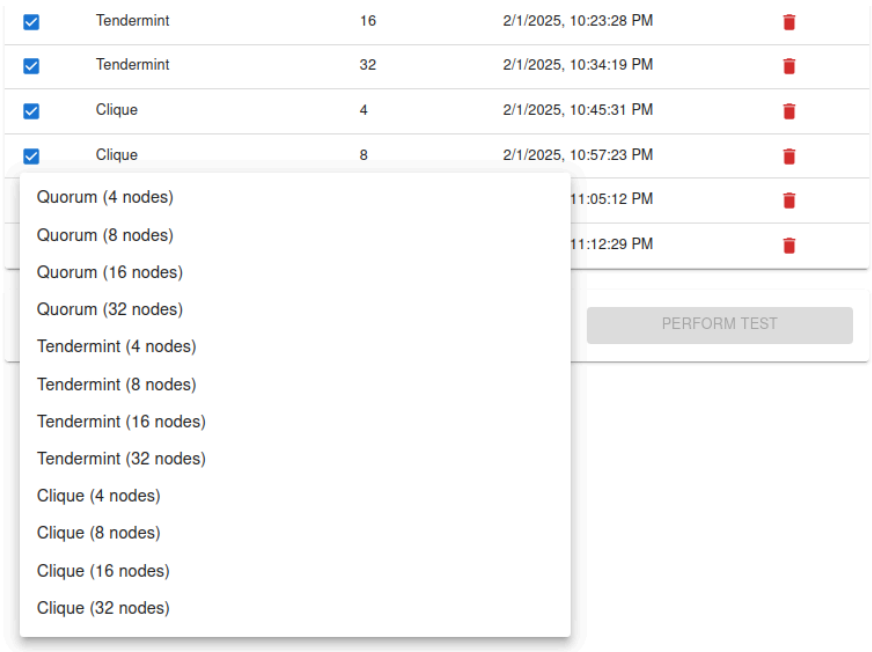

Figure 4.2.: Network configuration selection interface

Figure 4.3.: Benchmarking results dashboard with comparative metrics visualisation

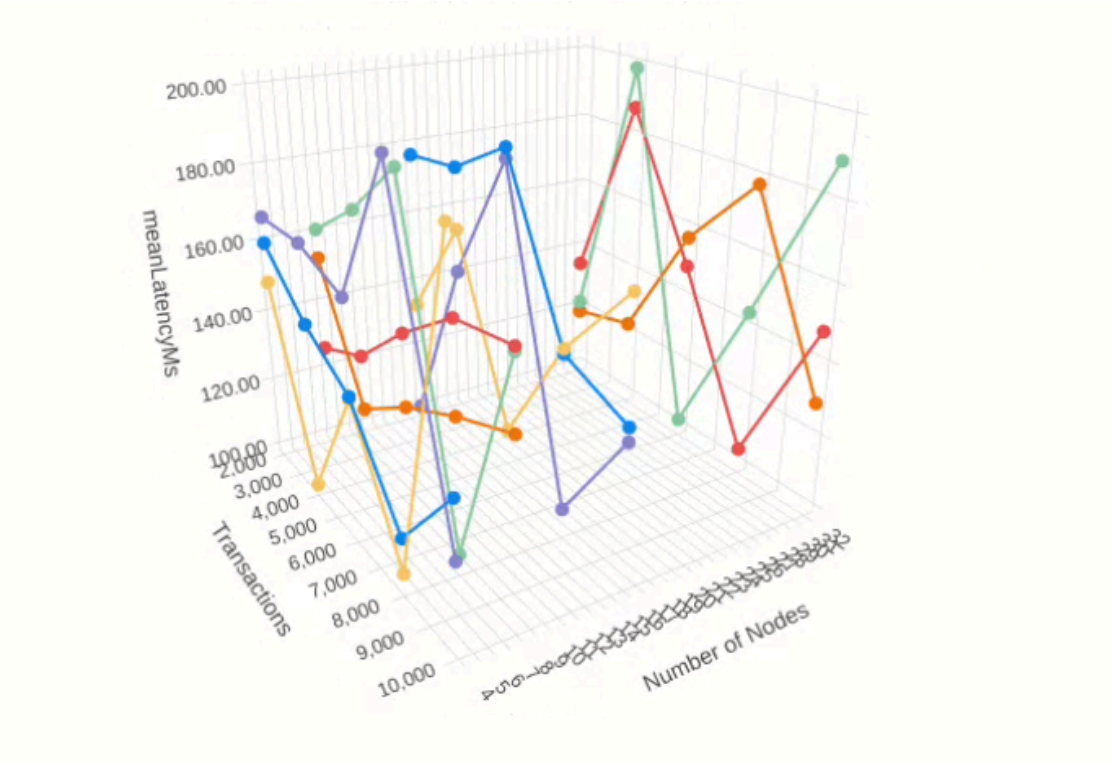

Figure 4.4.: Deprecated 3D visualization approach demonstrating reduced interpretability

- Extensive developer adoption and community support
- Rich ecosystem of data visualisation libraries
- Instant graph rendering capabilities for comparative analysis

4.1.3. Blockchain Configuration files

To achieve the goal of easy extensibility, the framework centralises all blockchain-specific details into a configuration file named blockchain_config.json. This JSON file serves as a registry of supported blockchain systems and their respective operations. Listing 4.1 demonstrates an example configuration entry for the Quorum blockchain network with 16 nodes.

```
{
    "name": "Quorum",
    "nodes": 16,
    "startCommand": "cd ./quorum/16nodes && make localnet-start",
    "stopCommand": "cd ./quorum/16nodes && make localnet-stop",
    "warning":"  ",
    "endpoint": "http://localhost:22001",
    "requestBody": {
        "jsonrpc": "2.0",
        "method": "eth_sendTransaction",
        "params": [{
            "from": "0xed9d02e382b34818e88b88a309c7fe71e65f419d",
            "to": "0xca843569e3427144cead5e4d5999a3d0ccf92b8e",
            "value": "0x9184e72a"
        }]
    }
}
```

Listing 4.1: Configuration entry for Quorum with 16 nodes

Each blockchain entry must include the following fields:

- **name:** Identifier of the blockchain platform (e.g., "Quorum", "Tendermint", "Clique").
- **nodes:** The network size (number of nodes) for which this configuration is defined. There are multiple entries per blockchain for different node counts.
- **startCommand:** The shell command to launch the blockchain network in a Docker environment. For example, an entry might have a field "startCommand": "cd ./tendermint make localnet-start NUM_NODES=4" to spin up a 4-node Tendermint network. The backend later executes this command in a subprocess.
- **stopCommand:** The command to cleanly shut down the network and remove containers once the test is complete (e.g., make localnet-stop). Proper teardown is essential to free system resources and allow subsequent tests to start fresh.

- **endpoint:** The network endpoint where the blockchain listens for client transactions. This is a REST URL address to which the load generator should send transactions. For instance, a Quorum node might expose an HTTP RPC at http://localhost:8545 for transactions.
- **requestBody:** A template or example of the request payload to be sent for each transaction. Different blockchains have different transaction formats. The load generation module uses this to construct actual requests.
- **warning (optional):** Any special instructions or warnings for the user. In our case, for Tendermint configurations, we include a warning that the test will generate many files and that removal requires administrator privileges (sudo). This warning is also displayed in the UI before launching the test, alerting the user that they must manually clean up the resources. Such a warning field is a design convenience for handling exceptions; ideally, all blockchains would clean up automatically, but Tendermint's file permissions necessitate this caution.

In essence, the configuration file acts as a plug-in mechanism for blockchain drivers. This configuration-driven approach simplifies the process of adding a new blockchain. It eliminates the need to modify the core backend code for new blockchain systems, provided they can be launched with a **startCommand** and accept transactions via an API. It's important to note that the framework currently supports only the Linux platform due to its reliance on Docker and Unix shell commands. Cross-platform support would require adjustments to the commands and is left for future work.

4.1.4. Benchmarking Workflow

The end-to-end benchmarking process follows this sequence (illustrated in Figure 4.5):

1. **User Input:** The user selects a blockchain platform and a network size (e.g. Quorum with 16 nodes) from the frontend UI and clicks "Start Test". The frontend may also display a warning if applicable.
2. **API Call:** Frontend triggers backend execution via REST call, including in the request, which blockchain and network size were chosen.
3. **Network Deployment:** Upon receiving the request, the backend looks up the corresponding entry in blockchain_config.json and executes Docker-based network initialisation. The backend waits until the network is ready.
4. **Workload Execution:** Next, the backend's workload generator module uses the API endpoint from the config and starts sending transactions. The generator uses the requestBody template for Quorum to craft each transaction request. The system performs:
    - 5 test iterations for statistical significance
    - 10,000 transactions across 50 threads (default)

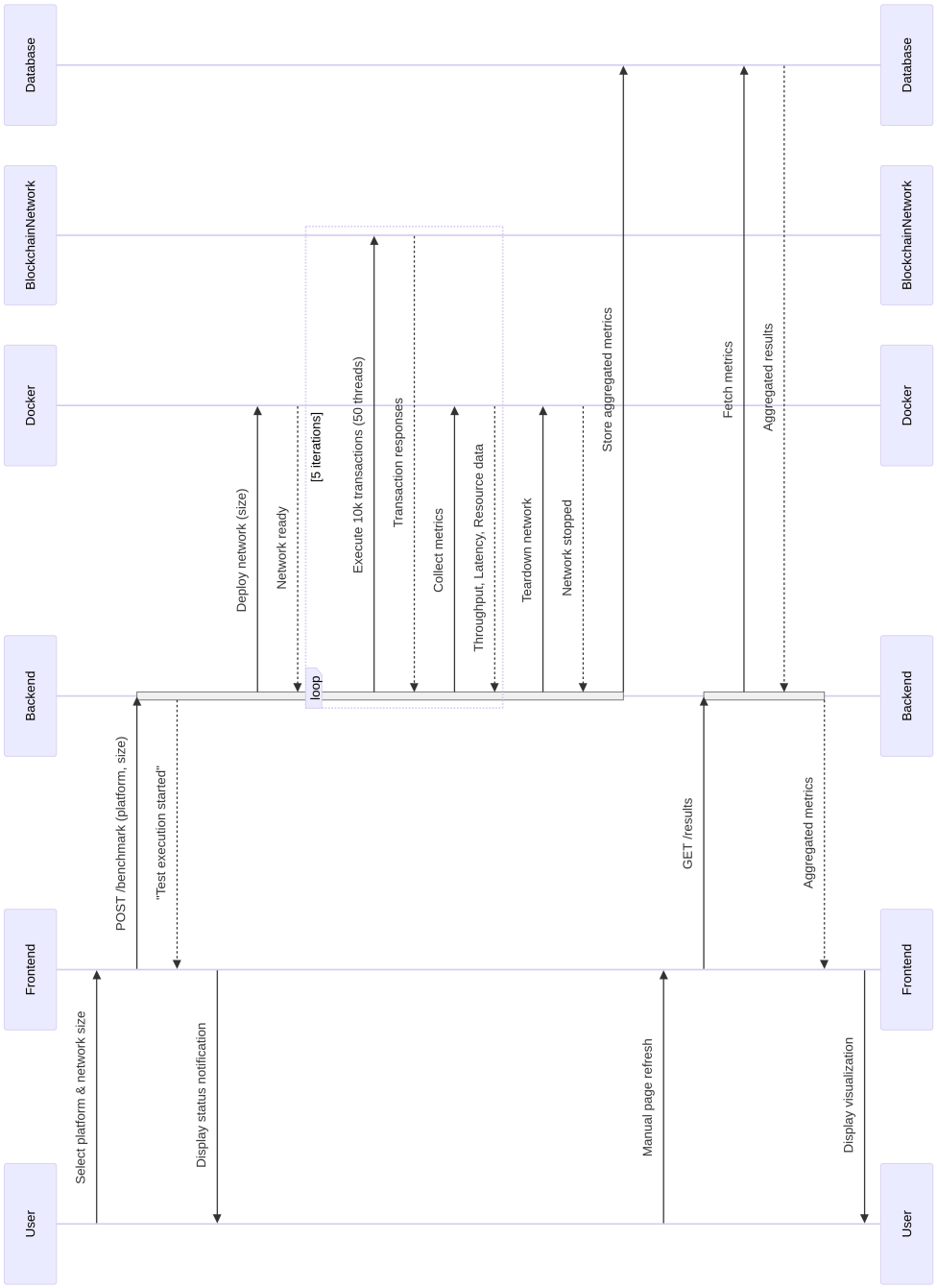

Figure 4.5.: Sequence diagram for benchmarking workflow

- Platform-specific transaction formatting

5. **Metrics Collection:** Comprehensive monitoring of:
   - Throughput: $\frac{\text{total transactions}}{\text{total time}}$
   - Latency: $\frac{\sum \text{response times}}{\text{transaction count}}$
   - Constant resource utilisation data fetching via Docker API. Average and peak % CPU usage across all nodes, and peak and average memory consumed by any node during the run.
6. **Network Teardown:** After sending all transactions and collecting metrics, the backend executes the stopCommand from the config to tear down the blockchain network. All containers, associated networks, and any temporary files or directories that were created will be deleted. In the case of Tendermint, where sudo is required, the user must perform this step manually, for which he is warned at the start of the test.
7. **Result Aggregation:** The backend computes and stores aggregate metrics in the simple JSON database.
8. **Visualisation:** Since test runs can take up to 20 minutes, we don't make the initial API call wait for the response, but rather the user needs to refresh the page to see the results of the latest run. The new data point (Quorum, 16 nodes) is added to the dataset and rendered in the charts on the UI.

The test execution sequence is fully automated – once the user triggers it, no manual intervention is required (except for the cleanup of the Tendermint file).

4.2. Framework Implementation

This section outlines the technical implementation of our benchmarking framework, including the technology stack, configuration parameters, and key design decisions. We provide detailed implementation information for metric computation and justify our tool selections in relation to industry standards and best practices.

4.2.1. Backend Implementation

The backend service was implemented using Node.js with the ExpressJS framework. Three key considerations influenced the choice of Node.js:

- Asynchronous event-driven architecture for efficient orchestration of concurrent Docker processes and load executions
- Lightweight HTTP server capabilities with minimal overhead
- Extensive npm library ecosystem for rapid development

For transaction simulation, we integrated the `loadtest` npm package, a well-established load testing solution that provides comprehensive performance statistics, including:

- Minimum/average/maximum request Latency
- Achieved throughput (requests/second)
- Error rates and distribution

The collected metrics are persisted in a structured `results.json` file following the schema shown in Listing 4.2:

```
{
 "id": 5,
 "blockchainName": "Quorum",
 "numberOfNodes": 16,
 "timestamp": "2025-05-01T10:30:00Z",
 "totalErros": 0,
 "rps": 64,
 "meanLatencyMs": 802.64,
 "averageCpu": 41.98,
 "averageMemory": 1562.53,
 "peakMemory": 1652.12,
}
```

Listing 4.2: Metric result data structure schema

Each record is given a unique ID (for reference in deletion or comparison) and a timestamp. We include identification info (blockchain type and size) because the UI uses that to label the results. This JSON file is small (a few KB per result), so even with dozens or hundreds of results, it remains efficient to load into memory. Throughput is represented by the **rps** field, which denotes requests per second, while the meanings of the remaining fields are self-explanatory.

We implemented a set of HTTP endpoints in Express to allow the frontend to interact with the backend. Table 4.1 summarises the core four REST API endpoints:

These endpoints represent the core functionality of the framework. The asynchronous design of the `/blockchain` endpoint was explicitly implemented to handle long-running tests (up to 20 minutes) without blocking the UI. Users monitor test completion via backend logs and refresh the frontend to view results.

4.2.2. Docker Orchestration Implementation

The implementation experience strongly encouraged us to use Docker, as the examples of existing blockchains often included Docker sample networks that we could use and modify according to our needs. Additionally, using Docker allowed us to treat each node as identical from the perspective of resource monitoring. Implementing Docker orchestration required writing Docker Compose files for each blockchain system: Quorum, Tendermint, and Ethereum Clique.

Endpoint	Method	Functionality
`/networks`	GET	Retrieve the list of supported blockchain configurations. Returns a JSON list of available blockchain systems and network sizes (populated from `blockchain_config.json`). The frontend uses this to populate dropdowns (Figure 4.2).
`/blockchain`	POST	Initiate a blockchain test. The request body contains the selected blockchain name and node count. Triggers network startup, workload execution, and metric collection. Returns a confirmation that the test started (and eventually, the results are available for retrieval via `/results`).
`/results`	GET	Fetch all recorded results. Returns the contents of the results from the JSON file. The frontend calls this to retrieve data for plotting on the graphs.
`/results/{id}`	DELETE	Delete a specific result entry identified by `{id}` from the results history. This allows users to remove unwanted or test entries.

Table 4.1.: REST API endpoints provided by the benchmarking framework backend

- **Quorum:** Extended the official 7-node reference implementation to support 4, 8, 16, and 32-node configurations using custom Docker Compose files. We used the flexibility of the Makefiles to bring up the network (*make localnet-start*) and to shut it down (*make localnet-stop*).
- **Tendermint:** Adapted the core team's sample localnet script that can initialise a network of a given size. The system writes data to directories which require `sudo` for file permission handling. Hence, the clean-up requires a **sudo rm -rf** admin-level command that needs to be executed by the user.
- **Ethereum Clique:** Modified the sample PoA implementation with a fixed validator set (7 signers maximum, limited by the design) and additional observer nodes. Therefore, in networks larger than 7, only 7 were actual signers, and the rest were regular nodes.

Due to memory constraints, the 32-node Quorum configuration could not be executed on a local machine with 16 GB of RAM, as it resulted in a system crash. Consequently, the deployment was carried out on an Azure D8S v3 virtual machine with 32GB of RAM.

4.2.3. Frontend Implementation

The frontend was developed using React. The decision for React was primarily based on the author's familiarity with the framework and its rich ecosystem for data visualisation. We evaluated several React chart libraries (Recharts, Chart.js, etc.) and ultimately chose Recharts due to its simplicity and seamless integration with React state. The dashboard implements real-time chart updates through React's state management system, automatically re-rendering visualisations when new test results are loaded or when users filter the displayed datasets.

5. Experimentation - Results and Analysis

This chapter presents a rigorous performance evaluation of three permissioned blockchain platforms (Quorum, Tendermint, and Ethereum Clique) across network scales ranging from 4 to 32 nodes. The evaluation covers key metrics: Throughput (transactions per second), Latency (transaction confirmation time), CPU usage (%), and Memory usage (MB). By comparing these systems side by side with actual experimental data, we highlight how different consensus mechanisms impact scalability and efficiency. All tests were conducted on a Microsoft Azure cloud Virtual Machine (16 vCPUs, 64 GB RAM) to accommodate up to 32 dockerised nodes per network. The framework automatically deployed each blockchain network with the specified number of nodes, executed a fixed workload of 10,000 transactions (at 50 concurrent clients), and gathered performance metrics. The target metrics are defined as:

- **Throughput:** The number of transactions processed per second by the network. Higher throughput indicates that the network can handle a greater load.
- **Latency:** The average time from submitting a transaction to it being confirmed in a block (client-side measurement). Low Latency implies fast confirmation, an important factor for user experience and certain applications
- **CPU Usage:** The average and peak CPU utilisation per node during the test. High CPU usage indicates that consensus and transaction processing are computation-intensive.
- **Memory Usage:** The average and peak memory used per node (in MB) during the test. Memory is important for scalability – if memory grows significantly with more nodes or load, it limits the real-world application of the system for a bigger audience.

5.1. Quorum

Quorum implements an Ethereum-compatible blockchain with Raft consensus, providing crash fault tolerance (CFT) through a leader-based architecture. However, it is not Byzantine fault-tolerant; therefore, it assumes nodes are honest and not malicious. Raft's appeal lies in its simplicity and immediate finality in small networks.

Figure 5.1 demonstrates how Quorum's performance changes as the network grows:

- **Throughput:** Quorum demonstrates significant throughput degradation, declining 81% from ∼339 TPS at 4 nodes, dropping to ∼64 TPS at 32 nodes. This steep

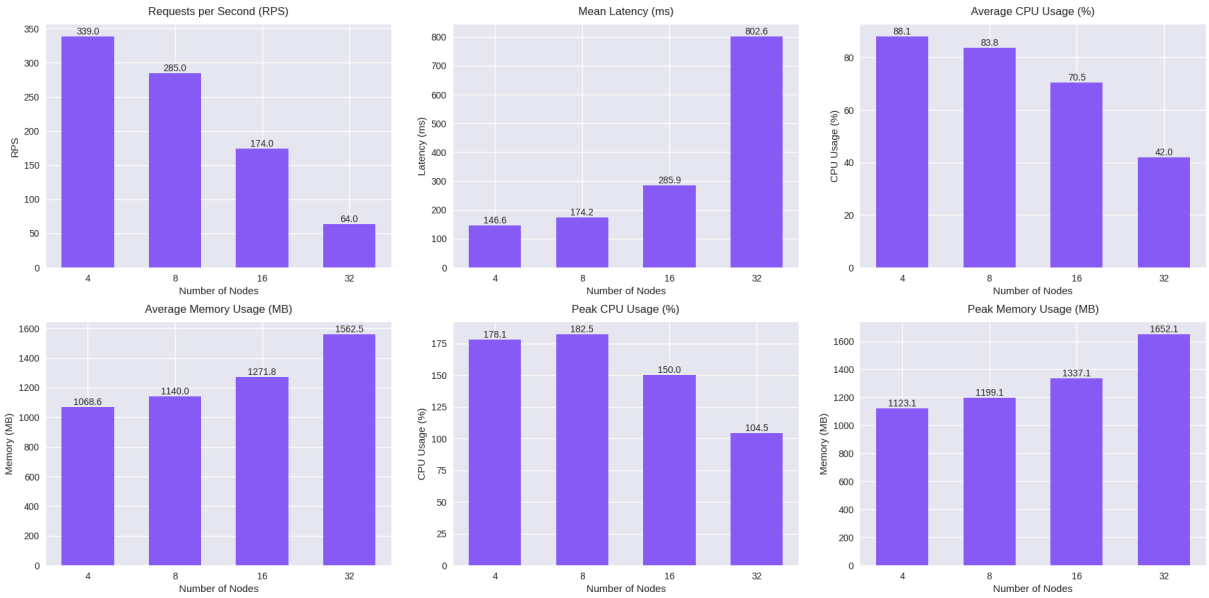

Figure 5.1.: Quorum performance metrics across network scales (4-32 nodes)

fall stems from Raft's $O(n)$ communication complexity - the leader must broadcast blocks to all followers and await acknowledgements. Therefore, the communication overhead and coordination costs increase with the addition of more nodes. Thus, while Raft is lightweight in small clusters, the throughput bottleneck becomes evident at larger scales.

- **Latency:** Latency grows nearly 6 times when scaling. Average transaction latency increased from ∼147 ms (4 nodes) to ∼803 ms (32 nodes). At 32 nodes, the consensus round likely experiences delays; the leader must wait for confirmations from a majority of nodes before finalising each block. More nodes result in increased network communication messages, which in turn lead to higher Latency. The trend indicates that adding nodes for Quorum sharply worsens Latency, which could hinder real-time applications on large networks.

- **CPU:** CPU usage drops as network size increases. At 4 nodes, Quorum's nodes were busy (88% avg CPU), but by 32 nodes, the average CPU dropped to ∼42%. The results suggest that in larger Quorum networks, nodes spend more time idle (with lower CPU usage). At the same time, the leader is active and performing at the maximum capacity of the container. Essentially, the network couldn't utilise all the nodes' CPUs to their maximum potential.

- **Memory:** Memory footprint grows with increasing number of nodes. Average memory per node rose from ∼1069 MB (4 nodes) to ∼1563 MB (32 nodes). The

reason for this is that Quorum's Geth-based nodes maintain more peer connections and store block data in memory when more nodes are present.

Quorum's ease of use and immediate finality make it a good choice for smaller networks. It offers ∼150 ms finality time and decent throughput of 339 TPS at four nodes. However, our findings show that performance degrades substantially as more nodes are added, clearly reflecting Raft's limitation. In larger networks, Raft's lack of scalability leads to under-utilisation of resources and poor throughput. For a consortium that needs to scale to dozens of members and handle Byzantine behaviour, Quorum with Raft would not be a suitable choice. Alternative consensus mechanisms or architectural approaches would be necessary for large-scale permissioned networks.

5.2. Tendermint

Tendermint implements a Byzantine Fault Tolerant (BFT) consensus algorithm that requires a $\frac{2}{3}$ majority of validators to commit each block. This provides safety guarantees against malicious nodes while maintaining liveness. Tendermint is often praised for high performance in small networks and is used in Cosmos[1] and other projects for fast transactions. Figure 5.2 presents our experimental results across network scales:

- **Throughput:** Tendermint achieved high throughput at low node count, hitting ∼4505 TPS with just four nodes, which is a significantly higher number than Quorum or Clique have in the same 4-node. This underscores the efficiency of Tendermint's consensus in a small, controlled setting – the protocol can finalise blocks very quickly when the validator count is low and network Latency is minimal, thanks to its optimised BFT consensus. Even at eight nodes, it managed ∼4230 TPS. By 32 nodes, performance dropped significantly and dropped to ∼859 TPS at 32 nodes, an 81% decline from 4 nodes. The steep fall reflects the known scalability limitation of BFT consensus. The communication complexity increases, since Tendermint's algorithm involves 2 phases of voting for each block, so roughly $O(n^2)$ increase in the number of messages exchanged per block. As more validators participate, the network spends more time on coordination and voting, reducing overall throughput.
- **Latency:** One of Tendermint's strengths is consistently low Latency thanks to fast block finalisation. In the 4-node network, average transaction Latency was an extremely low ∼10.6 ms. Even at 32 nodes, Latency remained only ∼57.4 ms on average. The results highlight how Tendermint's protocol optimises for quick consensus rounds. Every block is committed in one consensus round, and transactions are confirmed as soon as 2/3 of the validators sign the block. The slight rise in Latency with more nodes is expected: more validators mean more votes to collect. Low Latency, combined with BFT safety, is a significant advantage for Tendermint in applications where speed and fast finality are critical, such as stock trading or bank transactions.

[1] https://cosmos.network/

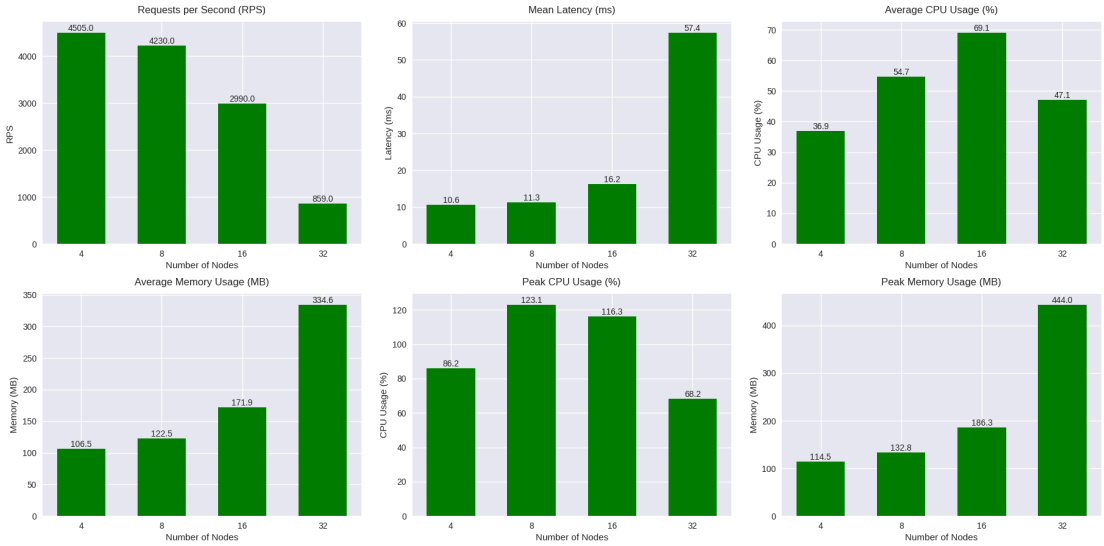

Figure 5.2.: Tendermint performance metrics across network scales (4-32 nodes)

- **CPU:** Tendermint's average CPU went from ∼37% (4 nodes) to ∼55% (8 nodes) to ∼69% (16 nodes), then dropped to ∼47% at 32 nodes. The CPU utilisation per node was under 70% on average across all networks, indicating a moderate computational load and some room for additional load.

- **Memory:** Memory usage remained low, from ∼107 MB (4) up to ∼335 MB (32) per node. Tendermint's implementation is lightweight in terms of memory; even at 32 nodes, it uses only a few hundred MB, far less than Quorum's usage in GB.

Tendermint showcases the classic BFT scalability trade-off, where it provides high performance in small networks, but significant degradation as the number of validators grows. With an increasing number of validators, the volume of voting messages increases quadratically since Tendermint's protocol has two stages of votes per block. The quadratic message complexity makes it less suitable for large validator sets (with more than 32 nodes), where throughput becomes constrained by network bandwidth rather than computational resources. In terms of performance metrics, Tendermint was relatively light. Even at peak throughput, the CPU stayed below 70% on average, and memory usage was under 0.4 GB per node. Tendermint is an excellent choice for medium-sized networks that require fast finality and Byzantine fault tolerance. For larger networks, it is essential to be aware of the throughput decline. Results show very high throughput at small node counts, but a significant decrease at 32 nodes. Latency remained very low, of tens of milliseconds, even at a 32-node network size.

5.3. Ethereum Clique

Clique is an Ethereum consensus protocol implementing Proof-of-Authority (PoA). In the PoA network, a fixed set of trusted validators takes turns producing blocks. Clique assumes validators are honest, as it provides immediate sealing of blocks by a single validator's signature, without requiring all validators to vote on every block. With only a maximum of 7 validator nodes allowed by design, any additional non-validator nodes do not participate in consensus. This means that the consensus load in Clique did not increase after seven nodes were added; adding more nodes primarily resulted in increased networking overhead, but not in more signatures or voting. This design prioritises performance over decentralisation, as shown in Figure 5.3.

- **Throughput:** Clique showed high throughput and relatively graceful degradation, compared to the other two systems. Clique processed ∼2296 TPS at 4 nodes, decreasing to ∼1352 TPS at 32 nodes (∼41% drop). The reason for this strong scalability is that the bottleneck in Clique is primarily block propagation, not consensus voting, due to its instant finality. Since only up to 7 validators sign blocks in a round-robin fashion, the act of producing a block remains a constant workload, regardless of the total number of nodes. More nodes mean more network messages to deliver each block. The throughput decline (2296 -> 1352 TPS) likely comes from slightly increased propagation delay.
- **Latency:** Clique's Latency was low and increased slightly with more nodes. Latency in Clique was ∼21 ms at four nodes, up to ∼36 ms at 32 nodes. Propagation delays in reaching all nodes could again explain the slight rise with more nodes.

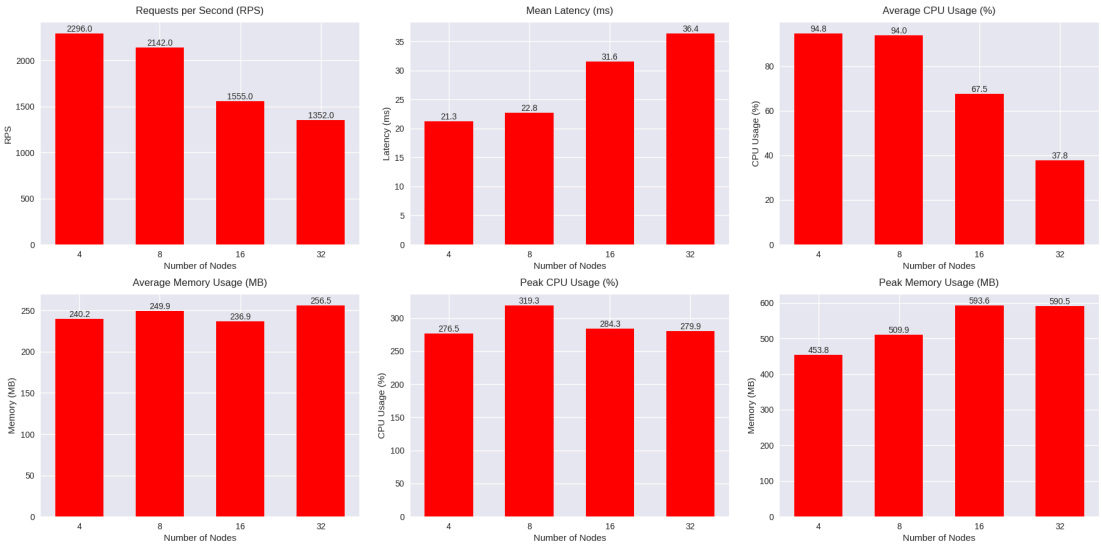

Figure 5.3.: Clique performance metrics across network scales (4-32 nodes)

- **CPU:** Clique's CPU usage was significantly higher than that of the other systems in some cases, particularly at smaller node counts; however, the average CPU usage decreased with an increasing number of nodes in the network. At 4 and 8 nodes, the average CPU was ∼95%, meaning those nodes were maxing out their cores. By 16 nodes (7 validators, 9 non-validators), average CPU dropped to ∼68%, and at 32 nodes (7 validators, 25 non-validators), it was ∼38%. This dramatic drop is because in a 32-node network with 7 validators, the 25 non-validator nodes do not perform mining work; they verify blocks, which is much less CPU-intensive. Thus, when averaging, the busy validators (using up to ∼320% CPU each at peak) plus many idle nodes, on average, yield a lower value. The validators themselves were working hard – the protocol demands continuous block production, which on Geth uses CPU to execute transactions quickly. This can be observed from the peak CPU usage graph.

- **Memory:** Memory usage remained flat and low, without heavy jumps. Clique nodes used ∼240 MB on average at four nodes, and about ∼256 MB at 32 nodes. This is expected because the base Geth node doesn't require a lot of memory for a short-running test, as there is no significant chain buildup over time.

Ethereum's Clique PoA consensus delivered consistently high performance and scaled gracefully across our tested range. Clique's consensus involves one round of communication, unlike BFT consensus, which requires multiple rounds. Clique with a fixed ∼7 validators sees minimal change in consensus overhead as total nodes grow. This is why throughput and latency degrade only gently, primarily due to the increase in the number of non-validator nodes, rather than due to heavier consensus steps. We should interpret Clique's results carefully, due to its inability to scale beyond seven validator nodes. Clique (PoA) is ideal for scenarios where participants are known and trusted, and where throughput is a key requirement. Examples include supply chain networks or enterprise private blockchain ledgers. Clique provided the best combination of high throughput and stable scaling in our experiments. However, the main reason for that is Clique's design, which relies on a fixed, small set of validator nodes.

5.4. Comparative Evaluation and Insights

Our experimental results (Figure 5.4) reveal fundamental trade-offs between consensus mechanisms in permissioned blockchain systems. We analyse these findings through four critical dimensions:

- **Throughput Comparison:** Tendermint had the highest raw throughput in small networks, but Clique had the highest throughput at the largest network size (32 nodes) among the three. This implies that Tendermint excels in throughput until BFT overhead takes effect, whereas Clique offers more stable throughput across all network sizes. The gap between Clique and Tendermint at 32 nodes reflects the

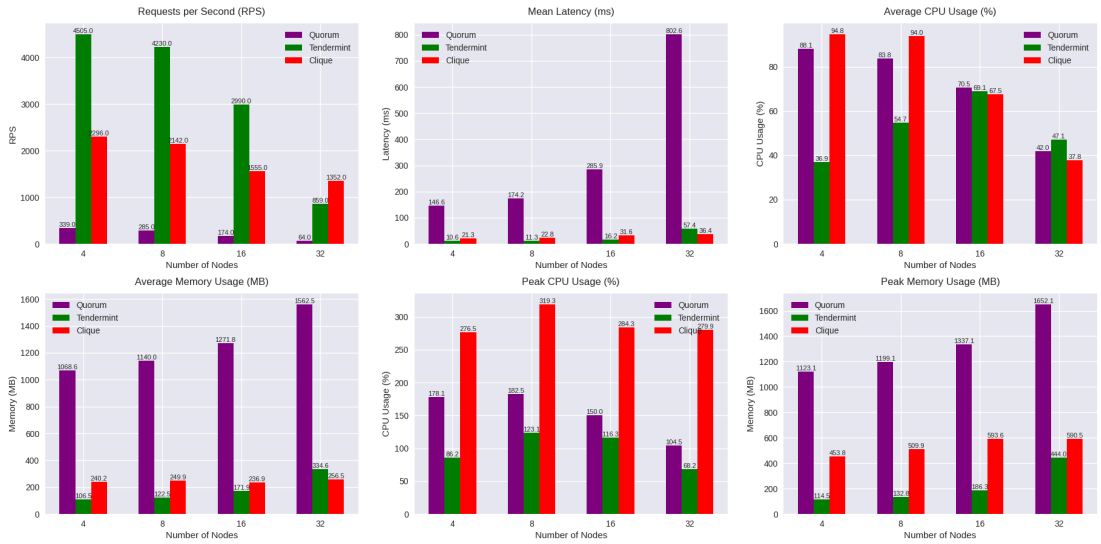

Figure 5.4.: Comparative performance metrics of Quorum (Raft), Tendermint (BFT), and Ethereum Clique (PoA) across network scales (4-32 nodes)

extent to which Tendermint has lost due to scalability limitations. Quorum trailed far behind in all cases.

- **Latency Comparison:** Tendermint consistently had the lowest latency, followed closely by Clique. At 32 nodes, 57 ms (Tendermint) vs 36 ms (Clique) are both very low and might not be practically distinguishable in many applications. Quorum's 802 ms is again significantly lagging. If low latency is a critical requirement, Tendermint or Clique are preferable, with Quorum being unsuitable beyond small networks.

- **Scalability and Efficiency:** Clique demonstrated the best scalability (slight performance loss when scaling up), Tendermint moderate scalability, and Quorum the worst. Efficiency-wise, Tendermint and Clique used far less memory than Quorum for the same task, making them more feasible on commodity hardware for large networks. When it comes to CPU efficiency, Quorum and Clique may utilise hardware resources extensively in smaller setups, whereas Tendermint uses them moderately. However, at the larger scale of the network, all three systems tend to use on average a similarly moderate amount of CPU resources.

- **Consensus Algorithm Impact:** The choice of consensus algorithm involves fundamental architectural compromises between trust assumptions, fault tolerance, and performance scalability.
  - **Raft (Quorum) – Great for trust, bad for scale:** It's slow compared to the other two consensus mechanisms, non-scalable and not Byzantine tolerant. Use when all parties are in one organisation and the node count is small.

- **BFT (Tendermint) – Fast and secure until it hits complexity limits in large networks:** It can combine speed and security up to a moderate network size. Use for multi-party scenarios that require security against some corrupt actors, as long as the consortium isn't too large. Each added validator increases overhead, so it's essential to find a balance.
- **PoA (Clique) – High performance through centralisation:** It scales well in performance, but doesn't grow beyond seven validator nodes. Use when a degree of central trust is acceptable or desired (e.g., enterprise or government networks), to get the highest throughput and simplicity. Be aware of the limited governance nodes.

5.5. Choosing the Right Blockchain

The decision of which platform to use should be made by matching the blockchain's consensus to the use case. Based on our experimental results, we propose the following decision framework:

- If the network is **small (4-5 nodes)** and likely to remain so, Tendermint could be a workable solution, given its high throughput, small latency and moderate resource consumption. Clique can also provide a similar experience.
- If the network is **medium (5–20 validators)** and members are mutually untrusted or require a high degree of security, Tendermint is a good option. It provides both performance and BFT security.
- If the network is **large (20+ nodes)** but one can compromise on trust, then Clique is a strong candidate as it will scale with far less performance loss. Alternatively, one could consider Tendermint if BFT is needed at a large scale.

These recommendations are limited to the three evaluated systems within our test parameters. The optimal choice ultimately depends on the specific balance of performance requirements, trust assumptions, and fault tolerance needs in each deployment scenario.

5.6. Experimental Setup and Environment

This section provides a comprehensive guide to replicating the benchmarking environment, explaining both the purpose of each component and the rationale behind configuration choices.

5.6.1. Hardware Requirements

The hardware specifications were determined through iterative testing to ensure stable operation during peak network loads:

- **Virtual Machine Configuration**:

  - `Instance Type`: Azure Standard D16ads v5 - Selected for its balanced compute-to-memory ratio (16 vCPUs with 64 GiB RAM)
  - `Memory`: 64 GiB DDR4 - Required to prevent OOM (Out-of-Memory) errors during Quorum's memory heavy network operation
  - `Storage`: 30 GiB SSD - Accommodates Docker images and blockchain ledger growth during testing
- **Operating System**: Ubuntu 22.04 LTS was chosen for its simplicity and native Docker compatibility

Performance Note: During stress testing, Quorum required $\geq$ 32 GB RAM for 32-node networks due to its Raft consensus implementation maintaining large in-memory transaction logs. Tendermint and Clique showed better memory efficiency, operating successfully on a machine with 16 GB RAM.

5.6.2. Remote Access Configuration

The remote desktop setup enables secure GUI access for monitoring and launching benchmarking tests:

- `xRDP Service`: Lightweight remote desktop protocol server for Linux
- `XFCE Desktop`: Provides a responsive GUI environment without excessive resource usage

5.6.3. Software Stack Installation

System Foundations

Essential packages for development environments:

- `build-essential`: Contains GCC compiler and make utility for building native dependencies
- `git`: Version control system for cloning and managing the benchmark codebase
- `curl/wget`: Tools for downloading remote packages securely

Containerization Layer

Critical components for blockchain network orchestration:

- **Docker Engine**:
  - Provides isolated environments for each blockchain node
  - Enables reproducible network configurations
  - Manages resource allocation through cgroups

- **Docker Compose**:
  - Coordinates multi-container deployments
  - Simplifies scaling node counts
  - Manages container dependencies

Blockchain-Specific Dependencies

Specialised tools required by individual platforms:

- **Go 1.22.3**:
  - Compiles Tendermint core components
  - Required for ABCI (Application Blockchain Interface) implementations
  - Version-pinned for compatibility
- **Node.js 18.x**:
  - Runs the benchmarking control plane
  - Executes transaction workload generators
  - LTS version ensures stability

5.6.4. Application Deployment

1. **Repository Cloning**: Obtains latest framework code
2. **Dependency Installation**: Resolves JavaScript packages and sets up development environment
3. **Environment Configuration**: Customizes network parameters

5.6.5. Validation Procedures

Once all necessary packages are installed and environments are configured, the user can start executing various tests on blockchain systems. The tool already has a script for installing all of the packages mentioned above. However, many factors could prevent it from completing successfully, and users are advised to execute each step manually.

Key Troubleshooting Scenarios:

- `Error: Cannot connect to Docker daemon`:
  - Verify user is in docker group: `groups $USER`
  - Check daemon status: `sudo systemctl status docker`
- `Port conflicts during deployment`:
  - Identify used ports: `sudo netstat -tulnp`

– Adjust in `.env` or docker-compose.yml

This comprehensive setup provides a controlled environment for reproducible blockchain benchmarking, with each component carefully selected to ensure valid performance comparisons across platforms. The resulting data from the experimental execution for all supported blockchain systems, with all network sizes, can be found in Appendix A.1.

6. Framework Extension Guide

This chapter provides a systematic methodology for extending the benchmarking framework to support additional permissioned blockchain systems. The intended users fall into three primary categories:

- **Blockchain Researchers**: Academic investigators conducting comparative studies of consensus mechanisms in permissioned environments
- **Distributed Systems Engineers**: Professionals evaluating blockchain platforms for enterprise deployment scenarios
- **Protocol Developers**: Teams implementing novel consensus algorithms requiring performance validation against established baselines

Using Hyperledger Besu as a case study, we demonstrate the four-phase extension process that maintains the framework's modular design while ensuring consistent performance measurement across platforms. The integration of new blockchain systems follows a standardised workflow:

1. Network deployment script preparation
2. Configuration file updates
3. Load test handler implementation (Ethereum-based platforms can reuse the existing method)
4. Validation and verification

6.1. Phase 1: Network Deployment Preparation

Each blockchain has its implementation for setting up a network. The user needs to provide a way to launch and tear down a multi-node network using Docker automatically. Docker ensures consistency in the environment setup. Often, blockchain frameworks already provide an example network setup using Docker, and even if not, the example network can be easily converted to support and run on Docker. Once the Docker Compose files for starting blockchain networks with 4, 8, 16, and 32 node containers are set up, we need to write a Makefile that uses these Docker Compose files to start and stop the networks. Listing 6.1 demonstrates an example of the Makefile:

```
localnet-start-4:
    echo "Starting the network with 4 nodes"
    docker-compose -f docker-compose-besu-4.yml up -d

localnet-start-8:
    echo "Starting the network with 8 nodes"
    docker-compose -f docker-compose-besu-8.yml up -d

localnet-stop:
    docker-compose down
    rm -rf ./data
    docker system prune -f
```

Listing 6.1: Example Makefile for Hyperledger Besu

The **localnet-start-4** command uses a Besu-specific **docker-compose-besu-4.yml** file to launch a 4-node network, while the **localnet-stop** command brings down the network and cleans up data directories. Removing data ensures a fresh state for the following tests.

6.2. Phase 2: Configuration Specification

The framework uses a JSON configuration file to define how to handle each supported blockchain. The `blockchain_config.json` file serves as the integration point for new systems, requiring:

- **name:** A Blockchain system name.
- **nodes:** Network size.
- **startCommand:** The shell command to start the network.
- **stopCommand:** Command to stop the network and clean up.
- **endpoint:** The HTTP endpoint where the blockchain listens for transactions
- **requestBody:** A template of a transaction that will be sent for load testing.

For Hyperledger Besu, Listing 6.2 illustrates an example configuration for a 4-node network:

```
{
    "name": "Besu",
    "nodes": 4,
    "startCommand": "cd ./besu && make localnet-start-4",
    "stopCommand": "cd ./besu && make localnet-stop",
    "endpoint": "http://localhost:8545",
    "requestBody": {
        "jsonrpc": "2.0",
        "method": "eth_sendTransaction",
        "params": [{
```

```
            "from": "0x...",
            "to": "0x...",
            "value": "0x9184e72a"
        }]
    }
}
```

Listing 6.2: Cofiguration for Hyperledger Besu for 4 nodes

After this step, the frontend will automatically list "Besu (4 nodes)" as an option to run tests on when calling GET /networks.

6.3. Phase 3: Load Test Implementation

Our backend has a generic runLoadTest function (using the loadtest library) for HTTP endpoints. If the new blockchain is Ethereum-based, we can reuse the existing function. If not, we implement a custom method. Since Hyperledger Besu is Ethereum-based, the same logic we used for Quorum can be applied. If the new blockchain had a different interface, we could implement a *runCustomLoadTest()* method that looks as follows (Listing 6.3):

```
async runCustomLoadTest(blockchainConfig) {
    return new Promise((resolve, reject) => {
        const options = {
            url: blockchainConfig.endpoint,
            maxRequests: 10000,
            concurrency: 50,
            method: 'POST',
            body: JSON.stringify(blockchainConfig.requestBody),
            headers: { 'Content-Type': 'application/json' }
        };

        loadtest.loadTest(options, (error, result) => {
            if (error) reject(error);
            else resolve(result);
        });
    });
}
```

Listing 6.3: Method for loadtesting non-Ethereum based blockchain system

This skeleton utilises the same loadtest library as in other methods, but you can insert additional logic for the transaction body. Once the method is implemented, we need to make a small update to the `app.post()` method implementation.

```
app.post('/blockchain', async (req, res) => {
    const { blockchainName } = req.body;
    let result;

    if (blockchainName === "Besu" || blockchainName === "Quorum") {
        result = await runLoadTest(blockchainConfig);
    } else if (blockchainName === "CustomChain") {
```

```
        result = await runCustomLoadTest(blockchainConfig);
    }
    // ...
});
```

This pseudocode shows adding a simple condition for our new chain. Because Besu can be handled by runLoadTest, we don't even need a new function. For demonstration purposes, we included an example of how one would handle a hypothetical "CustomChain" that might not be Ethereum-based. The implementation of the transaction load test differs significantly between different blockchain systems that are not based on Ethereum. Therefore, it is not possible to create a common method for all potential blockchain systems, and this implementation relies entirely on the user's ability to implement that method into the framework's workflow. At this point, the backend knows how to start/stop the network and generate load for Besu. We have effectively extended the core benchmarking loop to support Hyperledger Besu, without altering the core logic of the framework or how metrics are collected; this remains consistent across all supported blockchain systems.

6.4. Phase 4: Validation Protocol

After implementing the steps above, it is essential to verify every step of the testing process to ensure it works as expected. This involves:

1. **Check the Configuration File**: Use the existing **GET /networks** endpoint to ensure the new blockchain network entries appear correctly in the UI.
2. **Launch the Test Network**: Before running tests and capturing metrics, verify that the created Makefiles work properly. For instance, run cd ./besu make localnet-start-4 in a terminal. Then, check with **docker ps** command to see if 4 Besu containers are running. Common issues:
   - **Port conflicts:** Often, the ports in Docker are not exposed correctly, preventing the backend from interacting with the network. It's best to check that there are no conflicts by manually checking if the nodes can be reached via their endpoints.
   - **Data directories/Permissions:** Double check if the **localnet-stop** command worked as expected and cleaned up the created data by the network. It may require admin rights, as it was with Tendermint.
3. **Run a Test via the Frontend or API:** Use the UI to select the new blockchain and start a test. Monitor the backend logs to verify that the network was successfully started and that the blockchain network is outputting transaction or chain-related logs.
4. **Metrics verification:** Ensure that the collected metrics don't deviate significantly from other blockchain frameworks, and always consider the initial run as a test run, and it's recommended to discard it, since at the first run, many packages and files need to be downloaded by Docker.

To recap, adding a new blockchain requires understanding the blockchain system's architecture and adapting its load test method implementation. The framework was designed to be scalable and provide flexible integration of additional blockchain systems. By reducing the effort required to integrate new systems, the framework can evolve with the blockchain landscape, allowing for continuous comparison and evaluation as new platforms emerge. This ensures the framework's long-term relevance in both research and industry for permissioned blockchain systems.

7. Conclusion

Permissioned blockchain systems have emerged as critical infrastructure across various industries, providing secure and transparent data-sharing solutions. However, the numerous platforms and consensus mechanisms have made it challenging to choose the right system. This Master's Thesis addresses this challenge through the development of an extensible benchmarking framework that automates network deployment, load testing, and performance metric collection, significantly reducing manual effort in comparative evaluations.

Permissioned blockchains utilise a range of consensus mechanisms, each with its inherent strengths and weaknesses. Our experimental results quantitatively validate fundamental trade-offs in consensus mechanisms, such as the performance-centralisation compromise of Proof-of-Authority, the security-resource demand balance of Practical Byzantine Fault Tolerance, and the simplicity-scalability limitations of Raft.

Our framework addresses key limitations in existing tools (BlockBench, BCTMark, BlockCompass), which often focus narrowly on specific platforms, lack support for newer blockchain versions or provide inadequate extension mechanisms.

Key framework innovations include:

- Modular architecture supporting diverse blockchain systems
- Comprehensive automation of the benchmarking pipeline
- Intuitive user interface for non-technical users
- Clear extension guidelines for new platforms

7.1. Future work

While our evaluation covered three consensus algorithms (Raft, Tendermint BFT, Clique PoA), many others remain untested. Consequently, the findings presented in this Master's Thesis should be interpreted with caution. A natural continuation would be to integrate and benchmark a broader set of systems, such as Hyperledger Fabric, Hyperledger Besu, and R3 Corda. Doing so will strengthen the conclusions and perhaps reveal new insights. One of the key motivations for this thesis was the lack of clear documentation in existing benchmarking tools regarding their extensibility to support additional blockchain platforms. This gap was addressed through the inclusion of a comprehensive extension guide, presented in Chapter 6. Future versions of the framework could further reduce the technical barrier by incorporating features such as an interactive wizard for adding new blockchains or supplementary resources like video tutorials. Given the rapidly evolving

nature of the blockchain domain, with new consensus mechanisms continually emerging, continuous maintenance will be required to keep the framework relevant. This includes not only supporting additional blockchain systems but also keeping existing integrations up to date with their latest versions, which frequently introduce performance enhancements.

7.2. Final Thoughts

This Master's thesis introduces a modern benchmarking framework designed to enable the systematic evaluation of multiple permissioned blockchain systems in a side-by-side manner. The framework's extensibility and automation ensure its continued relevance as the underlying technologies evolve. Given the growing role of permissioned blockchains in enterprise applications, such tools are essential for supporting their development by offering rigorous, empirical performance data. The experimental results yield several practical insights—for instance, Clique (Proof of Authority) and Tendermint demonstrate the ability to achieve high throughput, whereas Quorum (Raft) exhibits comparatively lower performance. These findings underscore the importance of carefully selecting a consensus mechanism based on specific application requirements. Overall, the proposed framework serves as a valuable decision-support tool in a landscape where no single blockchain solution is suitable for all use cases. By providing clear, extensible, and up-to-date benchmarking data, the framework helps organisations make informed design choices and avoid potentially costly implementation errors.

Bibliography

[ABB+18a] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[ABB+18b] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[AEVL16] A Azaria, A Ekblaw, T Vieira, and A Lippman. Using blockchain for medical data access and permission management. In *2nd International Conference on Open and Big Data (OBD)*, pages 1–2, 2016.

[AMQ13] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: redundant Byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306. IEEE, 2013.

[B+13] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1(22-23):5–7, 2013.

[BCC+19] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State machine replication in the Libra blockchain. *The Libra Association, Tech. Rep.*, 7, 2019.

[CDE+16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains: a position paper. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[CGT19] Mauro Conti, Ankit Gangwal, and Michele Todero. Blockchain trilemma solver Algorand has dilemma over undecidable messages. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–8, 2019.

[Cha16] J. M. Chase. Quorum whitepaper. https://github.com/jpmorganchase/quorum-docs, 2016. Online; accessed: 2023-12-01.

[CKR12] Nikos Chondros, Konstantinos Kokordelis, and Mema Roussopoulos. On the practicality of practical Byzantine fault tolerance. In *Middleware 2012: ACM/IFIP/USENIX 13th International Middleware Conference*, pages 436–455. Springer, 2012.

[CL+99] Miguel Castro, Barbara Liskov, et al. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[Cli17] Clique proof-of-authority consensus protocol. https://eips.ethereum.org/EIPS/eip-225, 2017.

[Cor16] Intel Corporation. Proof of elapsed time (PoET). https://sawtooth.hyperledger.org/docs/1.2/sysadmin_guide/about_dynamic_consensus.html#poet-consensus-label, 2016. Accessed: 2023-12-01.

[DKTA20] Mohammad Dabbagh, Mohsen Kakavand, Mohammad Tahir, and Angela Amphawan. Performance analysis of blockchain platforms: empirical evaluation of Hyperledger Fabric and Ethereum. In *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, pages 1–6. IEEE, 2020.

[DWC+17] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: a framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1085–1100, 2017.

[FLKM22] Caixiang Fan, Changyuan Lin, Hamzeh Khazaei, and Petr Musilek. Performance analysis of Hyperledger Besu in private blockchain. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 64–73. IEEE, 2022.

[Fou20] OpenEthereum Foundation. Aura - authority round - wiki. https://openethereum.github.io/Aura.html, 08 2020.

[GHW24] Dominic Grandjean, Lioba Heimbach, and Roger Wattenhofer. Ethereum proof-of-stake consensus layer: participation and decentralization. In *International Conference on Financial Cryptography and Data Security*, pages 253–280. Springer, 2024.

[HB16] Mike Hearn and Richard Gendal Brown. Corda: a distributed ledger. https://docs.r3.com/whitepapers/corda-technical-whitepaper.pdf, 2016. Corda technical white paper.

[KB19] Jae Kwon and Ethan Buchman. Cosmos whitepaper. *A Netw. Distrib. Ledgers*, 27:1–32, 2019.

[KHD17] Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. Digital supply chain transformation toward blockchain integration. 2017.

[Kwo14] Jae Kwon. Tendermint: consensus without mining. *Draft v. 0.6, fall*, 1(11):1–11, 2014.

[LRP20] Stefanos Leonardos, Daniel Reijsbergen, and Georgios Piliouras. Presto: a systematic framework for blockchain consensus protocols. *IEEE Transactions on Engineering Management*, 67(4):1028–1044, 2020.

[MCDN22] Marco Mazzoni, Antonio Corradi, and Vincenzo Di Nicola. Performance evaluation of permissioned blockchains for financial applications: the ConsenSys Quorum case study. *Blockchain: Research and Applications*, 3(1):100026, 2022.

[MXC+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, 2016.

[Nak08] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system. *Decentralized business review*, 2008.

[NBF+16] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. Bitcoin and cryptocurrency technologies, 2016.

[OM14] Karl J O'Dwyer and David Malone. Bitcoin mining and its energy footprint. 2014.

[OO14] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, 2014.

[PRLT21] Julien Polge, Jérémy Robert, and Yves Le Traon. Permissioned blockchain frameworks in the industry: a comparison. *ICT Express*, 7(2):229–233, 2021.

[RHF23] Mohammadreza Rasolroveicy, Wejdene Haouari, and Marios Fokaefs. BlockCompass: a benchmarking platform for blockchain performance. 2023.

[Sawst] Hyperledger Sawtooth. Introduction, latest. [Online; accessed: 01-12-2023].

[Sec18] A Secure. The Zilliqa project: a secure, scalable blockchain platform. 2018.

[SGVGO21] Cyril Naves Samuel, Severine Glock, François Verdier, and Patricia Guitton-Ouhamou. Choice of Ethereum clients for private blockchain: assessment from proof-of-authority perspective. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5. IEEE, 2021.

[She22] Zongda She. Vechain: a renovation of supply chain management—a look into its organization, current activity, and prospect. In *Proceedings of the 2022 International Conference on Educational Informatization, E-commerce and Information System*, pages 29–30, 2022.

[SLM20] Dimitri Saingre, Thomas Ledoux, and Jean-Marc Menaud. BCTMark: a framework for benchmarking blockchain technologies. In *AICCSA 2020 - 17th IEEE/ACS International Conference on Computer Systems and Applications*, pages 1–8, Antalya, Turkey, Nov 2020.

[Tas19] John Taskinsoy. Is Facebook's Libra project already a miscarriage? *Available at SSRN*, 2019.

[YMR+18] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069*, 2018.

[ZGW+19] Jiashuo Zhang, Jianbo Gao, Zhenhao Wu, Wentian Yan, Qize Wo, Qingshan Li, and Zhong Chen. Performance analysis of the Libra blockchain: an experimental study. In *2019 2nd International Conference on Hot Information-Centric Networking (HotICN)*, pages 77–83. IEEE, 2019.

A. Appendix

A.1. Experiment Output Data

The following JSON contains the raw output of performance experiments for Quorum, Tendermint, and Clique blockchains:


```
{
  "data": [
    {
      "id": "cbe7306f-3522-49a7-ae7d-fe8990fd0642",
      "blockchainName": "Quorum",
      "numberOfNodes": 32,
      "timestamp": "2025-02-01T17:55:38.106Z",
      "totalErrors": 0,
      "rps": 64,
      "meanLatencyMs": 802.64,
      "averageCpu": 41.98,
      "averageMemory": 1562.53,
      "peakCpu": 104.54,
      "peakMemory": 1652.12
    },
    {
      "id": "b0ff17ff-6348-4f11-8d93-a53416fcc543",
      "blockchainName": "Quorum",
      "numberOfNodes": 16,
      "timestamp": "2025-02-01T19:26:04.199Z",
      "totalErrors": 0,
      "rps": 174,
      "meanLatencyMs": 285.9,
      "averageCpu": 70.48,
      "averageMemory": 1271.84,
      "peakCpu": 150.01,
      "peakMemory": 1337.14
    },
    {
      "id": "d4caba86-d89c-41d6-b5b1-c9aa9e62e139",
      "blockchainName": "Quorum",
      "numberOfNodes": 8,
```

```
    "timestamp": "2025-02-01T20:19:47.161Z",
    "totalErrors": 0,
    "rps": 285,
    "meanLatencyMs": 174.24,
    "averageCpu": 83.75,
    "averageMemory": 1139.95,
    "peakCpu": 182.53,
    "peakMemory": 1199.1
  },
  {
    "id": "7eb09963-280e-4fcf-9648-6c55ae41c80b",
    "blockchainName": "Quorum",
    "numberOfNodes": 4,
    "timestamp": "2025-02-01T20:57:55.835Z",
    "totalErrors": 0,
    "rps": 339,
    "meanLatencyMs": 146.62,
    "averageCpu": 88.06,
    "averageMemory": 1068.57,
    "peakCpu": 178.05,
    "peakMemory": 1123.12
  },
  {
    "id": "c52b934a-cdc4-4923-8334-ceafa9d0c23e",
    "blockchainName": "Tendermint",
    "numberOfNodes": 4,
    "timestamp": "2025-02-01T21:06:57.087Z",
    "totalErrors": 0,
    "rps": 4505,
    "meanLatencyMs": 10.6,
    "averageCpu": 36.87,
    "averageMemory": 106.52,
    "peakCpu": 86.16,
    "peakMemory": 114.51
  },
  {
    "id": "0ae9d382-396d-4706-bf98-d6137ee6770a",
    "blockchainName": "Tendermint",
    "numberOfNodes": 8,
    "timestamp": "2025-02-01T21:15:48.157Z",
    "totalErrors": 0,
    "rps": 4230,
    "meanLatencyMs": 11.32,
```

```
  "averageCpu": 54.66,
  "averageMemory": 122.47,
  "peakCpu": 123.13,
  "peakMemory": 132.78
},
{
  "id": "56fd72fb-fcab-4ae2-b633-2ef93e9e06ac",
  "blockchainName": "Tendermint",
  "numberOfNodes": 16,
  "timestamp": "2025-02-01T21:23:28.699Z",
  "totalErrors": 0,
  "rps": 2990,
  "meanLatencyMs": 16.18,
  "averageCpu": 69.08,
  "averageMemory": 171.91,
  "peakCpu": 116.35,
  "peakMemory": 186.28
},
{
  "id": "7ededb18-f04e-4cf7-a987-0032ce2fdf12",
  "blockchainName": "Tendermint",
  "numberOfNodes": 32,
  "timestamp": "2025-02-01T21:34:19.872Z",
  "totalErrors": 0,
  "rps": 859,
  "meanLatencyMs": 57.42,
  "averageCpu": 47.15,
  "averageMemory": 334.55,
  "peakCpu": 68.22,
  "peakMemory": 444.04
},
{
  "id": "b0a48131-0fe4-4525-9c52-66b56cc7537f",
  "blockchainName": "Clique",
  "numberOfNodes": 4,
  "timestamp": "2025-02-01T21:45:31.519Z",
  "totalErrors": 0,
  "rps": 2296,
  "meanLatencyMs": 21.26,
  "averageCpu": 94.8,
  "averageMemory": 240.24,
  "peakCpu": 276.53,
  "peakMemory": 453.76
```

```
    },
    {
      "id": "01f8a850-09d2-40f8-89c1-0e1710aa6403",
      "blockchainName": "Clique",
      "numberOfNodes": 8,
      "timestamp": "2025-02-01T21:57:23.593Z",
      "totalErrors": 0,
      "rps": 2142,
      "meanLatencyMs": 22.76,
      "averageCpu": 94,
      "averageMemory": 249.92,
      "peakCpu": 319.3,
      "peakMemory": 509.94
    },
    {
      "id": "7d11704c-93fe-4e81-8d55-48c5f5be7227",
      "blockchainName": "Clique",
      "numberOfNodes": 16,
      "timestamp": "2025-02-01T22:05:12.741Z",
      "totalErrors": 0,
      "rps": 1555,
      "meanLatencyMs": 31.58,
      "averageCpu": 67.54,
      "averageMemory": 236.88,
      "peakCpu": 284.28,
      "peakMemory": 593.62
    },
    {
      "id": "ac99b861-3d14-42d3-afe4-1b6a6d9df0ef",
      "blockchainName": "Clique",
      "numberOfNodes": 32,
      "timestamp": "2025-02-01T22:12:29.669Z",
      "totalErrors": 0,
      "rps": 1352,
      "meanLatencyMs": 36.42,
      "averageCpu": 37.81,
      "averageMemory": 256.52,
      "peakCpu": 279.92,
      "peakMemory": 590.54
    }
  ]
}
```