Scientific Research Publishing

Journal of Computer and Communications, 2025, 13(7), 204-222
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Investigating Suitable Consensus Protocols for Secured Blockchain Based System

Velucian I. Fabian, Estha Kazinja, Othmar O. Mwambe, Nizetha D. Kimario, Isakwisa G. Tende

Computer Studies Department, Dar es Salaam Institute of Technology, Dar es Salaam, Tanzania
Email: nb19107@shibaura-it.ac.jp

How to cite this paper: Fabian, V.I., Kazinja, E., Mwambe, O.O., Kimario, N.D. and Tende, I.G. (2025) Investigating Suitable Consensus Protocols for Secured Blockchain Based System. *Journal of Computer and Communications*, **13**, 204-222.
https://doi.org/10.4236/jcc.2025.137010

Received: June 12, 2025
Accepted: July 21, 2025
Published: July 24, 2025

Abstract

In an attempt to address the challenge of selecting the most optimal ordering service for achieving high throughput and low latency in permissioned Hyperledger Fabric production networks, this evaluates the characteristics and suitability of the three primary consensus protocols: Kafka, Raft, and Solo. This evaluation examines their fault tolerance mechanisms, deployment models, architectural properties (decentralization, scalability), and intended use cases. The obtained results suggest that Solo, as a single-node protocol lacking fault tolerance and decentralization, is suitable only for development and testing environments and not for production. Kafka, relying on an external cluster and offering crash fault-tolerance, is reliable but is indicated to have scalability and flexibility limitations in larger networks. Raft, also providing crash fault-tolerance and operating internally within the network, demonstrates improved decentralization and scalability, positioning it as the preferred choice for many production deployments, although its performance can be variable depending on network size. The results highlight an ongoing debate regarding the best balance between throughput and latency among these protocols, indicating that each presents trade-off. Therefore, the analysis suggests that determining the most optimal protocol for a specific production environment requires further, tailored evaluation based on its unique requirements and scale.

Keywords

Consensus Protocols, Secured Blockchain, Blockchain Based System

1. Introduction

The evolving of Blockchain technology brought a significant transformative innovation in network and system transaction by offering a decentralized and secure

method for managing transactions and data [1] [2]. In this regard, Hyperledger fabric, a private or permissioned blockchain framework, plays a big role by enabling private and controlled networks communications (Uddin *et al.*, 2021) [3] [4].

Participants are being authenticated before performing any transaction. To achieve this authentication, consensus protocols are essential components in any blockchain network, they ensure that all participants agree on the order and they check the validity of intended transactions (Nasrulin *et al.*, 2022 [5]).

The primary consensus protocols utilized in Hyperledger Fabric are Kafka, Raft and Solo, each of them provides a unique mechanism for transaction ordering (Petrescu & Petrescu, 2020 [6]). Therefore, selecting the most suitable consensus protocol to use in production environment that balances throughput, latency, decentralization and scalability factors, remains a critical challenge for network and system designers (G. Yang *et al.*, 2022 [2], Y. Hao, *et al.* 2018 [3]). Kafka, a crash fault-tolerant protocol, relies on external Kafka cluster to maintain consensus, but it is embedded with some limitations in flexibility and scalability (Wang, C. *et al.* 2020 [7]). Raft is also a crash fault tolerant consensus protocol operating within a blockchain network that offers better decentralization and scalability, making many blockchain system designers opt for it in production environment. Solo in contrast is a single node protocol designed for development and testing purposes, it is regarded as a lacking robustness in live environment (Huang *et al.*, 2020 [8]).

Considering these differences, there is ongoing consensus protocol selection debate among blockchain practitioners (Nguyen, M. *et al.* 2021 [9]), the debate focuses on identifying which consensus protocol gives ideal balance between performance and fault-tolerance in varying environment settings, like network sizes, large volume of transactions, the protocol that can process a big volume of transaction at a short time without crashing the system.

This paper aims at investigating the strengths and limitations of Kafka, Raft and Solo ordering service as consensus protocols in Hyperledger Fabric.

The study focuses on investigating Hyperledger consensus protocol capabilities in securing blockchain-base information systems, through a detailed analysis that was carried out in Hyperledger Fabric system termed as Medical Provision Information System. The system was purposely developed to identify the most suitable consensus protocol for production environments that process many transactions and need a short time to finish them. These need high throughput, and low latency while scaling the network.

2. Background Information and Related Works

Blockchain technology has experienced significant advancements since its beginnings, evolving from centralized systems to decentralized platforms like Bitcoin and Ethereum, and more recently, to specialized private blockchain networks tailored for enterprise applications (Hao *et al.*, 2018 [3]). These modern networks

prioritize enhanced security, scalability, and customization, with Hyperledger Fabric emerging as a leading framework (Munasinghe & Halgamuge, 2023 [1]).

Developed under the Linux Foundation, Hyperledger Fabric features a permissioned blockchain architecture that allows organizations to create secure and immutable networks. In this setup, participant authentication is facilitated through public keys managed by a Membership Service Provider (MSP), while a trusted Certificate Authority (CA) oversees registration. A key element of this architecture is the consensus mechanism or ordering service, which maintains network-wide agreement on the validation and sequencing of transactions after they are endorsed by peer nodes.

Hyperledger Fabric supports various consensus protocols, each with its own benefits and limitations. Kafka is noted for its crash fault tolerance and high throughput, making it suitable for smaller networks, but it faces scalability challenges due to its reliance on external infrastructure. Raft, an internal consensus protocol, offers improved scalability and fault resilience, making it ideal for production environments with higher transaction volumes, despite a slight decrease in throughput compared to Kafka. Solo, a simplistic, single-node protocol, is mainly intended for development and testing but is not suitable for real-world production due to its lack of decentralization and fault tolerance (Huang *et al.*, 2020, Petrescu & Petrescu, 2020 [6]).

Research has highlighted these trade-offs, examining how the choice of protocol impacts performance metrics such as throughput, latency, fault tolerance, and scalability. While Kafka performs well in controlled settings, it struggles with larger networks. Raft provides a robust solution for scalability, although it introduces complexity, and Solo serves as a lightweight option for developers (Chandrakant, 2024 [10]).

A significant challenge remains in the absence of clear guidelines for protocol selection, especially in critical areas where performance and security are crucial, such as healthcare. Ensuring data integrity, low latency, high throughput, and resilience against failures is vital in these sectors, making the choice of consensus protocol essential (Altarawneh *et al.*, 2020 [11]).

This study aims to build upon previous research by evaluating the performance of Kafka, Raft, and Solo in a blockchain-based medical provision information system. The objective is to determine which protocol achieves the best balance among throughput, latency, scalability, and fault tolerance in a real-world production context. The findings from this research hold the potential to guide the development of more secure and efficient blockchain systems across industries such as healthcare, finance, and supply chain management, assisting designers in making informed decisions for deploying Hyperledger Fabric effectively. Ultimately, this study seeks to contribute to the creation of resilient and trustworthy decentralized systems.

Significance of the Study

The importance of blockchain technology in many decentralized systems

should be stated at large, particularly its application in securing sensitive information and ensuring transactions transparency, data integrity and immutability. Blockchain ordering services or Consensus protocols, as the backbone of these blockchain systems directly influence the performance, scalability, and security of the tailored systems. Furthermore, the lack of clarity regarding which consensus protocol provides the most effective solution in production environments remains a significant challenge. This study aims to recommend the most suitable Hyperledger Fabric consensus protocol by conducting a performance evaluation of three protocols in a real-world blockchain-based medical provision system. The focus is on addressing critical performance and security factors, which are essential for both academic research and industry application.

The findings from this study not only guide system designers in selecting the most appropriate consensus protocol based on specific needs but also provide a roadmap for enhancing the efficiency of Hyperledger Fabric in critical sectors such as healthcare, finance, and supply chain management. By identifying the optimal balance of performance metrics like throughput and latency, this research offers valuable insights that can improve the security and reliability of decentralized systems across various production environments.

How a Suitable Consensus Protocol Addresses the Problem.

Selecting suitable consensus protocols is important to maintaining the integrity, security, and performance of a blockchain network. They ensure agreements on the order and validity of transactions among participants, which is crucial in environments that comprise multiple stakeholders.

In such settings, achieving common consensus is vital to prevent fraud and ensure data accuracy while maintaining transparency within the system. The choice of a suitable consensus protocol allows organizations to optimize their systems for high performance without compromising security. This balance is critical for fostering trust among participants and ensuring the reliability of transactions in sensitive sectors like healthcare.

Kafka, Raft, and Solo ordering services each address blockchain consensus challenges by offering distinct trade-offs related to throughput, latency, scalability, and fault tolerance.

Kafka is recognized for its crash fault-tolerant mechanism; it provides high reliability within controlled environments, but is known to introduce performance bottlenecks as transaction volumes grow. This reduces throughput and increases latency, therefore making it less ideal for large-scale systems (Chandrakant, 2024 [10]).

Raft is known to offer a more scalable and decentralized approach that is better suited for larger networks and higher transaction volumes without compromising performance. Solo is not intended for production rather to enable rapid testing and development by reducing the complexity of managing multiple nodes and allowing faster iteration within a single node (Petrescu & Petrescu, 2020 [6]).

The significance of this study lies in evaluating which of these consensus pro-

tocols can deliver the highest throughput and lowest latency in a production setting.

High throughput is critical for systems that process a large number of transactions in real time or a short time, for instance, medical information systems that need to handle patient records, prescriptions, military data, whether data, big data, and other sensitive data. Low latency is also an important factor as it ensures that the transactions are processed quickly and efficiently, maintaining the responsiveness and reliability of the system.

Gaps in Existing Literature

There is confusion or a gap in understanding which consensus protocol offers the highest level of security and performance in the Hyperledger Fabric blockchain systems, and this gap remains underexplored (Nasrulin *et al.*, 2022 [5]). Some existing studies focused on the performance characteristics of these protocols in security evaluated them in real-world applications, or explored their specific impact on security and scalability in a blockchain-based information system.

Hao *et al.* (2018) [3] evaluated consensus protocols' performance, including Raft, Kafka, and Solo, but the study was limited by its controlled environment and use of a small number of nodes, failing to represent diverse or real-world condition, the evaluation didn't test fault tolerance to discover node failure under many transaction hence making their analysis unbelievable.

Chursin & Dag (2024) [12] evaluated DAG-based Byzantine Fault Tolerant consensus protocols, highlighting systems like Mysticeti and Shoal++, which achieve high throughput and low latency by improving upon earlier models. They introduced the Adelie protocol, which addresses vulnerabilities of uncertified DAGs, which demonstrated impressive results, reaching 450 K transactions per second with 630 ms latency, even under node failures. However, their evaluation lacks focus on key Hyperledger Fabric consensus protocols Raft, Kafka, and Solo, hence limiting its relevance to enterprise blockchain systems that rely heavily on these frameworks.

Huang *et al.* (2020) [8] conducted a performance analysis of the Raft consensus protocol, focusing on how network conditions, particularly packet loss, affect its stability and performance in blockchain networks. Their study showed that increasing the election timeout can help reduce the chances of a network split caused by packet loss, but larger networks tend to have more availability issues. While this model provides useful insights into Raft's performance, it does not address other consensus protocols like those used in Hyperledger Fabric.

Altarawneh *et al.* (2020) [11] analysed the Raft consensus protocol's performance, focusing on how network conditions like packet loss impact its stability. They developed a predictive model indicating that increasing election timeouts can mitigate network splits, though larger networks face availability challenges. However, their evaluation lacks a comparative analysis with other consensus protocols used in Hyperledger Fabric, hence limiting the understanding of Raft's relative strengths and weaknesses and reducing its applicability for selecting appro-

priate consensus mechanisms in various blockchain scenarios.

3. The Proposed Hyperledger Fabric Blockchain-Based Approach

This study employs an experimental comparative design to evaluate the performance of Kafka, Raft, and Solo ordering services in a production-like Hyperledger Fabric environment. The evaluation focuses on their performance characteristics within a developed blockchain-based medical provision information system. It gives a roadmap for selecting a suitable consensus protocol to be employed in a system to obtain the expected performance.

Conceptual model for a blockchain-based weighbridge system aimed at enhancing data integrity, security, and operational efficiency in Tanzanian ports, particularly focusing on cargo such as general cargo and grains. The proposed system integrates Hyperledger Fabric to address challenges related to data tampering, inefficiencies, and lack of transparency in traditional weighbridge operations. Within the context of port logistics, various stakeholders—including port authorities, logistics companies, importers/exporters, and regulatory agencies—play critical roles. These stakeholders interact through a blockchain-powered collaborative platform that records and manages weighbridge data in an immutable and secure manner. The system's architecture ensures that weighbridge measurements and operational data are stored in a decentralized permissioned ledger, making it accessible to all stakeholders as permissioned.

This system also seeks to establish a collaborative environment where data generated by weighbridge operations can be transformed into actionable insights. By integrating blockchain with the current weighbridge system at the ports, the system ensures accurate weight measurement, minimizes fraud, and enhances compliance with trade regulations. The platform facilitates seamless communication between port stakeholders, enabling efficient coordination of tasks such as charge payment, clearance for ship dispatch. It also fosters a more inclusive approach by incorporating feedback and operational data from stakeholders, thereby creating synergies between the port, logistics companies, and regulatory bodies.

3.1. Experimental Setup

The conceptual model describes the flow of medical transactions in the blockchain-based system, which is tested using Raft, Kafka, and Solo consensus protocols. Transaction flow begins with a client or user initiating a transaction by sending a proposal to the endorsing peers. These peers execute the chain code, simulating the transaction based on the current ledger state. After the simulation, each endorsing peer signs the transaction and returns it as an endorsement to the client. Endorsing peers, responsible for executing the chain code, also maintain the ledger and the current state, accessible through ledger APIs. Once all endorsements are collected, the client broadcasts the endorsed transaction to the ordering service, which plays a key role in the consensus mechanism.

The ordering service collects, orders, and packages the transaction proposals into blocks, ensuring they are processed sequentially. After ordering, the service delivers the block to all peers, including the committing peers. Committing peers validate the transactions against the endorsement policy to ensure that all conditions have been met. If validated, the peers commit the transaction to the blockchain ledger and update the state database (StateDB). This ensures the transaction is permanently recorded and available for future queries, concluding the transaction process. Figure 1 below shows the transaction process flow in the blockchain-based information system.

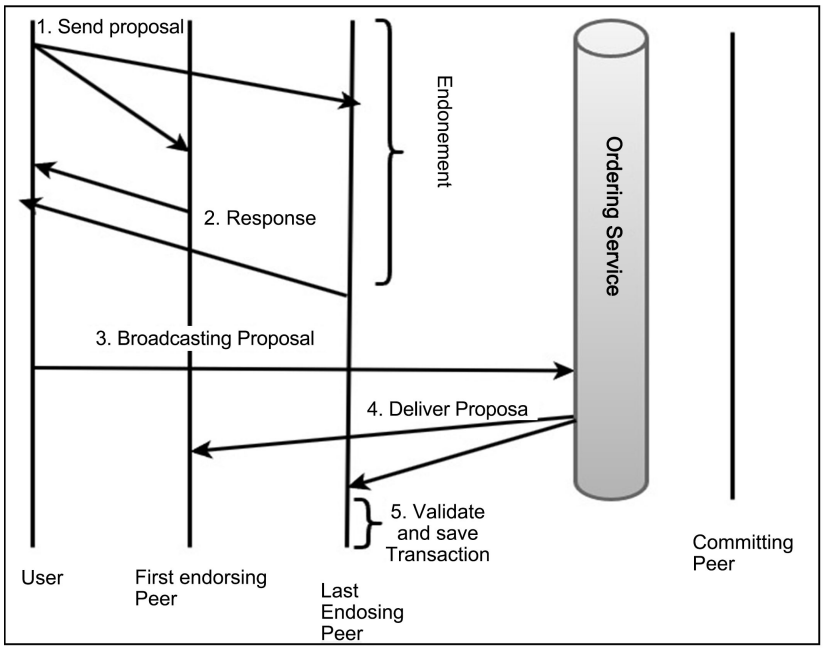

Figure 1. Blockchain transaction flow.

The study employs a medical provision information system designed to manage medical record transactions, ensuring data integrity, security, and transparency. Figure 2 below indicates transaction validation and verification in a medical information system that is used to benchmark the performance of consensus protocols.

Below are the steps undertaken to complete one transaction and save it to the ledger.

1) Transaction Initiation.

A transaction is initiated when a peer within the system generates a request to add or update a medical record. This request is submitted to the network for processing.

2) Validation by Membership Service Provider (MSP).

The transaction is sent to the MSP for validation, where the authenticity and authorization of the transaction are verified. The MSP ensures that only authorized users can execute transactions.

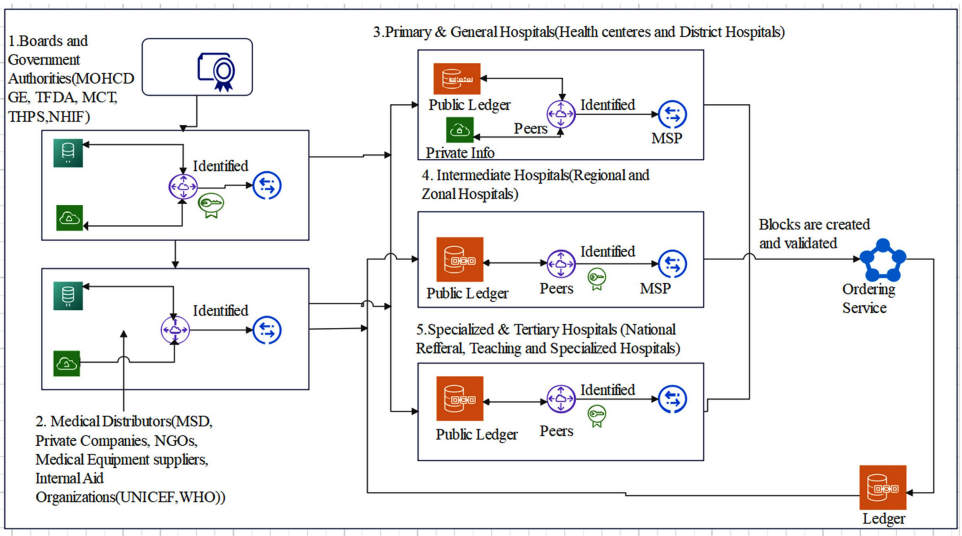

Figure 2. Validation and verification process in the Hyperledger Fabric Blockchain System.

3) Transmission to Ordering Service.

Once validated by the MSP, the transaction is being forwarded to the ordering service for further processing, also retains a transaction copy is retained for comparison purposes.

4) Validation by Ordering Service.

The ordering service verifies the transaction against the system's chain code (smart contract) and predefined business rules. The ordering service orders the transactions based on the consensus protocol in use.

a) Raft follows a leader-based replication model, ensuring that a leader node orders the transactions and replicates them to other nodes.

b) Kafka relies on an external distributed messaging system, where transaction ordering is handled by Kafka clusters.

c) Solo, as a single node protocol used for testing environments, offers a limited fault tolerance and scalability.

5) Ledger Update.

Once the ordering service validates and sequences the transaction, it updates the blockchain ledger, and then transaction details are recorded immutably on the ledger.

6) Broadcast to Peers.

The updated ledger is broadcast to all peers in the network to ensure that every peer has a synchronized version of the ledger.

7) Peer Verification and Acceptance.

Each peer verifies the transaction by comparing it with their local copy of the ledger. Once verified, the peers accept the transaction and update their ledgers.

8) Transaction Completion

Upon verification and acceptance by all peers, the transaction is deemed complete. The blockchain ensures that all nodes agree on the final state of the ledger while maintaining transparency and immutability of the network ledger.

3.2. Research Design

The research adopted an experimental design by conducting performance tests on the three consensus protocols in a controlled Hyperledger Fabric network environment. The network is a private, permissioned blockchain that processes medical transaction data, which is critical for ensuring patient privacy and data integrity. The performance of each protocol is assessed based on throughput, latency, fault tolerance, and scalability to identify the most suitable option for production systems.

3.3. System Development Life Cycle (SDLC) Phases

The study used SDLC methodology to ensure a structured and systematic approach for system implementation, by following all six stages of the SDLC from requirement gathering, system design, implementation, testing, deployment, and maintenance.

Requirements Gathering

System requirements were gathered through an extensive stakeholders' analysis to develop a secured Blockchain-based medical provision information system for evaluation and comparison purposes. The functional requirements focused on key features including transaction validation, patient records management, and logging and auditing.

Non-functional requirement based on high throughput, low-latency, security, and fault tolerance attributes. These requirements were informed by consultations with healthcare sector stakeholders to ensure that the system meets real-world needs.

System Design

The design of the system architecture depicted in immutable and temper free ledger in this context by considering several essential components like Certificate Authority (CA), Membership Service Provider (MSP), Smart Contracts (Chain code or chain of blocks), Private and Public Keys, harsh values and Consensus protocols (ordering services).

Peers and Ordering Nodes: Peers are responsible for endorsing transactions, while the ordering nodes utilize Kafka, Raft, and Solo consensus protocols to ensure the proper transaction sequencing.

Membership Service Provider (MSP): The MSP manages authentication and access control, ensuring that only authorized participants can access the system (network).

Ledger and Smart Contracts: The Blockchain Ledger is the database that is used to record and save all transaction data, whereby smart contracts or chain code define business logic to the system.

Implementation

During the implementation phase, the Hyperledger Fabric network was established in a private infrastructure, and the medical provision information system was deployed. Each consensus protocol, Kafka, Raft, and Solo, was configured as the ordering service for specific instances of the blockchain network. The network

configuration is maintained constantly (unchanged) to ensure a fair comparison of the consensus protocols' performance across all implementations.

The experimental environment was set up as shown in **Figure 3** below including the important key criteria, which are participating servers, also known as peer nodes, ordering services, also known as consensus protocols, and transaction data. Peer nodes are deployed across multiple servers to endorse intended transactions. Ordering Services or consensus protocols were tested in separate trials to assess performance differences.

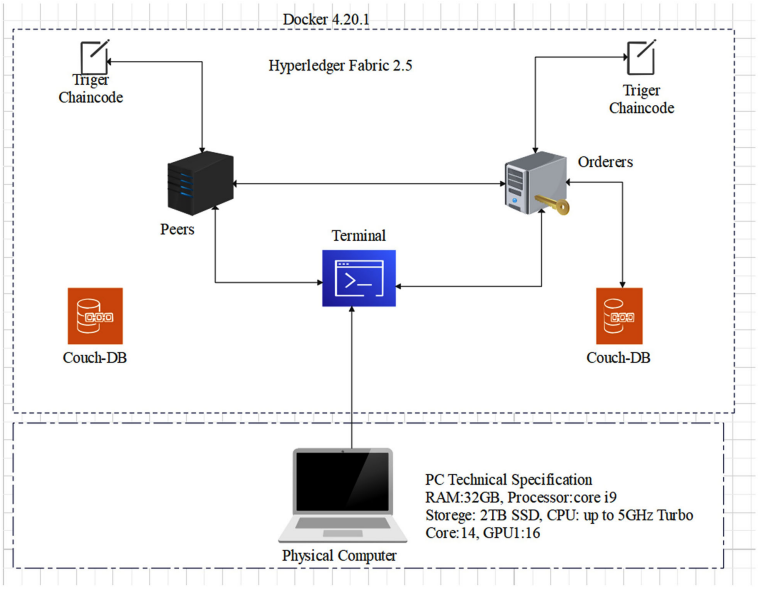

Figure 3. Project environment setup.

Testing Phase

Testing phase involved measuring performance through rigorous evaluations of throughputs and latency across varying transaction volumes in 10, 100, 1,000, 10,000, 100,000, to 1,000,000 transactions ranges. During the maintenance phase, different issues were tracked. The issues included 40 bugs, 20 feature requests, 15 system improvements, and 5 other items. Real-time metrics were monitored using Python, and Grafana was used to visualize system performance, helping to identify performance issues.

Deployment

A blockchain-based medical provision information system was deployed in a maintained environment that simulated real-world conditions. The system was monitored for its responses to various stress scenarios, such as transaction overload, network partitioning, and peer node failures, to evaluate its resilience and reliability.

Maintenance

After system deployment, continuous monitoring was conducted to further evaluate performance and carry out maintenance tasks. Several updates were implemented to improve system stability, and the resulting data was analyzed to as-

sess the long-term reliability and scalability of the consensus protocols. It was anticipated that some malfunctions could occur during this phase. Consequently, tracking and resolving bugs, as well as addressing feature requests, became an essential factor to ensure optimal functionality. Figure 4 illustrates the number of system issues that were reported.

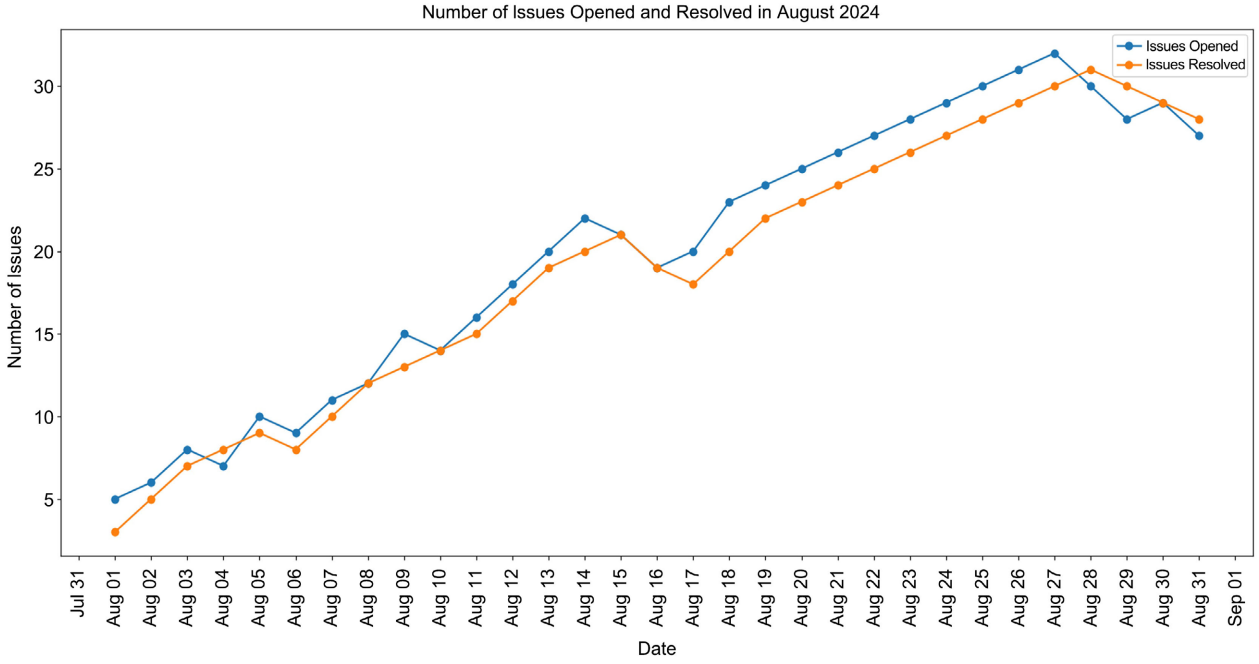

Figure 4. Number of issues opened and resolved.

Secure Hash Algorithm (SHA256)

Figure 5 illustrates the proposed system utilizes the Secure Hash Algorithm (SHA256) to generate cryptographic hashes for ensuring transaction integrity and security. This guarantees that all transactions recorded on the ledger are tamperproof and immutable, which means no one can edit a record prior and post transaction.

Elliptic Curve Digital Signature Algorithm (ECDSA).

ECDSA is used to sign transactions by providing cryptographic authenticity and non-repudiation. Each transaction is verified using ECDSA to confirm the legitimacy of the transaction initiators, thereby enhancing overall system security. ECDSA also ensures non-repudiation, meaning the user cannot deny having authorized the transaction since the signature is cryptographically linked to their private key. This is critical for maintaining accountability within the system. The use of ECDSA not only strengthens system security but also ensures that transactions are traceable, verifiable, and tamper-proof, which enhances the overall trustworthiness and scalability of blockchain networks.

The following diagrams elaborate on the pictorial presentation of an ECDSA and give necessary information on how the Elliptic Curve Digital Signature Algorithm works. Figure 6 shows ECDSA pictorial presentation.

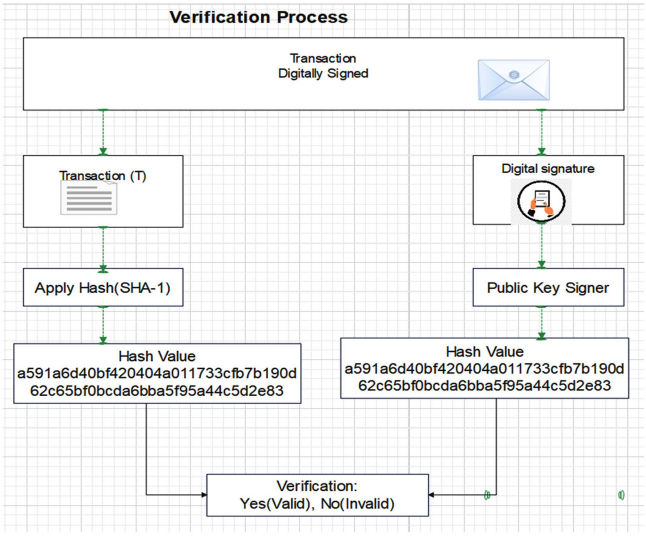

Figure 5. Transaction hashing verification.

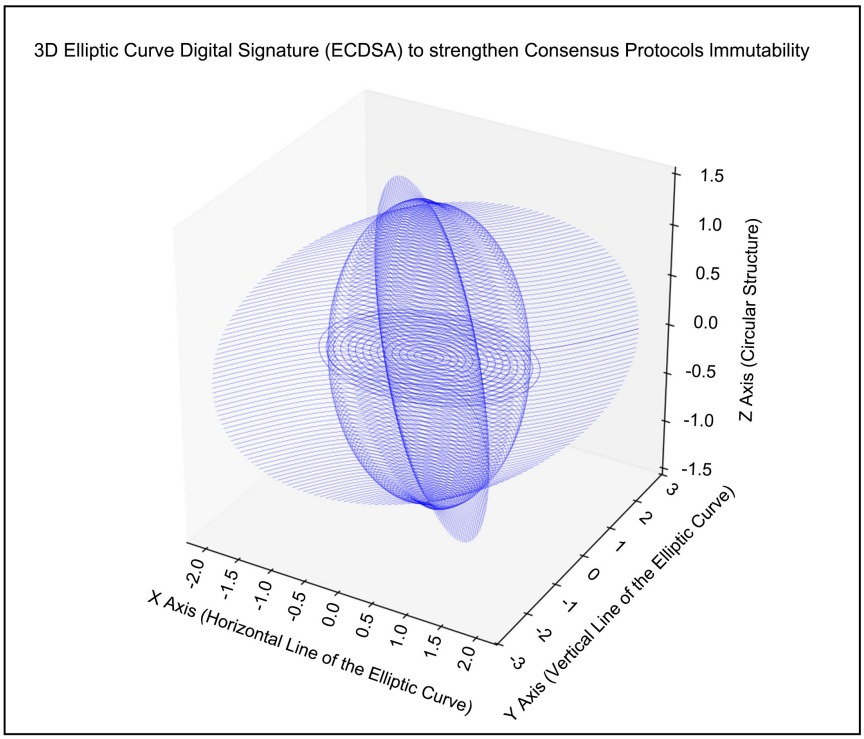

Figure 6. 3D Representation of an Elliptic Curve (ECDSA).

Consensus Protocol Design

The consensus protocols were integrated into the blockchain network as an ordering service, which is responsible for transaction ordering and ensuring that all nodes process transactions in a consistent sequence. This ensures consistent transaction ordering across all nodes. The implemented code establishes a blockchain system comprising a block class to define block structure, a blockchain class to manage the chain, and a transaction class for transaction details. A FIFO consensus mechanism is employed to process transactions in batches and to create new

blocks in a dedicated thread to maintain integrity and immutability.

Performance Evaluation Metrics

The following performance metrics are utilized to evaluate the consensus protocols.

- Throughput: The throughput performance was measured as the number of transactions processed per second (TPS) across different load conditions to provide insights into the efficiency of each protocol.
- Latency: Defined as the time taken to commit a transaction to the blockchain ledger, indicating the responsiveness of the system.
- Fault Tolerance: Assessed by evaluating the system's ability to handle node failures without disrupting network operations, which is crucial for maintaining service continuity.
- Scalability: Measured by the network's ability to maintain performance as the number of nodes or transaction volume increases, reflecting the protocol's capacity to grow with demand.

Evaluation Limitations

The current evaluation acknowledges several limitations.

- The study focused on a limited set of transaction volumes and predefined scenarios, which may not fully capture performance variations encountered in diverse operational conditions.
- The metrics used to measure throughput and latency provide a snapshot of performance but may not account for complexities in real-world deployments, including resource utilization and fault tolerance.
- Furthermore, the scope does not include integration with other blockchain components and external systems, which could impact overall performance and interoperability.

4. Result Analysis for Performance Evaluation

The findings of the performance evaluation conducted based on the Kafka, Raft, and Solo consensus protocols focused on latency and throughput by maintaining fault tolerance and scalability metrics to determine which protocol offers the best performance for production-level in any blockchain-based system.

The results conclusively indicate that Kafka is the most suitable consensus protocol for production environments (this can also be indicated in the abstract) that can give high throughput, low latency, fault tolerance, and scalability by processing many transactions. Kafka consistently outperformed Raft and Solo, particularly in larger and more complex systems, by demonstrating superior performance under heavy loads. Raft was effective in medium-scale environments but struggled to match Kafka's efficiency in larger-scale operations. On the other hand, Solo proved insufficient for production use by showing significant performance drops as the network grew.

4.1. Throughput Analysis

Throughput was measured as the number of transactions processed per second

(TPS) by each protocol under varying load conditions. The results indicated that Kafka consistently demonstrated the highest throughput, particularly in environments with high transaction volumes, outperforming both Rafka and Solo. The evaluation spanned transaction volumes from 10 to 1,000,000 transactions in the range of 10, 100, 1000, 10,000, 100,000 to 1,000,000.

Figure 7 illustrates the throughput performance for Kafka, Raft, and Solo to highlight Kafka's superior capability in high load scenarios. Raft performed well with large transaction volumes but exhibited higher latency. as compared to Kafka, whereby Solo showed the lowest throughput among the three protocols.

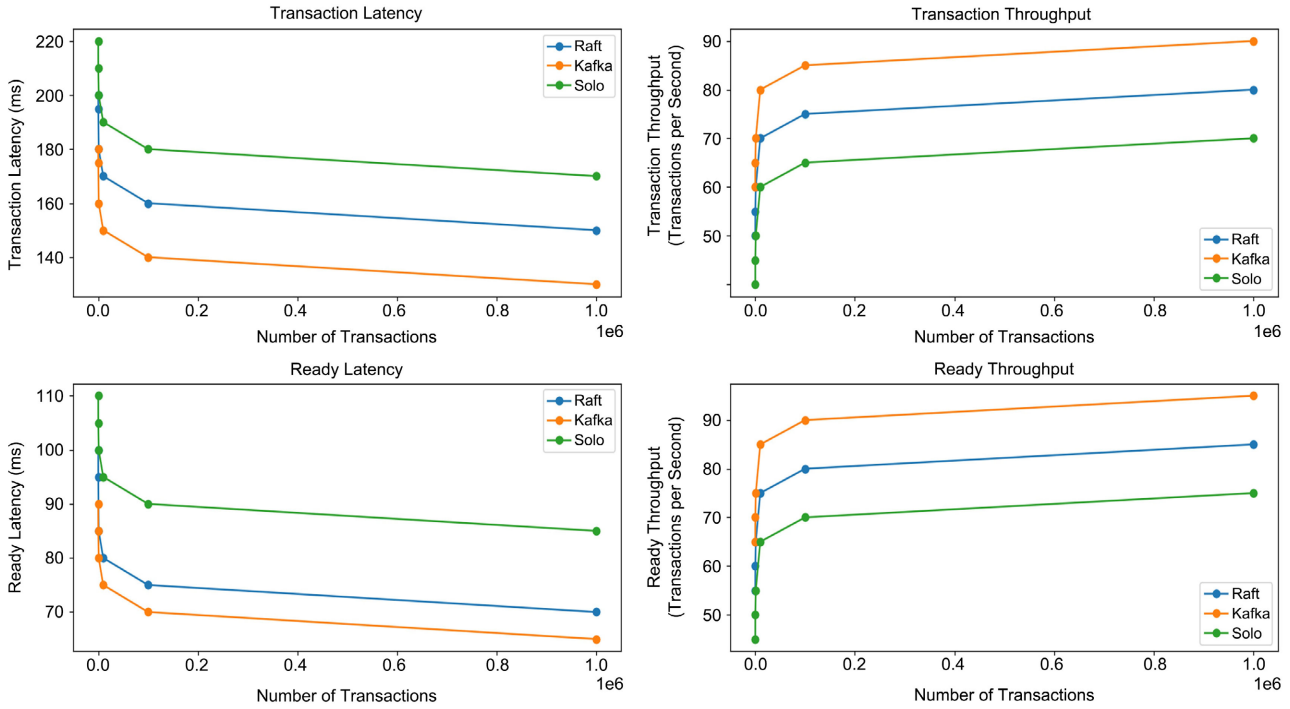

Figure 7. Throughputs vs Latency performance analysis.

4.2. Latency Analysis

Latency was evaluated by measuring the time taken for a transaction to be committed to the blockchain ledger. The analysis revealed that Kafka consistently outperformed Raft and Solo, exhibiting lower latency even under heavy network traffic conditions.

Figure 8 shows the latency performance for Kafka, Raft, and Solo, emphasizing Kafka's efficiency. Raft showed a significant latency increase as network traffic increased, rendering it less suitable for high-transaction environments. Solo showed a lower latency, but it has a reduced fault tolerance.

4.3. Fault Tolerance

Fault tolerance was tested by simulating node failures within the network. Kafka emerged as the most resilient protocol by maintaining consensus among the nodes and performance even in the face of multiple node failures.

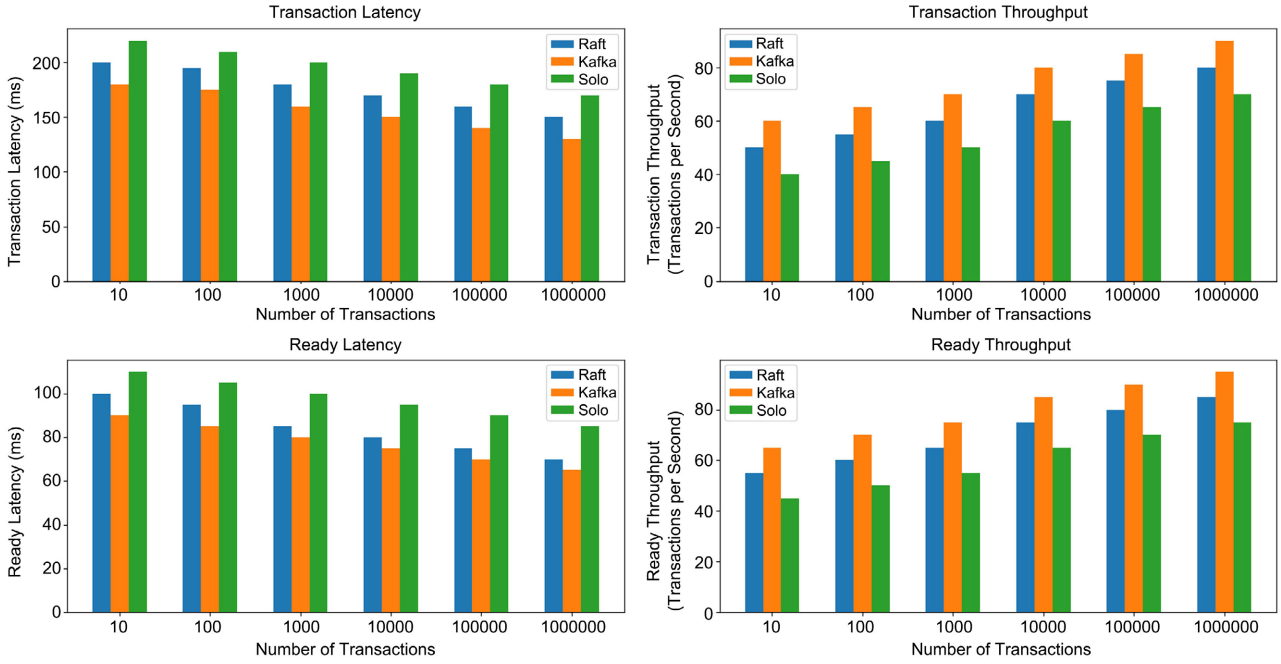

Figure 8. Consensus protocols performance comparison.

Raft also demonstrated strong fault tolerance by successfully handling node failures, although it performed slightly below Kafka in maintaining overall performance. However, Solo was unable to recover from node failures, highlighting its limitations for production use.

4.4. Scalability

Scalability was assessed by increasing the number of peer nodes and transactions within the network. Kafka demonstrated superior scalability by efficiently handling increased loads while maintaining its high throughput despite its reliance on external resources. Raft scaled efficiently, but its performance began to degrade under heavy transaction loads, falling behind Kafka. Solo performed adequately in small-scale environments but failed to meet the demands of large-scale production. As the number of nodes and transactions grew, its performance dropped sharply, making it unsuitable for handling high transaction volumes.

4.5. Comparative Trade-Offs between Kafka and Raft Consensus Protocols

Criteria	Kafka	Raft	Notes
Throughput	High (handles large-scale transactions)	Moderate (less efficient at scale)	Kafka excels in high-load environments.
Latency	Low latency	Moderate latency	Kafka generally provides faster commit times.
Fault Tolerance	High (supports broker failover)	Moderate (leader-based failover)	Kafka tolerates more broker failures; Raft depends on leader stability.

Continued

Operational Overhead	Requires managing an external Kafka cluster (Zookeeper, brokers)	Built-in within system; simpler deployment	Kafka's external components add complexity.
Energy Efficiency	Higher resource consumption due to distributed brokers	Lower resource footprint due to simpler setup	Important for sustainability and cloud cost.
Decentralization	Distributed architecture with multiple brokers	Leader-based consensus; more centralized	Raft centralizes decision-making; Kafka is more distributed.
Scalability	Highly scalable (horizontal scaling of brokers)	Limited scalability (leader bottleneck)	Kafka better for large clusters.
Security Features	Supports TLS, SASL, ACLs	Supports TLS, but fewer built-in features	Kafka offers richer enterprise-grade security.
Use Cases	Large-scale, fault-tolerant, high-throughput systems	Smaller clusters, simpler deployments, internal services	Choose based on complexity and scale needs.

4.6. Evaluation Limitations

The current evaluation of consensus protocols, Raft, Kafka, and Solo, presents some limitations that need to be addressed. The study focused on a limited set of transaction volumes and predefined scenarios, which may not fully capture the performance variations under different operational conditions. Metrics used to measure throughput and latency provide a snapshot of performance but may not account for the complexities involved in real-world deployments, including resource utilization and fault tolerance. Also, the scope of the evaluation does not include the integration of these protocols with other blockchain components and external systems, which could impact their performance and interoperability.

5. Discussion

The results indicate that Kafka outperformed other consensus protocols across throughput, latency, fault tolerance, and scalability key metrics on varying transactions.

Kafka demonstrated the highest transaction throughput under all conditions. Kafka effectively managed parallel data streams and large transaction loads without overwhelming the network. Its built-in fault tolerance enables quick recovery from failures while maintaining the integrity of the transaction ledger hence made it to lead in throughput and performance in many transactions.

Raft steady performance: Raft showed good performance in terms of consistency and fault tolerance. However, it managed to stretch higher latency compared to Kafka. Raft performed well under moderate transaction loads but struggled with larger volumes. Since Raft is designed to use one main node (the leader) to manage and coordinate everything, it ensures the system stays consistent.

Solo's Limitations: Solo as expected, was constrained by built-in single-node design. While it handled low-latency transactions in small-scale environments, it struggled with larger volumes. Without the redundancy of Kafka and Raft, Solo could not handle node failures effectively hence ended by giving poor results, therefore making it unsuitable for production environments that require high fault tolerance and scalability.

Transaction Volume Analysis: Kafka maintained large throughput experiment even if transaction volumes got increased, hence proving its suitability for environments with high loads. Raft performance declined as transaction volumes rose, concluding that it struggled with parallel transaction streams. Solo performance decreased in moderate transaction volumes, confirming its role as a testing tool rather than a real-world solution.

Fault Tolerance: Kafka's distributed design allowed it to recover from node failures faster than Raft, which experienced delays in electing a new leader. Kafka's automatic failover process minimized disruptions in transaction processing. Solo lacked meaningful fault tolerance and could not maintain system integrity during failures.

Scalability: Kafka scaled effectively with both transaction volume and the number of nodes. It handled the largest test environments without significant performance drops. Raft, although was capable of scaling but required more resources to maintain performance and began showing limitations under extreme conditions. Solo by its design, could not scale beyond its single-node setup.

In summary, Kafka architecture made it the best performer, especially in environments with high transaction loads where scalability and fault tolerance are crucial. Raft is suitable for systems with moderate transaction volumes. Solo should be restricted to testing due to its limitations.

6. Conclusions and Future Work

This study demonstrates that Kafka is the most suitable consensus protocol for a blockchain-based information system. Its distributed architecture allowed it to process more transactions with lower latency compared to Raft and Solo, and it offered superior fault tolerance and scalability. Kafka's ability to handle large-scale transactions and quickly recover from failures makes it the best choice for production environments giving both efficiency and resilience. Raft, while viable, was less efficient in handling high transaction volumes and showed limited scalability. Solo, as anticipated, was unsuitable for production due to its inability to handle large transactions and its lack of fault tolerance.

Future research should examine how consensus protocols perform in more complex environments, especially when integrated with external systems or exposed to varying network conditions. This includes measuring latency and throughput over 1 Gbps and 10 Gbps networks and across geographically distributed nodes to assess protocol sensitivity to bandwidth and latency variations. This exploration could provide deeper insights for selecting the right consensus proto-

col for blockchain-based systems in real-world applications, addressing critical issues of scalability, fault tolerance, and interoperability. To address current limitations, future work should expand the range of transaction volumes and workloads to better understand how these protocols scale in large-scale environments. Simulating node failures and network disruptions is essential for a comprehensive assessment of fault tolerance and reliability, testing Kafka with up to 50% broker node failures can reveal its robustness and recovery capabilities under extreme conditions. Moreover, integrating blockchain protocols with external systems such as identity providers, cloud based storage and APIs, would offer insights into interoperability challenges. Further exploration of advanced metrics and detailed latency breakdowns can optimize throughput and performance. Additional evaluation criteria should include user experience, operational impact, and security assessments to ensure the robustness and effectiveness of consensus protocols in real-world deployments.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Munasinghe, L. and Halgamuge, S.K. (2023) Performance Evaluation of Blockchain Consensus Protocols: A Survey. Concurrency and Computation: Practice and Experience.

[2] Yang, G., Lee, K., Lee, K., Yoo, Y., Lee, H. and Yoo, C. (2022) Resource Analysis of Blockchain Consensus Algorithms in Hyperledger Fabric. *IEEE Access*, **10**, 74902-74920. https://doi.org/10.1109/access.2022.3190979

[3] Hao, Y., Li, Y., Dong, X., Fang, L. and Chen, P. (2018) Performance Analysis of Consensus Algorithm in Private Blockchain. 2018 *IEEE Intelligent Vehicles Symposium* (*IV*), Changshu, 26-30 June 2018, 280-285. https://doi.org/10.1109/ivs.2018.8500557

[4] Uddin, M., *et al.* (2021) Blockchain for Healthcare Data Management: Opportunities Challenges, and Future Recommendations. *Healthcare*, **9**, 614.

[5] Nasrulin, A., *et al.* (2022) A Comprehensive Survey on Consensus Mechanisms in Blockchain. *IEEE Access*, **10**, 1200-1223.

[6] Petrescu, A. and Petrescu, M. (2020) Performance Analysis of Hyperledger Fabric Consensus Algorithms. *Procedia Computer Science*, **176**, 105-112.

[7] Wang, C. and Chu, X. (2020) Performance Characterization and Bottleneck Analysis of Hyperledger Fabric. 2020 *IEEE* 40*th International Conference on Distributed Computing Systems* (*ICDCS*), Singapore, 29 November-1 December 2020, 1281-1286. https://doi.org/10.1109/icdcs47774.2020.00165

[8] Huang, L., *et al.* (2020) Impact of Network Conditions on Raft Consensus in Blockchain Networks. *Future Generation Computer Systems*, **108**, 1-9.

[9] Nguyen, M.Q., Loghin, D. and Dinh, T.T.A. (2021) Understanding the Scalability of Hyperledger Fabric.

[10] Chandrakant, S. (2024) Evaluating Consensus Protocols in Hyperledger Fabric: A Comparative Study. *International Journal of Advanced Computer Science and Applications*, **15**, 45-52.

[11] Altarawneh, M., *et al.* (2020) Predictive Modelling of Raft Consensus Performance Under Network Variations. *Journal of Network and Computer Applications*, **168**, 102-110.

[12] Chursin, A. and Dag, S. (2024) Enhancing DAG-Based Consensus Protocols for High Throughput and Low Latency. *Distributed Ledger Technologies Journal*, **3**, 15-28.