

Redefining Hybrid Blockchains: A Balanced Architecture

Syed Ibrahim Omer
City University of Hong Kong
github.com/ibitec7/blockchain

Abstract. Blockchain technology has completely revolutionized the field of decentralized finance with the emergence of a variety of cryptocurrencies and digital assets. However, widespread adoption of this technology by governments and enterprises has been limited by concerns regarding the technology's scalability, governance, and economic sustainability. This paper aims to introduce a novel hybrid blockchain architecture that balances scalability, governance, and decentralization while being economically viable for all parties involved. The new semi-centralized model leverages strategies not prevalent in the field, such as resource and node isolation, containerization, separation of networking and compute layers, use of a Kafka pub-sub network instead of a peer-to-peer network, and stakes-based validator selection to possibly mitigate a variety of issues related to scalability, security, governance, and economic sustainability. Simulations conducted on Kubernetes demonstrate the architecture's ability to achieve over 1000 transactions per second, with consistent performance across scaled deployments, even on a lightweight consumer-grade laptop with resource constraints. The findings highlight the system's scalability, security, and economic viability, offering a robust framework for enterprise and government adoption.

Keywords. Hybrid Blockchain, Scalability, Security, Economic Viability, Kafka, Kubernetes, Permissioned Systems, Containerization.

1. Introduction

1.1 Background

The impact that blockchain technology has had on distributed systems and decentralized finance is revolutionary. Since its inception, it has led to increased trust and security in the systems using this technology. A distributed network of public validators ensures data integrity, fault tolerance, and trust by collectively agreeing on a specific state of data. However, widespread adoption of the technology by governments and enterprises has been limited by the challenges related to its scalability, governance, and economic value. Such organizations can not allow public validators to participate and ultimately influence the sensitive data that these organizations own. Therefore, they have adopted private and

semi-private blockchains that yield at least some authority to the hosting organizations. However, trying to host, maintain, and scale an entire distributed system on its own has been challenging for these organizations as the entire network is controlled by a single or, at most, a small group of organizations. Moreover, as the system scales, the financial burden on these organizations to maintain the underlying infrastructure increases. Existing solutions like public or permissioned blockchains fail to address all these issues thoroughly. In public blockchains, individual validators ensure scalability but compromise governance, as authority over sensitive data rests with public participants. Conversely, permissioned blockchains prioritize governance and data control but suffer from scalability challenges, reduced public trust, and high operational costs.

1.2 Motivation

To achieve broader adoption of blockchain technology for enterprise and government use it is imperative to tackle the issues related to scalability, economic viability, and governance adequately. The solution that such organizations need must be scalable, economical, and offer adequate governance for the sensitive data that these organizations possess. There is a need for a hybrid blockchain architecture that combines the data integrity and trust of public blockchains with the effective governance of private blockchains to provide a scalable and economical solution that these organizations can implement sustainably. This hybrid approach leverages the strengths of both public and private blockchains while addressing their respective shortcomings. Such solutions may also create opportunities for Central Bank Digital Currencies, Capital Markets, Supply Chain Management, etc.

1.3 Research Objectives and Contributions

By acknowledging the challenges that governments and enterprises have to confront for the adoption of blockchain technology, this paper aims to resolve such issues effectively.

Therefore, the objective of this paper is to:

- Design a novel blockchain architecture that can be implemented practically by governments and enterprises.
- Solve the issues related to scalability, governance, and economic viability that are hindering existing blockchain solutions.
- Implement and demonstrate the new architecture as a practical and viable proof-of-concept solution that solves the issues discussed.

Given these objectives, this paper contributes the following to the academic state of blockchain research:

- Introduction of a semi-centralized blockchain model for effective data governance and transparency.
- Separation of networking and consensus aspects of the blockchain architecture as separate modules that can scale independently.

- Introduction of applications as containerized microservices in the blockchain ecosystem that enhance scalability, security, and provide consistent performance.
- A stake-based validator selection mechanism for providing economic incentives and safety guarantees.
- A demonstration of scalability of the new architecture through simulations with consistent performance across scaled deployments even on resource-constrained hardware.

1.4 Paper Structure

The following paper is divided into 8 sections with each section serving a unique purpose. The rest of the paper is structured as follows:

- Section 2 discusses academic work related to the objective of this paper.
- Section 3 presents the proposed architecture of this paper.
- Section 4 provides theoretical analyses for the proposed architecture.
- Section 5 details the experimental setup and simulations to test the new architecture.
- Section 6 presents the results and evaluation of the experiment conducted.
- Section 7 discusses the results of the experiment, evaluates their alignment with the theoretical analyses presented in Section 4 and any other challenges or insights.
- Section 8 concludes the paper with the architecture's limitations, areas for improvement, future work and implications.

2. Related Work

2.1 Overview of Blockchain Architectures

Currently, there are three broad classes of blockchain architecture that offer varying levels of governance: Public blockchains, Permissioned or private blockchains, and hybrid blockchains. Public blockchains work based on decentralized authority without relying on a single trusted authority. Trust is built as a collective voting consensus among the network participants. Permissioned blockchains, however, limit the number of participants who can contribute to the network by only granting access to participants with permissions. The participants are usually trusted parties and operate on a smaller and more private scale. Hybrid blockchains combine different aspects of public and permissioned blockchains to provide a policy that balances the purposes of both. Such architectures are ideal for organizations aiming for a custom solution without implementing either the extreme of public or private chains.

2.2 Proof-of-Work

The original Bitcoin whitepaper introduces the Proof-of-Work consensus model that relies on the security of the blockchain by the cryptographic work done on the chain. In this model, nodes compete against one another by expending resources and energy to find the correct nonce: a number that gives a specific number of leading zeros when hashed together with the block header [1]. While this leads to a high degree of cryptographic security, the solution is not scalable and efficient, which is crucial for private adoption. On its own, this solution can only produce up to one block every ten minutes [2]. Furthermore, the amount of resources expended for the security is also impractical. As of December 8, 2024 Bitcoin's energy consumption has been estimated to be around 175.87 TWh per year [3]. For comparison, the total annual energy consumption of Pakistan (a country with a population of nearly 240 million) is 128.96 TWh as of 2023 [4]. Moreover, the paper argues that as long as the honest nodes possess more computational power than the attackers, they can outpace the attackers, and the probability of the attackers catching up to the chain and compromising the network decreases exponentially as more blocks are added. While this strategy guarantees unprecedented security for the blockchain, it does not adequately address the vulnerability of a potential 51% attack. The paper defends the case for a hypothetical 51% attack where the attackers have more computational power than the honest nodes by assuming they will likely use the surplus power to put honest work on the chain rather than compromise it since honest behavior would benefit them economically [1]. However, this assumption may not hold if a government implements such a system because the attackers may not be interested in the economic incentive gained. For instance an adversarial state-level attacker or a terrorist organization may not be interested in using the computational advantage for economic gain as it does not align with their objectives.

2.3 Proof-of-Stake

Ethereum 2.0 introduced the Proof-of-Stake consensus model, where validators that decide on the next block are chosen based on their stake in the network [5]. With the risk of losing their stake, this system assumes that the actors will act honestly. However, this model introduces the risk of an unpredictable level of centralization, which may be worse than if the governments had a fully decentralized network. If there is some form of centralization in such a model, it would be even worse because instead of having a decentralized system, they have instead stumbled upon a centralized system that outsources the central authority to a foreign entity. Suppose a small faction of validators gains the majority control by having a majority collective stake. They may collaborate in compromising or influencing the system at their will. Without a central authority to break this monopoly, there is almost no way for the honest nodes to gain back authority over the network. This raises serious concerns about the security and governance of sensitive data that these validators may have over the hosting organizations. This model guarantees the system's security by assuming that the significant amount of stake that the attackers would have to put up to compromise the system would be highly impractical. While this assumption may hold for decentralized cryptocurrencies,

state-level attackers may not have a shortage of funds to gain central control of the system through staking. So, such a problem raises concerns related to both security and governance.

2.4 Scalability Solutions

The low transaction throughput, such as 7 transactions per second for Bitcoin and 30 transactions per second for Ethereum 1.0 [5], along with high latency for transactions and resource-heavy consensus mechanisms, required innovative solutions that could address these issues. Various strategies have been employed, such as Layer 2 solutions, Sharding, and innovations in consensus protocols, to make the technology more scalable and practical for widespread use. Innovations in consensus protocols offer the most promising solutions as they aim to address scalability issues from the foundation. For instance, Tendermint employs a modular architecture that separates consensus and networking, allowing for independent scalability, a strategy that is also employed in this paper [6]. Layer 2 solutions are meant to be implemented on top of existing blockchain implementations to improve their scalability. These solutions try to settle interactions off-chain and only post the final or compressed results to the chain to decrease the load on the blockchain. Such solutions include the Lightning Network that tries to settle micropayments among users [7], rollups that execute transactions off-chain but only post the compressed proof of the transactions to the blockchain, and state channels that allow the execution of off-chain interactions privately, only posting the results on the chain [8]. However, these solutions rely heavily on localized trust, which may not be ideal for the governance needed for enterprise or government use cases. Lastly, sharding solutions have been employed to physically partition a blockchain's state and transaction processing workload across multiple smaller chains called shards [8]. Therefore, the blockchain can benefit from parallel processing, increasing transaction throughput. However, this method can introduce a degree of complexity that makes the solution difficult to scale and implement. Managing inter-shard communication is challenging as transactions crossing shards can cause increased network congestion and require increased synchronization, which can easily lead to high latency and poor performance.

2.5 Permissioned Blockchains

Permissioned blockchains aim to resolve the issues related to privacy and governance by enabling fine-grained access control. The governance structure of permissioned blockchains also aligns well with the regulatory requirements under which such entities must operate. Such architectures may also improve performance compared to public blockchains, as there is a smaller group of trusted validators for consensus. However, scalability in permissioned blockchains is limited as the responsibility for scaling lies primarily with the governing entities, which can incur a significant cost when scaling a distributed infrastructure. Permissioned blockchains also suffer from increased centralization, which risks a single point of failure that can compromise the system. Moreover, this model offers limited transparency as the data is not democratized, which may lead to decreased public trust and, in specific contexts, failure to adhere to regulatory compliances related to the freedom of information. Implementations like Hyperledger Fabric offer a high degree of modularity and configurability in the consensus mechanisms and permission policies [9]. However,

fundamentally, any possible configuration remains permissioned regardless of the leniency or improvisation they offer to achieve a hybrid policy. Regardless, permissioned blockchains lay the groundwork for hybrid architectures by offering robust and controlled environments for data governance.

2.6 Hybrid Blockchain Architectures

Most of the work done on hybrid blockchains has either been an improvisation of permissioned blockchains to offer leniency for public validators to participate, or an attempt to merge public and private blockchains. This improvisation causes such solutions to solve the governance issue at the expense of creating another issue related to security or scalability. For instance, Dragonchain, an attempt to merge public and private chains, allows enterprises to use private blockchains while interoperating with public blockchains like Ethereum or Bitcoin for data anchoring [10]. In doing so, it relies on external public chains for anchoring, creating performance and scalability bottlenecks. Moreover, such improvisations lead to interoperability issues as seamlessly connecting public and private chains is technically complex and often lacks standardized protocols due to distinct operating policies. Furthermore, such implementations can introduce the risk of new vulnerabilities, especially at integration points. Therefore, for a successful implementation of a hybrid blockchain, it is necessary that any new architecture addresses these issues inherently and cohesively implements the policy to reduce the risk of vulnerabilities, or performance and scalability limitations. Therefore, there is a need for a new hybrid architecture that, from its foundation, is designed to offer a hybrid governance policy while being scalable and secure, cohesively integrating all its features to avoid bottlenecks and security vulnerabilities.

2.7 Kafka and Microservices in Blockchain Systems

Apache Kafka is a distributed event-streaming platform known for its ability to handle high-throughput, real-time data pipelines [11]. In blockchain systems, Kafka has the potential to be used as a messaging middleware to handle node-to-node communication, reduce delays caused by synchronous validation by enabling asynchronous processing, and provide store and replay streams for effective auditing, fault tolerance, or debugging. Hyperledger Fabric, a permissioned blockchain, utilizes Kafka as an ordering service, ensuring transactions are batched and ordered before processing [12]. While effective for ordering transactions, this is a limited implementation of Kafka that can potentially be used as a distributed messaging and communications tool among nodes. The microservices architecture involves breaking applications into smaller, loosely coupled, and independently deployable services. When used in the context of blockchains, they have the potential to modularize the blockchain functionality, allowing developers to work on each module independently without compromising the entire system. Moreover, such applications can improve scalability by allowing services to scale independently. Such implementations can also improve fault isolation and simplify debugging without jeopardizing the system. Hyperledger Indy is a blockchain architecture that uses microservices principles by modularizing identity services such as verifiable credentials and decentralized identifiers [13]. However, its implementation is limited and lacks general blockchain application.

2.8 Summary and Research Gap

Mainstream public or permissioned blockchain implementations fail to fully address the issue of balancing governance, security, scalability, and decentralization needed for enterprise and government adoption, prompting the need for hybrid solutions. There has been a significant gap in research related to hybrid blockchains, with existing solutions being improvised implementations of existing mainstream blockchain solutions, causing new issues related to scalability, security, or governance. As highlighted by the Ethereum whitepaper, the blockchain trilemma of balancing security, scalability, and decentralization is a complicated issue that can not be resolved by an improvised solution or a layer 2 solution [5]. It requires a new consensus architecture to be built with clear prior objectives to solve these issues from the foundation of the new architecture. Kafka and microservices have been utilized in specific limited contexts but have not been implemented to benefit from their full potential due to the limitations of existing architectures. There has been a lack of research exploring using such technologies to solve governance, scalability, and security issues.

3. Proposed Architecture

3.1 Architecture Principles and Design Objectives

The solution that this paper proposes is based on three foundational principles: to offer a degree of decentralization with effective governance; implement a solution that is inherently designed to be secure, scalable, easily deployable and upgradable; and to formulate a consensus policy that is economical for all parties involved. Therefore, to operate on these principles practically the design of the architecture aims to: address the blockchain trilemma of balancing scalability, security and decentralization; introduce a middleware in the architecture that enables governance for the hosting authority while maintaining some form of a decentralized system; offer scalability solutions through seamless modularity using microservices and the segregation of networking and consensus protocols; offer a secure solution that is resilient to malicious actors and offer data integrity through fault tolerance; and designing a stake-based validator selection policy that can provide economic incentives for all parties while contributing a certain degree of security for the system.

3.2 System Components and Roles

The solution comprises two main governing bodies: the validator nodes and the master node. The validator nodes are responsible for processing the transactions and reaching a consensus on the blockchain's state. Therefore, the nodes are responsible for governing the consensus protocol and ensuring that only valid transactions make it to the blockchain. For instance the consensus implementation for this paper is a simple PBFT protocol. While validator nodes are primarily responsible for executing the consensus protocol, the master node complements this by ensuring that the validator selection process upholds the network's security and fairness. The architecture leverages Kafka brokers (Fig. 3.1) to mediate communication between nodes, ensuring both isolation and redundancy. Unlike direct communication in a peer-to-peer network (Fig. 3.2), this approach mitigates vulnerabilities by preventing malicious actors from

exploiting direct channels or sending harmful messages. Kafka's resilient architecture further guarantees seamless operations, even in case of broker failures which prevents it from exposing a single point of failure.

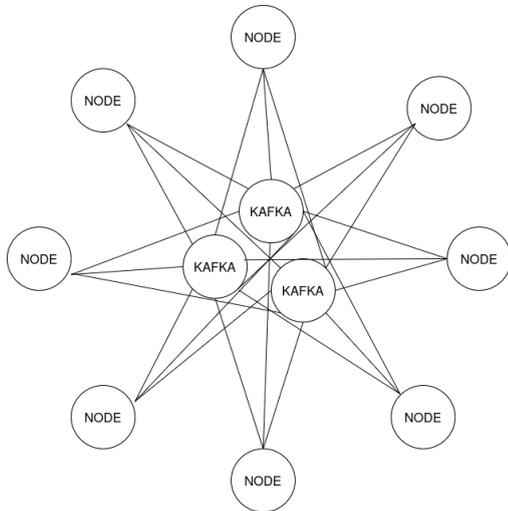


Fig 3.1

The proposed network architecture

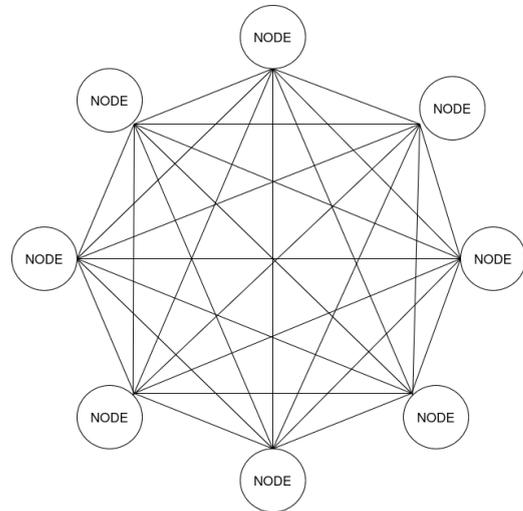


Fig 3.2

A peer-to-peer network

Beyond ensuring fault tolerance and secure communication through Kafka, the master node also plays a crucial role in managing the validator nodes, especially in their selection for consensus rounds. Unlike a traditional Proof-of-Stake system where monopolies could be made yielding control of the system to a faction of validators, in this implementation the master node can have the ability to break such monopolies by denying access to such nodes from participating in the consensus. In this way, the architecture is split into two separate components: the consensus protocol controlled by a decentralized network of validators who are responsible for data integrity, and the network protocol controlled by a centralized governing body. By balancing the decentralized data integrity with centralized network governance, the architecture achieves a harmonious blend of security and efficiency. This interdependence is further integrated through a robust Kubernetes deployment. The architecture is implemented on a single Kubernetes platform to enable seamless integration between the nodes, the master and the middleware. The Kubernetes platform not only ensures modularity by enabling each role or process to run as a separate service but also provides scalability, fault isolation, and consistent performance. By deploying containerized microservices, the architecture guarantees that failures in one service do not compromise the overall network, enhancing both reliability and security. Through this combination of decentralized consensus, centralized governance, resilient middleware, and modular deployment, the architecture achieves a secure, efficient, and fault-tolerant blockchain network that is scalable and robust against malicious actors.

3.3 Workflow and Processes

The workflow starts with transactions being submitted to the network, which are routed to the validator nodes. The validator nodes process these transactions, create a block and reach consensus through a PBFT protocol (the protocol may also be changed depending on the configuration). The master node oversees the process, ensuring secure communication and fair validator selection. Once the consensus is reached, the finalized transactions are added to the blockchain. Therefore, the workflow can be understood as three separate phases: the transaction flow, validator selection and consensus process. The transaction flow is not restricted by the architecture and can be handled separately. Transactions can be submitted directly to the validator nodes by clients for straightforward interactions. Alternatively, a separate Kafka cluster may be deployed for a pub-sub model of interactions (Fig 3.3) which may enable message queuing and ordering that would be ideal for larger deployments. Next, when the validators have prepared the block they would signal the master that they are ready for consensus by proposing their stake for the round. The master would then select a random subset of the validators to participate in the round using a verifiable random function based on the weighted probability of the share of stake each validator has in the network. Existing practices in Proof-of-Stake solutions may be employed here such as slashing, rotation of validators, staking limits, reward redistribution, etc. to further guarantee the security and decentralization of the system. Once the participants are selected for the round and broadcasted the consensus protocol begins. The participants execute the protocol as it would be in a fully decentralized system except that they communicate through the master using pub-sub interactions instead of peer-to-peer interactions. Additionally, the master also synchronizes the process by ensuring all participants are ready before proceeding to the next phase leading to consistent performance.

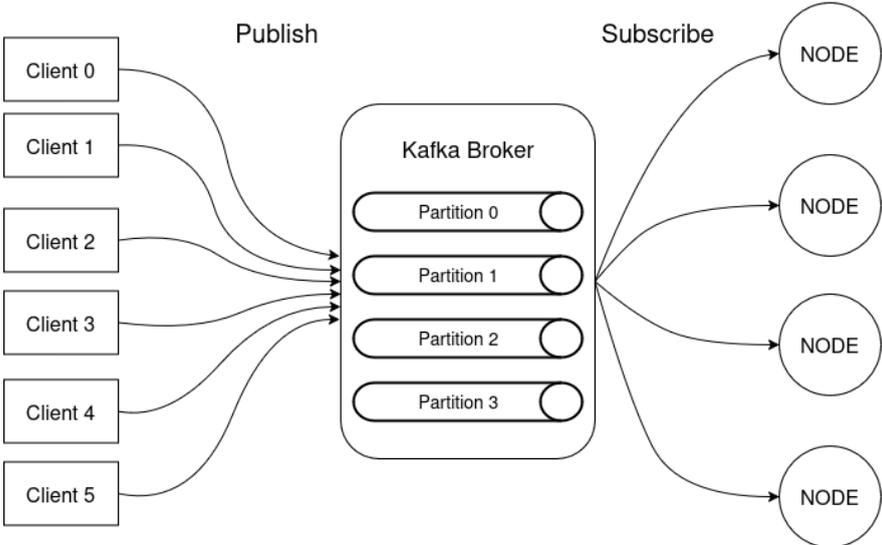


Fig 3.3

The pub-sub model for transaction messaging and communication

3.4 Key Innovations

This architecture offers a variety of innovations to the current state of hybrid blockchains. One of the most notable aspects being its hybrid approach to decentralization, achieved through the interoperability between the validator and master nodes. While the decentralized validator nodes ensure the integrity and security of data, the centralized master node ensures secure and efficient network governance by mediating all communications through a Kafka middleware. The separation of the consensus and network protocols also enables both components to scale independently. Moreover, the isolation of nodes through this networking strategy not only eliminates vulnerabilities related to direct peer-to-peer communication such as Sybil attacks but also enhances the fault-tolerance of the system by leveraging Kafka's replication and failover capabilities. Additionally, deploying the components of the architecture as containerized microservices increases the scalability by allowing components to scale independently. Furthermore, the utilization of microservices offers fault isolation by preventing failed components from compromising the system. Lastly, implementing a stake based validator selection policy helps improve trust between the validator and master nodes and opens the opportunity for economic incentives.

3.5 System Strengths and Limitations

The architecture offers increased protection against a variety of threats prevalent in direct peer-to-peer communication such as Sybil attacks or node compromise by moderating the influence of nodes across the network. By implementing a network that isolates nodes, the security of the network is enhanced significantly. Additionally, Proof-of-Stake policies for validator selection make such attacks impractical. By yielding the authority of implementing the Proof-of-Stake policy to the master node, the master can mediate in cases where monopolies are made in the decentralized network. Another notable strength of the architecture is its fault tolerance. This is achieved through the implementation of a fault tolerant middleware and containerized microservices that enable harmonious recovery and fault isolation. The architecture also accommodates growing network demands by enabling the dynamic scaling of Kafka brokers and validator nodes. This scalability is further amplified by the implementation of microservices that are easy to scale, debug and upgrade without affecting the network. The implementation of the whole architecture as containerized microservices on a Kubernetes platform ensures consistent performance across deployments and leads to an overall synchronized and coherent system. However, the architecture has its limitations as well that must be addressed. The mediation of the master node done through the Kafka middleware may introduce increased latency compared to a peer-to-peer network, especially if the Kafka cluster is misconfigured or the mediation of the master is computationally expensive. Additionally, by allowing the network and consensus to scale independently it must be necessary that the two are compatible with each other. For instance, if the validators have scaled up too much it may put strain on the network that may not have scaled accordingly. Moreover, the careful configuration of Kafka, Kubernetes and the microservices in the architecture may introduce complexity that would be a barrier in setting up or maintaining the system. The communication through a middleware introduces a

dependency on its configuration and reliability. Misconfiguration or cascading failures in the Kafka middleware could disrupt network operations causing delays, though it would not compromise the system. This is because while downtime in Kafka or the master node could disrupt immediate network or validator operations, the decentralized nodes retain the ability to fork the chain temporarily. This ensures that operations can continue independently until the issue is resolved. Once the network is restored, the system can reach consensus to reconcile the forks, preserving both data integrity and system resilience. Lastly, the system is not fully decentralized which may concern critics as it does not completely eliminate the reliance on a centralized third-party.

4. Theoretical Analyses

4.1 Security

There are several safety features in the architecture that can prevent attacks and compromise of the system. For instance, Sybil attacks pose a significant vulnerability to peer-to-peer networks, where an attacker generates multiple fake identities to gain voting power over the network and influence its decisions [14]. The architecture prevents such attacks by employing a stake-based validator selection mechanism that makes generating multiple fake identities economically expensive and impractical. Moreover, by employing additional mechanisms in this selection protocol such as validator rotation and Delegated Proof-of-Stake, having multiple fake identities does not equate to having the control over the voting mechanism. Additionally, mediating network traffic through a central master makes it difficult to generate multiple fake identities, as all communications must be authenticated by the master to ensure their legitimacy. Another common vulnerability of peer-to-peer networks is a 51% attack where a party that has accumulated majority control over the network can compromise it [15]. The proposed architecture can prevent such an attack by having the authority of breaking monopolies in the stake-based validator selection mechanism since it has the authority to mediate the network communications. Moreover, policies can be employed in the validator selection mechanism such as slashing, validator rotation, reward redistribution, etc. to further mitigate the probability of a 51% attack by making a majority indeterministic of the effects on the consensus. Moreover, having a mediated communication channel reduces the odds of a node compromise from compromising the system. For instance, if a node is compromised by being attacked by a computer virus, the master can prevent the virus from propagating across the network by restricting the compromised node from access to the network. Additionally, the use of microservices as part of the deployment ensures fault isolation hence reducing the probability that a compromised service risks the security of the whole system. The containerization of these microservices can help ensure that they have not been tampered with by using techniques like image signing and verifying the SHA-256 digest of the image. Lastly, the deployment of the architecture on a Kubernetes platform enables the system to handle faults gracefully as each pod (a single service component on Kubernetes) would have multiple replicas running that would take over the tasks as the issue is resolved.

4.2 Scalability

There are three key aspects of the architecture that enable it to be scalable: the separation consensus and network protocols handled by different entities, the deployment of the architecture as containerized microservices and the simplicity of the network compared to a complex peer-to-peer network. As discussed earlier, the separation of networking and consensus protocols enables scalability by allowing these two protocols to scale independently and easily. For instance, the consensus protocol can simply be scaled by incentivizing third-parties to participate in the network that would increase the supply of validator nodes. Alternatively, the network protocol can easily be scaled horizontally as the infrastructure is mostly network intensive rather than both network and compute intensive in the case of peer-to-peer network where nodes have to handle the load of both the networking and compute of the entire network. Additionally, the deployment of applications as containerized microservices allows bottlenecks to be identified and resolved easily without significantly compromising the system. This approach also enables modularity by allowing future extensions and upgrades to be seamlessly integrated into the architecture. Lastly, by having a simpler architecture compared to a peer-to-peer network the architecture can have improved scalability compared to a peer-to-peer network. The theoretical network complexity of a peer-to-peer network is approximately $O(n^2)$, where n is the number of nodes; in this model, every node is connected to every other node in the network, resulting in quadratic scaling [16]. In contrast, the complexity of the new architecture is approximately $O(n)$ or $O(bn)$, where n is the number of nodes and b is the number of Kafka brokers, meaning that the network scales linearly as the every node is only connected to a fixed number of Kafka brokers. Fig 4.1 illustrates the difference in complexity in both architectures as the network scales by adding more nodes. The peer-to-peer network is more feasible till around the point where the number of nodes are equal to the Kafka brokers, after which the new architecture becomes more feasible for larger deployments.

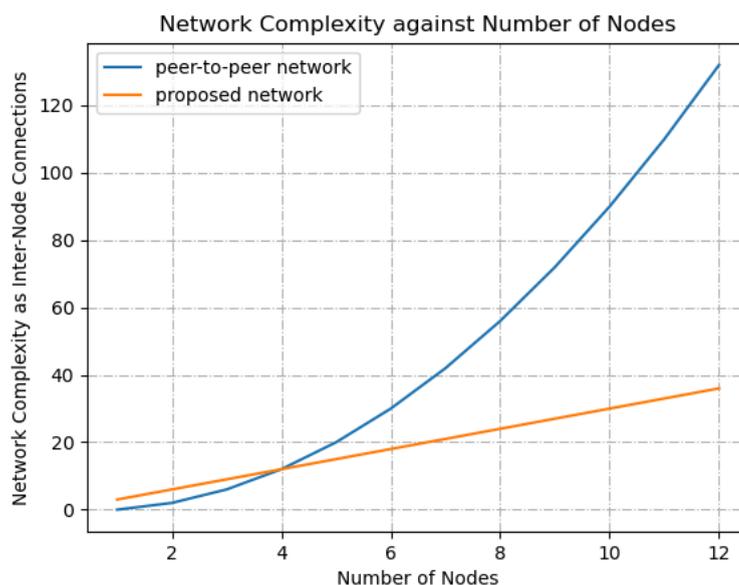


Fig 4.1

The proposed network complexity with three brokers compared to a peer-to-peer network

4.3 Governance

The architecture also enables effective governance of sensitive data without compromising too much on the benefits of decentralization. It achieves this by having a completely decentralized consensus protocol that improves the data integrity of the system that is separate from the network protocol. The network protocol on the contrary, is governed by a centralized master node that ensures the security of the network by allowing certain validators to participate in the consensus. By keeping these two protocols separate, the degree of centralization in the system is predetermined and constant. The only aspect of centralization that the system has is through the network protocol and that is known to be governed by a trusted authority. This constant, predetermined level of decentralization is in contrast to Proof-of-Stake and Proof-of-Work systems where the centralization of the system is unpredictable and is difficult to overcome once created. Moreover, the limited scope of centralization to the network protocol does not allow it to mediate or influence the decentralized consensus protocol which is a necessary guarantee for the protection of decentralization in the system. Furthermore, data privacy can be ensured by not directly revealing the identification of data to the validator nodes and rather aliasing the data using public keys that can offer anonymity. Additionally, another technique known as zero-knowledge proofs may be employed for the verification of data without having to reveal it, however, discussing the use of this technique is out of the scope for this paper.

4.4 Economic Viability

A number of features of this architecture make it economically viable when deployed in practice. The infrastructure involved in the architecture would be cheaper to scale when compared to a peer-to-peer network. This is because unlike peer-to-peer networking where nodes have to directly manage the load of the entire network by being connected to every other node, in the proposed architecture they only need to be able to handle the load from the central Kafka cluster. This leads to nodes having mostly compute-intensive infrastructure instead of both a compute-intensive and network-intensive infrastructure for the case of peer-to-peer networks making it cheaper to scale. Alternatively, the central Kafka cluster would have mostly network-intensive infrastructure and would also be cheaper to scale. Moreover, unlike permissioned or private blockchains where hosting, maintaining, and scaling an entire distributed infrastructure on its own would be extremely expensive, in the new architecture the distributed system can be scaled by incentivizing more nodes to participate in the network. Therefore, the cost and responsibility to scale the decentralized infrastructure would be handled by third-parties significantly reducing the financial burden of scaling on the hosting entities who would only have to scale the networking capabilities. Furthermore, having a stake-based validator mechanism introduces economics to the architecture providing opportunities for all parties to benefit from financially. For instance, by having stakes in the network, the validators can be rewarded by a proportional share of transaction fees from the consensus round. Alternatively, by having large pools of stakes deposited with the central authority, the central authority can benefit from the time value of money such a policy could provide. This notion is further reinforced by the assumption that if

the validators have invested in physical infrastructure for the system, it would be unlikely that their contribution would be of the short-term and that their stakes would remain in the long-term until they withdraw. Therefore, the validators can benefit from frequent cash-flows from the transaction fees and the central authority can benefit from the time-value of the stakes the validators have kept in the network.

4.5 Comparative Analysis

By comparing the proposed hybrid architecture to existing hybrid solutions that are prevalent, the architecture has a more defined and integrated approach to hybrid governance. This is because existing solutions are mostly improvised solutions such as lenient policies in permissioned blockchains or an integration of public and private blockchains. Permissioned blockchains with lenient policies are not fully decentralized solutions as they often provide leniency by allowing a trusted consortium to participate and not public validators. In contrast, the proposed architecture allows for the participation of public validators and is fully decentralized, but the decentralization is limited to the consensus protocol only. The other network protocol is centralized, allowing the hosting entity to govern its sensitive data effectively, but also fault-tolerant to prevent a single point of failure. While the lenient permissioned blockchain solution offers a spectrum of solutions with varying intensity of decentralization that apply to the entire blockchain, the proposed architecture clearly defines the centralized and decentralized aspects of the blockchain. The other prevalent solution of trying to merge public blockchains and private blockchains risks exposing vulnerabilities or creating performance bottlenecks at the integration points of the two blockchains. Such mergers focus on partly processing the data in private chains and the rest in public chains or vice versa. Since the two public and private blockchains operate at fundamentally different principles, they are not very compatible with each other leading to issues in integration to arise. The proposed architecture however, offers a more intricately and cohesively integrated solution where the network and consensus protocols are inter-dependent and both are used as a medium for the whole process of processing the data. The integration of these protocols as containerized microservices further strengthens the cohesiveness of the architecture reducing the risk of compromising the whole system.

4.6 Potential Bottlenecks and Limitations

The use of Kafka as a middleware for communications between validator nodes may introduce some additional network overhead that would not be prevalent in a peer-to-peer network. Whether this additional overhead would lead to significantly degraded performance would have to be tested. Kafka is known for high throughput and low-latency data streaming and is well optimized to reduce the effects of degraded performance from additional network overhead. It is also possible that the proposed architecture has improved performance compared to peer-to-peer networks. This is because in peer-to-peer networks all validators would have to receive and process messages independently that may cause more variation in performance due to the variation in network overhead each validator experiences. In contrast, broadcasting to Kafka would offer a more consistent platform for networking that may reduce the synchronization time among nodes causing better performance. Another potential

limitation is the centralization of the networking protocol. The central Kafka cluster would need to have an adequate number of brokers to offer significant fault tolerance to prevent the effect of a single point of failure. A significantly lower number of Kafka brokers compared to the number of participants would not only have degraded performance but a higher risk of having a single point of failure.

5. Implementation

5.1 Scope of Evaluation

For the purpose of this paper, the scope of evaluation for this implementation is the scalability and performance of the architecture. By evaluating scalability this paper aims to gauge how well the architecture can sustain increasing workload requirements. Similarly, by evaluating the performance of the architecture this paper aims to determine the practicality of the architecture using metrics like throughput and latency. This limited scope is chosen to validate the architecture's feasibility for large-scale and practical deployments which would require even tougher requirements. Moreover, testing aspects like security, fault tolerance or economic incentives would require specialized frameworks and methodologies which are out of the scope of this research, though some of these aspects are touched upon briefly by this implementation. For instance, the security and fault tolerance of the architecture is tested by simulating fraudulent transactions to check if the system correctly identifies them and how well it can gracefully continue its operations. But comprehensive analysis, such as simulated Sybil or 51% attacks or intentional failure of Kafka nodes or validator nodes is not tested. Rigorous testing and evaluation of the security, fault tolerance and economic incentives of the architecture could be the foundation of any future work to test the system's practicality.

5.2 Simulation Environment and Tools

The architecture is deployed on a local Kubernetes cluster as containerized microservices. Communication between the services is done through exposing the external network port (9092) instead of the internal port (29092). This is done intentionally to force communications to route through additional network layers such as the host network stack rather than directly within the container network which is optimized for internal communication. By doing so additional latency is introduced which may be similar to latency in practical situations by utilizing realistic network overhead making the test more realistic overall. The microservices are deployed as containerized images as it would be the case in a realistic setup. The applications themselves are programmed using Rust, a low-level systems programming language. This programming language was chosen because it offers zero cost abstractions which are necessary for testing the performance as abstraction overheads such as garbage collectors, interpreters and JVM may impact performance significantly making observations of performance inaccurate in gauging the architecture's potential. Additionally, the language offers excellent concurrency support which is necessary for running processes concurrently and safely. Lastly, the memory safety the language offers ensures secure and reliable deployments free of memory related vulnerabilities. The Kubernetes in its entirety is deployed on a lightweight consumer-grade laptop. The relevant hardware specifications of this device is

a Ryzen 5 7535 HS processor which has 6 CPU cores, 12 threads, 3.3GHz base and 4.6 GHz turbo clock speeds. Additionally, the device has 32 GB of DDR5 RAM with transfer speeds of up to 4800 MT/s. Since multiple nodes are simulated which spawn multiple processes, and that the number of CPU threads are limited to 12, the tasks are designed to run asynchronously. This however, introduces a run-time dependency to handle the execution of multiple threads that may impact performance slightly. This is important to note as there may be some additional performance cost due to the restrictions of the setup. Moreover, any realistic deployment would have better resource isolation and likely leverage true parallelism instead of asynchronous process management which would improve performance significantly.

5.3 Experimental Design

By conducting this experiment it is hypothesized given the nature of the architecture that the architecture should be able to scale linearly with impact on performance being minimal as more nodes are added to the network. In order to test this hypothesis, the number of participating nodes will be used as the independent variable which would be varied to observe its impact on the dependent variables which are the performance metrics. To ensure the isolation of the effect of the independent variable the control variables are kept constant. These control variables include the amount of CPU and memory resources each node is allocated, the same network hardware and configurations for all nodes and the same application for each node to execute. The control group for the experiment are the master node and the central Kafka cluster and would not be modified among experiment runs. The treatment group would be the validator nodes as the number of network participants will be scaled to test how well the network can handle the increased load. Aspects of the experiment that would be randomized are the assignment of users and their key-pairs, the transactions performed, the stakes proposed and the number of fraudulent transactions simulated. The transaction model followed is an account based model and each node is responsible for keeping track of each user's balance. The sample size of data collected is 130 blocks of 512 transactions or 66560 transactions to allow for ample data to draw generalized conclusions of the architecture's performance. The performance data is collected by timing different operation phases within the node's application and are written to a CSV file. The CSV file is extracted when the experiment is concluded and is analyzed to gauge the performance of the architecture.

5.4 Experimental Configurations

There are a number of components for the experiment that can be configured. For instance, for the nodes in the network the transaction throughput, batch size for the transmission of messages, block size and the number of validators to be chosen for each round. For the purposes of this experiment the transactions are transmitted in batches of 64 with the block size being 512 transactions. These settings can be configured by mounting configuration maps to the pods in Kubernetes. Moreover, Kubernetes also offers some configurations that can be tweaked to affect performance. A total of 10 CPU threads and 30 GB RAM are allocated to the local Kubernetes cluster. The exact distribution of these resources are presented in table

5.1. Additionally, the Kafka cluster, consumers and producers also have a number of configurations that can be used to balance aspects of networking such as throughput and latency. Table 5.2 presents some of the primary configurations that Kafka has to offer that affect performance significantly along with their purpose and setting for this experiment.

Service	Minimum Resource Limit	Maximum Resource Limit
Validator Nodes	2 CPU threads and 4 GB RAM	3 CPU threads and 6 GB RAM
Master Node	1 CPU thread and 2 GB RAM	2 CPU threads and 4 GB RAM
Transaction Simulator Pod	1 CPU thread and 2 GB RAM	2 CPU threads and 4 GB RAM
Kafka Pod	1 CPU thread and 2 GB RAM	2 CPU threads and 4 GB RAM
Zookeeper Pod	1 CPU thread and 2 GB RAM	2 CPU threads and 4 GB RAM

*Table 5.1
The Kubernetes services and their corresponding minimum and maximum resource limits requests*

Kafka Configuration	Purpose	Setting
linger.ms	How long the producer waits for messages to accumulate before sending them	10
min.batch.size	The minimum batch size of messages to be accumulated before being sent or received	64000
compression.type	The message compression to be used to transmit messages across the network	lz4
acks	The level of acknowledgement to be reached among partitions before sending or receiving messages (a higher setting ensures greater data integrity)	1
Number of Kafka Brokers	The number of Kafka brokers in the network to set the degree of fault tolerance	1
Number of Topic Partitions	The number of partitions for each topic (a large number of partitions may enable to parallel transmission of messages)	5

*Table 5.2
The different Kafka configuration settings, their purpose and their setting for this experiment*

5.5 Metrics for Evaluation

Several metrics are used for the evaluation of the performance and scalability of the architecture. To gauge the throughput of the architecture two particular metrics are used: the processing rate for individual transactions (transaction TPS) and the processing rate for transactions in block creation and consensus (block TPS). For the purpose of this paper the block TPS is more indicative of the architecture's improved performance as it evaluates the performance of the new network and consensus protocol, an innovation offered in this paper. However, both metrics are limited by resource constraints and the inability to leverage parallel processing that can significantly optimize performance, an important limitation to note. Moreover, the introduction of a runtime overhead due to asynchronous processing of tasks may also incur some effect on the performance. To evaluate the network latency and synchronization delays, the time taken for all the nodes to synchronize before each phase is also noted. Additionally, the time taken for each consensus phase to execute is noted to identify bottlenecks in the architecture. Another key metric, Time to Finality (TTF) is also measured which notes the maximum time it takes for a transaction to be committed to a block since its inception. This metric is aimed to evaluate the architecture's performance from a user's perspective to ensure the performance is practical enough to be deployed as a more convenient system for secure transactions.

5.6 Limitations and Assumptions

The scalability analysis of this experiment is limited by the consumer-grade hardware used in the simulation, which lacks the resource isolation and parallelism offered by dedicated server deployments. This may result in sub-optimal performance metrics that are not indicative of practical deployments. Additionally, deploying the architecture locally may not account for actual distributed network limitations such as varying geographical locations, bandwidth limitations, and high-latency links. There are also several limitations on the performance such as the run-time overhead discussed earlier for asynchronous processing. Moreover, the Kafka cluster configuration is intentionally simplified with one broker which provides a baseline but does not fully capture the trade-offs between fault-tolerance and performance in multi-broker, high-partition setup. The fixed resource allocation for the Kubernetes pods also introduces a limitation over dynamic scaling, which could improve throughput and latency in practical situations. Lastly, certain aspects of the architecture are not tested comprehensively enough such as the security and economic incentives which limits the scope of this experiment. There are also some assumptions under which the experiment relies on. All the nodes in the network are assumed to be homogeneous with similar hardware and network capabilities which may not be true in a practical deployment. Additionally, the transactions are assumed to be generated uniformly in batches of 64 and having a constant throughput. This assumption may not hold in actual deployments where transactions are produced randomly with varying throughput in peak and off-peak hours. Lastly, the configurations of the experiment are assumed to be kept constant. Certain configuration parameters such as the transaction batch size, block size and Kafka configurations may have to be adjusted dynamically to account for the difference in workloads.

6. Evaluation and Results

6.1 Evaluation Overview

Given this setup's hardware constraints, the experiment yielded excellent results. The average Block TPS is around 1030 transactions per second, a good throughput figure for the consensus performance. The pool processing throughput has been around 412 transactions per second, which is better than anticipated, given that parallel processing could not be utilized. The median time to finality has been 2.2 seconds, which is solid performance considering users would only have to wait around 2.2 seconds before their transaction gets included in the blockchain. As shown in Table 6.1, the block consensus time is approximately 500 ms, with synchronization time accounting for just 43 ms, showcasing efficient node communication. Across scaled deployments, this figure has only marginally decreased, which is favorable for scalability as scaling the number of nodes does not impact performance significantly enough.

6.2 Descriptive Statistics

Table 6.1 presents the five-number summary of the average performance metrics of the three deployments. The average time for the entire process was 2.3 seconds, with the time to finality being 2.2 seconds. Most of this time was spent pooling and processing transactions, contributing roughly 1.8 seconds to the total processing time, whereas the time spent in consensus contributes roughly 0.5 seconds. The pre-prepare and prepare phases of the consensus contribute to 43% of the total consensus time each. Table 6.2 presents the percentage change in the consensus performance among scaled deployments. Across the deployments, the consensus performance has changed negligibly, even with the shift from single-node to multi-node deployment. This can be credited to the resource isolation of the architecture that guarantees consistent performance. The commit time is an exception, where the highest percentage change was observed. However, this can be attributed to the fact that it was already negligibly low and barely affected the overall performance.

	Pool TPS	Block TPS	Pooling Time (ms)	Consensus Time (ms)	Sync Time (ms)	Total Time (ms)	Time to Finality (ms)
Min	406.55	1016.45	1559.75	491.42	41.17	2063.44	1998.25
Q1	407.07	1028.98	1566.42	495.17	42.37	2071.47	2005.63
Median	407.45	1031.83	1756.60	496.28	43.01	2263.51	2196.29
Q3	407.87	1034.11	2591.76	497.69	43.93	3097.58	3030.37
Max	409.39	1042.08	9824.61	504.06	51.81	10339.69	10271.36

Table 6.1

The 5 number summary of the architecture's performance

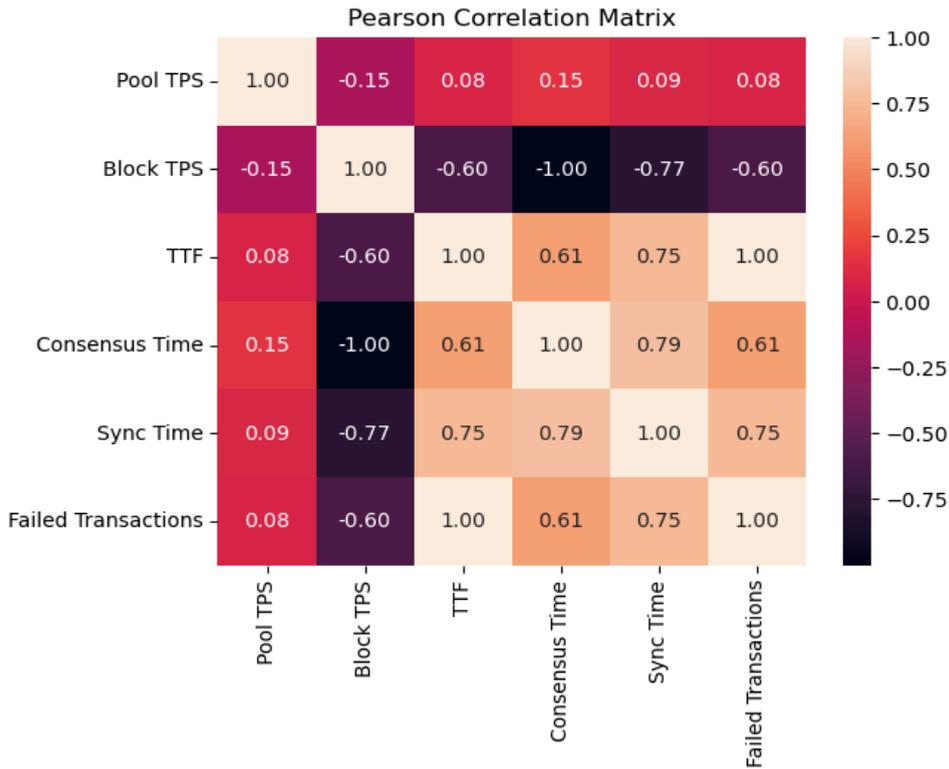
	Block TPS	Preprepare Time	Prepare Time	Commit Time	Sync Time	Consensus Time
Single Node to 3-Node	-1.83%	+1.15%	+0.04%	+77.36%	+8.20%	+1.89%
3-Node to 4-Node	-1.10%	-0.15%	+0.39%	+25.48%	+1.57%	+1.12%

Table 6.2

The percentage change in consensus performance among scaled deployments

6.3 Correlation Analysis

Fig 6.1 presents the correlation matrix of the average performance metrics among all scaled deployments. The strongest positive correlation among the performances is among the Time to Finality, Consensus Time, Synchronization Time, and the number of failed transactions detected. The perfect correlation between the time to finality and number of failed transactions indicates that an increase in fraudulent transactions directly delays finalizing valid transactions. This is because fraudulent transactions increase the processing load before a valid block can be created. The number of failed transactions, synchronization time, and consensus time also have a strong correlation, though it is not perfect. An explanation for this is that as the number of fraudulent transactions in the system increases, the time taken for nodes to synchronize among one another also increases due to increased load on the nodes, which causes an increase in the overall consensus time. Moreover, any notable negative correlation among the metrics seems to be between the Block TPS and the consensus time, synchronization time, Time to Finality, and the number of failed transactions. The explanation for the negative correlation between the block throughput and the consensus and synchronization times is relatively straightforward. As the time for block consensus decreases, the speed at which that block can be committed increases. The negative correlation of the block throughput with the number of failed transactions stems from the fact that an increase in fraudulent transactions causes an increase in synchronization time among nodes. This increase delays the total consensus time and causes a decreased block throughput. The negative correlation between the Time to Finality and the block throughput is because as the throughput decreases, the time taken for a transaction to be committed to a block increases, which causes an increased Time to Finality.



*Fig 6.1
Heatmap of the Pearson Correlation Matrix of the average performance metrics among all scaled deployments*

6.4 Potential Bottlenecks and Optimization Insights

Transaction processing and verification are significant bottlenecks in this experiment. On average, roughly 83% of the total processing time for the blockchain is spent on pooling and processing the transactions. The pool processing throughput is roughly 60% slower than the block creation and consensus throughput. While a lot of this performance setback is attributed to the lack of parallel processing of transactions, any future implementation should ensure adequate resources to enable parallel processing to overcome this bottleneck. Another identified bottleneck is the processing time for block creation and validation. In particular, the time taken for the prepare and pre-prepare phases is around 200 milliseconds each, which accounts for 84% of the consensus time. Most of this degraded performance is attributed to the resource constraints of the setup since both these phases include the computationally expensive processes of Merkle root generation, block creation, and hashing, which can be improved significantly using parallel processing.

In contrast, the commit phase, where votes are aggregated, and blocks are compared and committed, only took an average of 13 milliseconds. Lastly, the time to finality becomes a significant bottleneck in cases where there are a significant number of fraudulent transactions. The time to finality can take up to 4 to 6 seconds for cases where most transactions are fraudulent (three times the number of valid transactions in this case). While such situations occurring in practice by chance are rare, it would be the case in a DDoS (Distributed Denial

of Service) attack aimed at slowing down the system. Although such an attack would be highly impractical to implement in practice (at this particular point of the architecture, the transactions are most distributed), this still exposes a vulnerability in the implementation that should be addressed before practical deployment.

6.5 Trend Analysis

Fig 6.2 shows the distribution of the block throughput among the three deployments. The distribution median decreases as the number of nodes participating in the consensus increases. This indicates a decreasing trend in block throughput as the number of participants in the network increases. The decreasing trend is attributed primarily to the significant increase in the commit time across the deployments, referring back to Table 6.2. The secondary contributor to this trend is the increase in the synchronization time among the nodes, which causes a decrease in block throughput. Moreover, as the number of participants in the network increases, so does the variability in the block throughput, with the deployment of 4 nodes having the highest variability. This variability can be attributed to the fact that with more nodes, there is a higher degree of uncertainty with which they can execute the processes simultaneously, which causes higher variability in synchronization time and, thus, the block throughput.

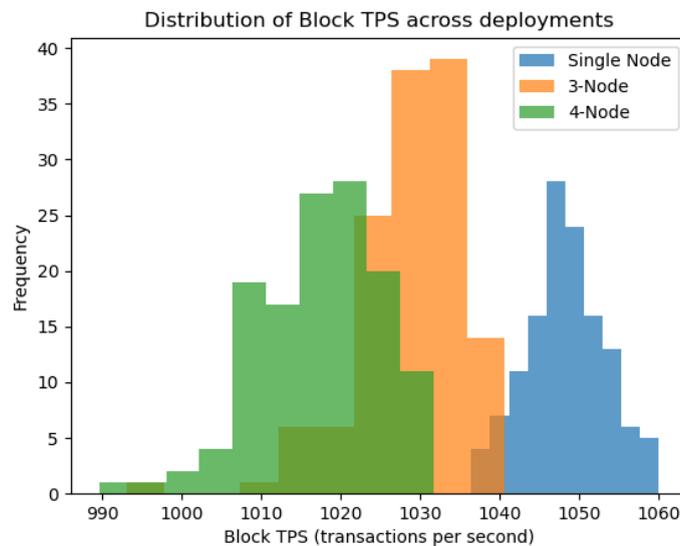


Fig 6.2

A histogram showing the distribution of Block throughput across the three deployments

Fig 6.3 presents the trend that the block throughput follows across the consensus rounds in different deployments. The distribution is stationary, with no trend in increasing mean while progressing through the consensus rounds. This highlights the consistent performance the deployment offers across consensus rounds with varying stress levels, such as an increasing number of fraudulent transactions. This means that the throughput is independent of the stress put on the system. There is, however, a clear distinction in the block throughput among

different deployments. The block throughput experiences a gradual decrease as the number of participating nodes in the deployment increases.

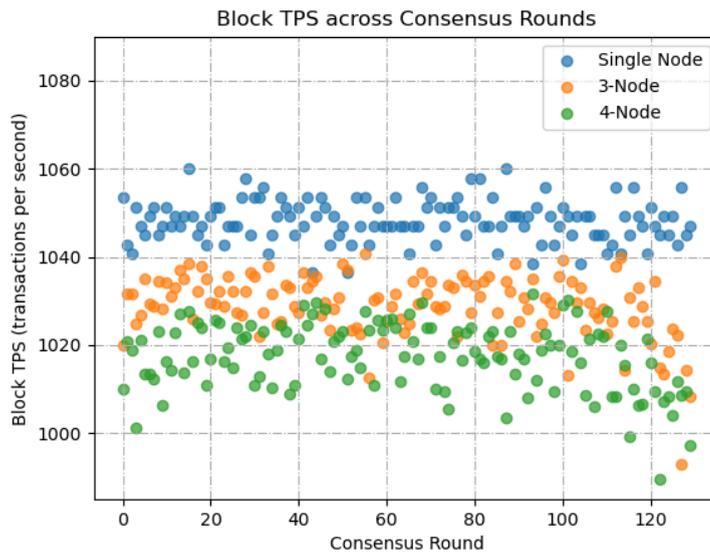


Fig 6.3

The block throughput across consensus rounds and deployments

Fig 6.4 compares the trend of the time to finality with the trend of the number of failed transactions across consensus rounds. As observed, the time to finality follows an identical trend to the number of failed transactions. This highlights how directly the two are related to each other. For this experiment, an exponential increase in fraudulent transactions led to an exponential increase in the time to finalize the transactions.

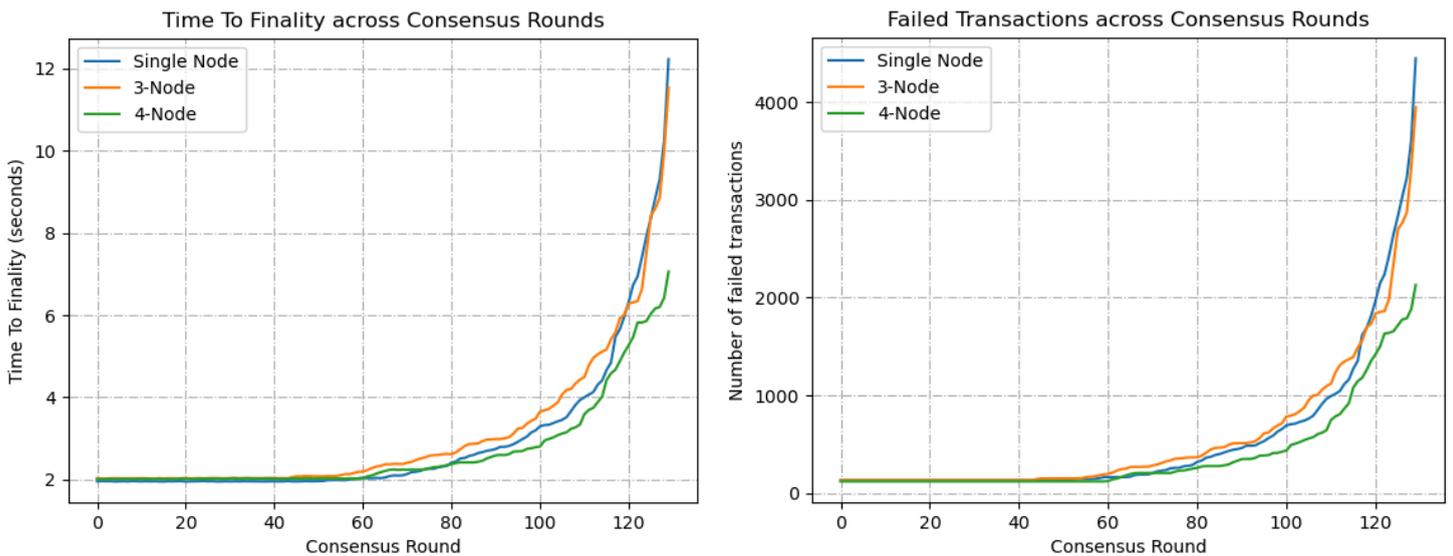


Fig 6.4

Time to Finality across consensus rounds compared to the number of failed transactions across consensus rounds

6.7 Conclusion and Implications

Overall, evaluating the architecture's performance and scalability has shown promising results. The new consensus and network protocol yields an average block throughput of 1030 transactions per second. This indicates a robust baseline for the architecture's throughput, especially given the constraints of the hardware setup. The median time to finality of 2.2 seconds is also noteworthy, as it suggests that users can expect prompt verification and completion of their transactions, which is essential for a convenient user experience and any practical deployment. The architecture's ability to scale without significantly impacting key performance metrics also indicates its potential for scalability. Most of the metrics have been affected by less than 1% across scaled deployments, indicating the architecture's robustness under increasing stress of scaled deployments. The consistent performance is not only among scaled deployments but also among consecutive consensus rounds, indicating a well-tested performance benchmark. However, certain observations have also been made that reinforce the claim of this paper that this implementation is a proof-of-concept and not a ready-to-deploy implementation. The significantly lower transaction pool processing throughput and the significant amount of time that the prepare and pre-prepare phases take of the total consensus time indicate potential optimization areas. Moreover, the potential area of vulnerability for DDoS attacks discussed earlier must also be resolved in any future work focusing on the security of the architecture. The implementation also needs to be stressed for scalability till the point of failure (granted there are enough hardware resources) to find the limit to which the architecture can sustain the load, and trends in scalability tend not to apply. Such a test would demonstrate the scalability of the architecture more comprehensively. In conclusion, this proof-of-concept implementation has remained successful as it has demonstrated the architecture's excellent performance and scalability, whose assessment was the focus of this implementation. The results of this implementation highlight the architecture's worthiness for future work to be conducted on it. In the scheme of extensive testing that this architecture has to go through before practical deployment, it has indeed passed the first test with excellent results.

7. Discussion

7.1 Key Observations

The key observations to be noted from the results of this experiment is that the proposed architecture's implementation is lightweight, high-performing and highly scalable. The results back this claim as the validator nodes have performed exceptionally while being provided with only a single CPU core to process transactions and reach consensus proving that the solution is lightweight. Moreover, the high block throughput figure of 1030 transactions per second indicates the architecture's potential for application where high transaction throughput is key. Lastly, the minimal change in block throughput and synchronization time among nodes across scaled deployments indicates the architecture's scalability without significantly degraded performance.

7.2 Implementation Challenges

While the results of the implementation and the architecture itself have rightfully been discussed extensively, the discussion of challenges experienced during the process of the implementation are also equally necessary. The two particularly notable challenges during development were data races and the Kubernetes deployment setup. The most challenging data race issues were not within the source code but within the implementation of the architecture using Kafka particularly in multi-node implementations. There were a variety of issues such as the difference in the scheduling of tasks, tasks not being executed exactly at the same instant in time and broadcasting of messages in channels that are not actively listened in on yet. The lack of task scheduling led to some cases where tasks were executed completely before their prior tasks that needed to provide the necessary data leading to a deadlock. This was resolved by implementing an automaton to schedule tasks in order, shown in Fig 7.1. This automaton ensured that the workflow was executed sequentially with the next task not starting before the prior tasks had started. The next issue that was causing the data race was the fact that tasks were not executing exactly at the same time that caused a different state of data being obtained. For instance, this caused a difference in block timestamps of the blocks created by the validators if half the nodes created the block in one second and the rest in the other second. Since the timestamps were hashed to form the block hash, this led to different block hashes and a failure to reach consensus. This issue was resolved by using a more general timeframe such as the timestamp in multiples of 5 seconds that allowed for less room for different timestamps. Lastly, the issue of messages being broadcasted in channels not being listened on was caused by the issue that some nodes moved ahead in the consensus and broadcasted pre-prepare or prepare messages before other nodes could reach that phase and actively listen on those channels. This issue was resolved by opening a messaging channel for communication among nodes that synchronized the consensus among them and ensured no node progressed before all other nodes were ready to progress within a given timeout.

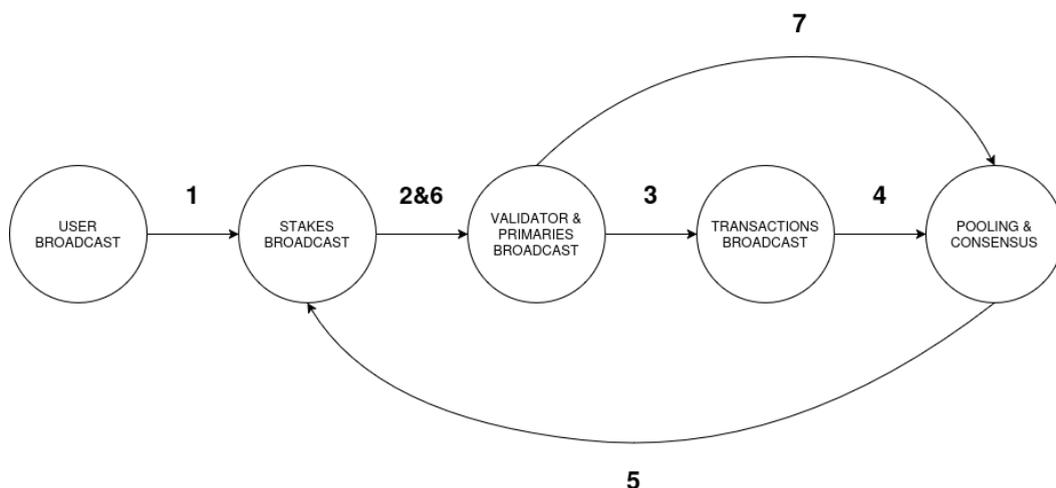


Fig 7.1

The automaton implemented to schedule tasks and ensure the correct order of execution

The implementation of the architecture on Kubernetes was also an issue that took up some time. In particular, deciding the adequate amount of resources to allocate to each node required some repeated trial and error to find the right amount of resources without starting to degrade performance. Moreover, the implementation itself led to another form of race condition where if all the pods were deployed simultaneously, some would begin running before the others and since the pods were designed to be executed in a sequential order it led to some issues. This issue was resolved by ensuring certain pods had started running before others were deployed to ensure the entire deployment was successful.

7.3 Broader Context and Comparison

The high block throughput speeds of the architecture indicate its potential to be used in practical situations where high operational throughput is necessary. For context, VisaNet processes 1667 transactions globally during typical operations [17]. The proof-of-concept implementation approaching Visa’s average operational throughput indicates its substantial potential for handling real-world transaction loads. Another example is PayPal, which typically handles 193 transactions per second globally [18]. The implementation’s throughput of 1030 transactions per second significantly outperforms this benchmark, suggesting enhanced capability of processing transactions at scale. Table 7.1 compares the performance of the implementation to some common benchmarks in the blockchain ecosystem. As it can be observed the architecture clearly outperforms Bitcoin and Ethereum in block throughput and time to finality indicating better suitability to be deployed at scale. The performance observed is similar to that of high-performing blockchains like Ripple and SKALE, indicating it is a viable proposal for this architecture. The proposal appears to have a better balance between time to finality and block throughput as Ripple and SKALE have high performance in a certain benchmark with some compromise on the other. Overall, the architecture meets the performance requirements of large-scale deployments and can offer competitive benchmarks to industry standard solutions.

Architecture	Block Throughput (transactions per second)	Time to Finality
Proposed Architecture	1030	~ 2.2 seconds
Bitcoin	7	~ 1 hour
Ethereum 1.0	15-25	~ 6 minutes
Ripple (XRP)	1500	~ 4 seconds
SKALE Network	400 - 700	~ 1.46 seconds

Table 7.1

Comparison of performance benchmarks with industry standard solutions [19] [20]

7.4 Implications for Scalability

In section 4, it was broadly discussed that the architecture would likely have high scalability because of two reasons: the separation of network and consensus as separate applications and the use of a microservices architecture that can scale deployments easily. The results from the implementation and the experiences during development both validate this claim. As it was observed from the experiment, there was negligible impact on performance and block throughput as the nodes were scaled. Referring back to table 6.2, the impact on performance by adding additional nodes was around 1% which can be considered as negligible change. This is because, by separating the consensus nodes and the network nodes, the network was able to handle increased throughput due to better resource isolation and utilization. If it was the case for a peer-to-peer network where such isolation may not exist, then it is possible that the two protocols would have to compete for resources under stressed environments leading to further degraded performance. Moreover, the results validate the notion that Kafka would be a high-throughput message streaming platform as despite the additional overhead of Kafka, most of the time utilized is in computing rather than inter-node communication and synchronization. This is because the time spent in inter-node communication and synchronization only accounts for 43 milliseconds of the total 490 milliseconds of consensus time. The scalability of architecture due to the microservices architecture has been highlighted during the development phase. During the debugging phase, it was easier to identify issues in the implementation using the microservices architecture than it would be with a monolithic architecture. This is because the use of microservices easily pointed out areas where faults would occur as certain services would fail prompting a focus to that particular service. Apart from debugging, application development and adding features once the implementation was finished was also simpler. This is because the development could be focused on implementing certain features without being concerned how they would be able to integrate with the whole implementation. Moreover, despite the obstacle of initial deployment on Kubernetes, it was relatively straightforward to scale the resources and replicas of the services once the setup was complete. Scaling was as simple as changing a number in the configuration files of the deployed pods or simply running a command on the terminal. Overall, the theoretical analysis was accurate in identifying the aspects of the blockchain that would improve scalability.

7.5 Key Takeaways

Choosing an efficient consensus algorithm was a pivotal decision in achieving high throughput and low latency in the implementation of the experiment. The choice of Practical Byzantine Fault Tolerance (PBFT) as the consensus algorithm was not only appropriate given the resource constraints of the hardware, but also an excellent opportunity to test the new architecture's effectiveness in networking and communication. This is because PBFT requires substantial inter-node communication during the consensus process that would be just enough to stress the Kafka network and get accurate data on its performance. Another critical decision during this implementation phase was the use of asynchronous processing for tasks. This ensured that the CPU spent the least possible time being idle ensuring efficiency. Moreover, using asynchronous processing was the closest the implementation could get to resembling

true parallelism in a practical deployment given the hardware constraints. It allowed for multiple threads to run concurrently regardless of the fact if there were enough CPU cores to execute them simultaneously. During development, the choice of using a microservices architecture was helpful in easily developing, debugging and deploying the implementation. Moreover, adding features and services to the implementation was easier this way and the architecture was more cohesive and integrated. The deployment of these microservices in a Kubernetes platform offered consistently replicable environments with configurability that is necessary in the context of high performance computing and experimentation. A crucial decision during the development of the new architecture was the choice to use Kafka as the inter-node communication middleware instead of utilizing a peer-to-peer network. This was an important decision as Kafka is a dedicated solution to high-throughput communication that has excellent scalability and configurability for a variety of use cases. This solution also led to the efficient utilization of network usage as the networking was not strained on the nodes and instead outsourced to a system that was optimized to handle such network loads.

7.6 Takeaways for Future Development

Before any future work is done on this architecture, it is important to learn from the lessons of this paper's implementation to speed up development and improve effectiveness. The most crucial takeaway from the implementation challenges in section 7.2 was to have an idea of task scheduling and orchestration ready before development. The development of the architecture as microservices led to this oversight as applications were developed independently without in-depth consideration of how they would integrate together and execute the entire process. This oversight cost a lack of synchronization, data races and challenges in deployment that required major refactoring in the code base of the applications. Having a superficial idea of the orchestration of the services in the context of the whole system helped in developing the architecture more coherently. Moreover, before deploying a new service or feature on Kubernetes, it is necessary to test the application for bugs natively first. The testing workflow should be as follows: test the application itself to check if it has any bugs, test the application in a native single-node deployment to check if it integrates well with the Kafka middleware and other services and then finally test the application in a native multi-node deployment to check if it can integrate and synchronize with other nodes in the system. Using this framework tends to be easier in identifying the issues and resolving them effectively and timely. It is important to note that while the microservices architecture allows for applications to work independently and isolate faults, considerations on how the services will collaborate to orchestrate a system needs to be taken into account to have a cohesive implementation.

7.7 Broader Implications

The proposal of the architecture and its development and implementation as a viable proof-of-concept solution has introduced a novel solution to the academic state of hybrid blockchains. Application of hybrid blockchains is particularly in demand where adequate governance is needed with the data integrity, fault tolerance and trust of public blockchains. Such applications include governments and enterprises where transparency of data can be a

requirement or highly preferable. For instance, if a Central Bank Digital Currency would be implemented it would be highly preferable, if not a requirement, to ensure data transparency, but at the same time effective governance will also be necessary for the protection of personal data. In such an application, allowing public validators to participate in the process would not only align with principles of democracy but also with the right to the freedom of information whereas the control of the network protocol by a trusted government entity would ensure adequate governance. The use of a stake-based validator selection model opens the opportunity for economic incentives that would be beneficial for both the hosting entity and the participants. Furthermore, the use of Apache Kafka for managing the communication protocol in a consensus is a novel contribution that introduces the discussion to utilize modern distributed frameworks more extensively in the blockchain ecosystem. The use of Kafka and other distributed frameworks in solving issues related to scalability, performance and governance can contribute substantially to the current understanding of possible solutions to address them. The use of microservices in deploying the entire architecture has also expanded the possibility of using microservices in the blockchain ecosystem. Prior blockchain implementations had seen limited application of microservices which did not provide substantial evidence for the advantages of utilizing them in a broader scope.

8. Conclusion

8.1 Summary

The purpose of this paper was to develop a hybrid blockchain architecture that solves the issues related to scalability, governance and economic viability effectively. Solving these issues is necessary for the deployment of blockchains in government and enterprise applications. Current hybrid blockchain solutions could not address all these issues effectively by compromising on certain aspects. These solutions were mostly improvised solutions on top of existing mainstream permissioned and public blockchains. Therefore, there was a need to develop a new blockchain architecture that would aim to tackle the issues related to scalability, governance and economic viability from its foundation. There were significant gaps in research and applications that utilized the solutions that the proposal offered. For instance, the implementation of Kafka as a middleware solution to manage communication among decentralized and isolated nodes, and the use of a microservices architecture to enable fault isolation and scalability. By proposing, developing, implementing and testing a comprehensive case was made for the viability for the architecture in practical use cases and providing a viable proof-of-concept to work on. The key findings of the implementation having a block throughput of 1030 transactions per second, 2.2 second time to finality and a 43 millisecond synchronization time across scaled deployments highlight the architecture's potential for practical application and effective scalability and performance

8.2 Success of Objectives

To recap, the research objectives of this paper were to: introduction of a semi-centralized blockchain for effective governance and transparency, separation of network and consensus protocols to enable independent scalability, introduction of the microservices architecture for

the blockchain applications, an implementation of a stake-based validator selection mechanism for economic incentives and a demonstration of the architecture as a viable proof-of-concept solution. The semi-centralized model with effective governance and transparency was achieved by proposing a centralized network protocol for inter-node communication and selection that offered governance of data, and having a decentralized consensus protocol of public validators that enabled transparency. This separation also fulfilled the second objective of separating the network and consensus protocols for independent scalability. This was implemented by introducing a Kafka middleware that handled communication between the nodes and offering effective governance of data. This implementation also enabled scalability as the Kafka brokers could be scaled easily to match the requirements of the network. The microservices architecture was achieved by developing the blockchain applications as modular, containerized services and deploying them on a Kubernetes platform. This enabled fault isolation and individual scalability along with consistent performance due to the highly replicable environments offered by both Kubernetes and containerization. A stake-based validator selection model was also implemented by yielding the authority to a central master node governed by the hosting entity. This provided economic incentives for both the host and the participants as the host could benefit from the time-value of the stakes and the participants could get frequent cash flows of transaction fees. Lastly, the architecture was also practically developed and implemented to test its potential for practical adoption. The results obtained from the experiment showcased the architecture's excellent scalability and performance proving it to be a viable proof-of-concept solution.

8.3 Future Work

There are mainly three effective directions for future work on this proposal: test the architecture more comprehensively, test other aspects of the architecture or to develop further variations of this architecture tailored to certain use cases or improve on the current implementation. To test the architecture more comprehensively, it needs to be optimized with all performance bottlenecks addressed adequately. For instance, the bottleneck identified for transaction processing must be addressed by enabling parallel processing or other solutions. Moreover, adequate resources must be provided to ensure applications do not compete for resources causing degraded performance. Any new tests should be conducted on a dedicated distributed environment with resource isolation to better replicate the environment in practical use cases. Next, some of the other aspects of the architecture that can be tested are its security, governance, and economic viability that have not been tested in depth in this paper. The testing of these aspects may require different frameworks discussed here such as vulnerability reports for testing security or field experiments to test the economic viability and governance. For instance, some of the tests for security may require simulating Sybil and 51% attacks, penetration testing and testing the architectures ability to recover from failures. Lastly, certain variations or extensions can be made to the architecture for certain improvements or use cases. For instance, an interesting variation that could be discussed is the possibility of the central network protocol being decentralized. This would mean that the architecture would be split into separate consensus and network protocols, both of which are being governed by two distinct decentralized entities that have different incentives in common. Some questions to ask

in such an implementation might be to discuss whether such an implementation would truly be decentralized and whether the difference in both the entities' incentives be enough to guarantee honesty in the system and prevent any centralization in the network.

References

- [1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>. [Accessed: Jan. 6, 2025].
- [2] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," 2013. [Online]. Available: <https://ethereum.org/en/whitepaper/>. [Accessed: Jan. 6, 2025].
- [3] Statista, "Bitcoin energy consumption worldwide 2017-2025", 2024. [Online]. Available: <https://www.statista.com/statistics/881472/worldwide-bitcoin-energy-consumption/>. [Accessed: Jan. 6, 2025].
- [4] Ministry of Finance, Government of Pakistan, "Energy," in *Pakistan Economic Survey 2023-24*, 2024, ch. 14, pp. 1-10. [Online]. Available: https://www.finance.gov.pk/survey/chapter_24/14_energy.pdf. [Accessed: Jan. 6, 2025].
- [5] K. Lau and Crypto.com, "Ethereum 2.0," report, 2020. Available: https://assets.ctfassets.net/hfgyig42jimx/7j3AVp5aCnx2Ct2P6T6kY/e8991d5da1972d5d760233868f237609/Crypto.com_Macro_Report_-_Ethereum_2.0.pdf [Accessed: Jan. 8, 2025].
- [6] E. Buchman, J. Kwon, and Z. Milosevic, "The Latest Gossip on BFT Consensus," arXiv, 2018. [Online]. Available: <https://doi.org/10.48550/arxiv.1807.04938>. [Accessed: Jan. 8, 2025].
- [7] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," Jan. 2016. [Online]. Available: <https://lightning.network/lightning-network-paper.pdf>. [Accessed: Jan. 8, 2025].
- [8] Ethereum Foundation, "Scaling," Ethereum.org, May 7, 2024. [Online]. Available: <https://ethereum.org/en/developers/docs/scaling/>. [Accessed: Jan. 8, 2025].
- [9] Hyperledger Fabric Documentation, "Introduction," [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/whatis.html>. [Accessed: Jan. 10, 2025].
- [10] Dragonchain, "Enterprise Interoperability with Dragonchain," Dragonchain Blog, [Online]. Available: <https://dragonchain.com/blog/enterprise-interoperability-with-dragonchain>. [Accessed: Jan. 10, 2025].
- [11] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," in *Proceedings of the NetDB*, Athens, Greece, 2011. [Online]. Available: <https://notes.stephenholiday.com/Kafka.pdf>. [Accessed: Jan. 10, 2025].

- [12] Hyperledger Fabric Documentation, "Bringing up a Kafka-based Ordering Service," [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.5/kafka.html>. [Accessed: Jan. 10, 2025].
- [13] LF Decentralized Trust, "Hyperledger Indy," [Online]. Available: <https://www.lfdecentralizedtrust.org/projects/hyperledger-indy>. [Accessed: Jan. 10, 2025].
- [14] J. R. Douceur, "The Sybil Attack," in *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, 2002, pp. 251-260. [Online]. Available: <https://dl.acm.org/doi/10.5555/646334.687813>. [Accessed: Jan. 11, 2025].
- [15] M. Saad et al., "Exploring the Attack Surface of Blockchain: A Systematic Overview," *arXiv preprint arXiv:1904.03487*, 2019. [Online]. Available: <https://arxiv.org/abs/1904.03487>. [Accessed: Jan. 11, 2025].
- [16] B. Haeupler, G. Pandurangan, D. Peleg, R. Rajaraman, and Z. Sun, "Discovery through Gossip," arXiv preprint arXiv:1202.2092, 2012. [Online]. Available: <https://arxiv.org/abs/1202.2092>. [Accessed: Jan. 11, 2025].
- [17] M. Vranken, "Sustainability of bitcoin and blockchains," *Current Opinion in Environmental Sustainability*, vol. 28, pp. 1-9, 2017. [Online]. Available: https://www.researchgate.net/publication/320123454_Sustainability_of_bitcoin_and_blockchains. [Accessed: Jan. 15, 2025].
- [18] M. Vranken, "Sustainability of bitcoin and blockchains," *Current Opinion in Environmental Sustainability*, vol. 28, pp. 1-9, 2017. [Online]. Available: https://www.researchgate.net/publication/320123454_Sustainability_of_bitcoin_and_blockchains. [Accessed: Jan. 15, 2025].
- [19] M. Raczyński, "What is the Fastest Blockchain and Why? Analysis of 43 Blockchains," Aleph Zero, 2023. [Online]. Available: <https://alephzero.org/blog/what-is-the-fastest-blockchain-and-why-analysis-of-43-blockchains/>. [Accessed: Jan. 15, 2025].
- [20] SKALE Network, "Best Blockchain Performance: Results from Dartmouth Blockchain Study | SKALE," *SKALE*, Aug. 23, 2024. Available: <https://skale.space/blog/the-quest-for-the-best-blockchain-performance-results-from-dartmouth-blockchain-study>. [Accessed: Jan. 15, 2025].