FINAL YEAR PROJECT REPORT
BACHELOR OF ENGINEERING

Blockchain-enabled Domain Name System for Private Network

Student:	Zhou Zihan
GUID:	2510791
1st Supervisor:	Lei Zhang
2nd Supervisor:	Li Yu

2022-2023

Coursework Declaration and Feedback Form

Student Name: Zhou Zihan	Student GUID: 2510791
Course Code : UESTC4006P	Course Name : INDIVIDUAL PROJECT 4
Name of 1st Supervisor: Lei Zhang	Name of 2nd Supervisor: Li Yu
Title of Project: **Blockchain-enabled Domain Name System for Private Network**	
Declaration of Originality and Submission Information	
I affirm that this submission is all my own work in accordance with the University of Glasgow Regulations and the School of Engineering requirements *Signed (Student) : Zhou Zihan*	UESTC4006P TEC-IT.COM
Date of Submission : 2023/4/26	
Feedback from Lecturer to Student – to be completed by Lecturer or Demonstrator	
Grade Awarded: Feedback (as appropriate to the coursework which was assessed):	
Lecturer Demonstrator:	Date returned to the Teaching Office:

Abstract

Traditional domain name system faces challenges such as centralization, vulnerability to attacks, and single points of failure, which become increasingly critical as network size and heterogeneity grow. Meanwhile, blockchain technology has emerged as a promising approach to address these limitations through decentralization, security, and fault tolerance. In this report, the author proposed *Blockchain-enabled Domain Name System*, a novel domain name system, effectively integrating the advantages of blockchain technology and the practicality of the consensus algorithm. The authors put forward an innovative consensus algorithm, SyncRaft, which is derived from the well-established Raft consensus algorithm. This innovation enhances the degree of decentralization within the network. The evaluation of Be-DNS focuses on latency, throughput, and fault tolerance in comparison to well-established consensus algorithms. The results indicate competitive performance, achieving relatively low latency, reasonable throughput, and fault tolerance. Be-DNS demonstrates medium decentralization levels, improved security, and reduced risk of single points of failure, alongside lower energy consumption and medium scalability. By leveraging the strengths of blockchain technology and the SyncRaft consensus algorithm, Be-DNS offers enhanced security and scalability while maintaining competitive performance. Consequently, Be-DNS has the potential to become a reliable, secure, and efficient domain name system, contributing to the evolution of the internet infrastructure. The complete source code and implementation details for Be-DNS can be found in the author's GitHub repository at: https://github.com/xiaoZ857/FYP-Be-DNS.

Acknowledgements

I would like to express my deepest gratitude to my supervisors for their invaluable guidance, expertise, continuous support throughout the course of this project, and insightful advice and encouragement during the research process.

Special thanks go to Xiaozhu Luo, my girlfriend, for her unwavering love, patience, and understanding, which provided the motivation and inspiration necessary for the completion of this work. I would also like to express my heartfelt gratitude to my parents for their unconditional love and support throughout my academic journey.

I am grateful for the camaraderie and assistance provided by my roommates, who have contributed to a positive and productive environment, fostering personal and academic growth. Lastly, I would like to thank Yixuan Fan and Ruiyu Wang for their guidance and support, as well as Chenxiao Guo and Zihao Feng for their help and inspiration throughout this project.

Contents

1 Introduction

As the digital landscape continues to evolve, the emergence of groundbreaking technologies such as the Internet of Things (IoT) [1], 5G networks [2], and edge computing [3] has further underscored the importance of a robust and reliable Domain Name System (DNS). These cutting-edge technologies rely on seamless and low-latency connectivity to support real-time data processing, communication, and control. The IoT, for instance, necessitates efficient domain name resolution to accommodate the vast number of connected devices [1], while 5G networks demand rapid and secure DNS services to enable ultra-reliable, high-speed communication [2]. In the context of edge computing, DNS plays a vital role in facilitating the optimal routing of data between devices and edge nodes [3]. As these state-of-the-art technologies become increasingly prevalent, addressing the limitations of conventional DNS architectures and exploring innovative approaches to DNS, such as distributed systems and blockchain technology, is essential for maintaining a secure, reliable, and interconnected digital ecosystem that can fully harness the potential of these technological advancements.

1.1 Background

1.1.1 DNS

DNS serves as a pivotal element of internet infrastructure, facilitating the translation of human-readable domain names into Internet Protocol (IP) addresses, which computers utilize for locating and accessing online resources [4]. DNS is instrumental in ensuring seamless connectivity, and thus, maintaining the stability and performance of the World Wide Web is of paramount importance. Nevertheless, conventional DNS systems grapple with a multitude of challenges, encompassing centralization, susceptibility to assorted attacks, and single points of failure.

Centralization in traditional DNS architectures engenders considerable risk, rendering the system vulnerable to targeted attacks and potentially causing critical service unavailability. In response to these concerns, researchers have investigated distributed systems capable of providing augmented fault tolerance, security, and scalability [5]. Distributed systems rely on a network of interconnected nodes to sustain system functionality and distribute the workload, mitigating the consequences of single points of failure and bolstering overall system resilience.

1.1.2 Blockchain technology

Blockchain technology, a form of distributed system, has garnered substantial attention in recent years due to its capacity to enhance security, transparency, and decentralization [4]. A blockchain constitutes a decentralized and distributed digital ledger that employs cryptographic techniques to safeguard data integrity and maintain consistency among participating nodes [5]. This technology hinges on consensus algorithms to achieve agreement among nodes and validate transactions or updates within the distributed ledger [6].

Consensus algorithms are integral to distributed systems, including blockchains, as they ensure all participating nodes concur on the system's state. Numerous consensus algorithms have been proposed and scrutinized, such as Paxos [7], Proof of Work (PoW) [5], Practical Byzantine Fault Tolerance (PBFT) [6], Proof of Stake (PoS) [8] and Raft [9]. Each consensus algorithm exhibits unique strengths and weaknesses, with varying degrees of fault tolerance, security, performance, and energy efficiency. In this context, the exploration and assessment of various consensus algorithms are crucial to understanding their potential advantages and drawbacks, as well as to identifying the most suitable algorithm for specific use cases or applications.

1.2 Motivation and Challenges

The growing reliance on the internet for diverse applications, such as business operations, communication, critical infrastructure services, and the Internet of Things, has underscored the need for a robust and secure domain name system [10]. Traditional DNS systems, despite their proven efficacy, face challenges in reliability, security, and scalability, rendering them ill-equipped to meet the escalating demands of modern interconnected systems [11]. This situation has spurred research into innovative approaches harnessing distributed systems, blockchain technology, and consensus algorithms [12].

One key motivation behind exploring these technologies is their potential to address the risks posed by centralization in traditional DNS systems. By capitalizing on the inherent decentralization of distributed systems and blockchain technology, a more resilient domain name system can be developed, capable of withstanding targeted attacks and bypassing single points of failure [10]. Moreover, employing consensus algorithms in a decentralized DNS system ensures data consistency, integrity, and availability across participating nodes, addressing the limitations of conventional DNS architectures [12].

Another critical motivating factor is the urgent need to enhance the environmental sustainability of internet infrastructure. Traditional consensus algorithms, for example, Proof of Work, have attracted criticism for their high energy consumption, exacerbating the environmental impact of blockchain technologies [11]. Researchers are exploring alternative consensus algorithms with improved energy efficiency and performance to strike a balance between security and sustainability in novel domain name system development [12].

Furthermore, the rapid expansion of interconnected systems, along with increasing network heterogeneity and complexity, necessitates scalable domain name systems that can accommodate a growing number of nodes and operations without significant performance degradation [11]. The integration of distributed systems, blockchain technology, and consensus algorithms shows promise in delivering a scalable and efficient solution for future domain name systems [12].

Given these motivations, this report aims to contribute to the ongoing discourse on next-generation domain name systems by presenting an innovative approach that amalgamates the advantages of distributed systems, blockchain technology, and consensus algorithms. The goal is to lay a foundation for further research and practical implementations, ultimately leading to a more reliable, secure, and scalable domain name system.

1.3 Contributions

This thesis seeks to advance the understanding and application of distributed systems, blockchain technology, and consensus algorithms in the context of domain name systems. The main contributions of this work are as follows:

- **Proposing Be-DNS:** The proposal of Be-DNS introduces a novel domain name system that addresses the limitations of traditional DNS systems by enhancing security, reliability, and performance. To achieve this, the author designed and built a blockchain system from scratch, effectively integrating blockchain technology within the Be-DNS framework.
- **Proposing SyncRaft:** The proposal of SyncRaft, a polished consensus algorithm based on Raft, offers a more decentralized approach than the traditional Raft algorithm while retaining its practicality. By integrating the benefits of blockchain technology, SyncRaft enhances decentralization and promotes a secure, tamper-resistant environment.
- **Performance testing and analysis:** The comprehensive evaluation of Be-DNS in terms of latency, throughput, and fault tolerance. The analysis demonstrates the competitive performance of Be-DNS, highlighting its potential as an effective alternative to traditional DNS systems.

The structure of this thesis is organized as follows: A literature review on blockchain technology and DNS systems is presented, followed by the system design of Be-DNS and SyncRaft algorithm. The implementation is then discussed, and the performance of Be-DNS is evaluated. Finally, the paper concludes with key findings, implications, limitations, and future research directions.

2 Literature Review

2.1 Blockchain technology

The advent of blockchain technology has brought forth a revolutionary paradigm shift in the way data is stored, managed, and exchanged across a plethora of industries. By leveraging the principles of decentralization, cryptography, and consensus algorithms, blockchain offers a robust, transparent, and immutable platform for maintaining distributed ledgers. Blockchain technology has emerged as a groundbreaking innovation with the potential to transform various industries, including finance, supply chain, healthcare, and information technology [13].

2.1.1 Foundational concepts

Blockchain technology embodies a decentralized and distributed ledger system, fundamentally consisting of a chronological series of data blocks, each encapsulating a set of transactions [11]. Cryptographic techniques safeguard each block, and a distinctive hash value connects it to the preceding block in the chain. This intrinsically secure chain of blocks ensures the maintenance of data integrity and immutability, as any attempt to alter a block would require the recomputation of the cryptographic hashes for all subsequent blocks, making such an endeavor computationally infeasible [14].

- **Hash Algorithm**: A hash algorithm is a mathematical function that takes in a variable-length input, such as a message or data, and produces a fixed-size output, called a hash or a digest [5]. The output has a unique value that represents the input data, which makes it useful for verifying the integrity of data and ensuring that it hasn't been altered or corrupted. The hash function is designed to be one-way, meaning it is easy to compute the hash of the input data, but it is practically impossible to retrieve the original input data from the hash. Hash algorithms are used in a variety of applications, such as password storage, digital signatures, and blockchain technology, where they are used to secure and verify the authenticity of data. Some popular hash algorithms include SHA-256, MD5, and SHA-3 [15].
- **Blockchain Structure**: The blockchain data structure is composed of a chain of blocks, where each block contains a cryptographic hash of the previous block, a timestamp, and a batch of transactions. The hash of each block is computed based on the hash of the previous block, which creates a chain of blocks that are cryptographically linked, hence the name "blockchain." Each block also contains a nonce, which is a random number used in the proof-of-work consensus algorithm to validate the block and add it to the blockchain [16]. The block structure includes a header section, which contains the block's metadata, and a body section, which contains the transaction data. The header section includes the block's hash, nonce, previous block hash, and timestamp. The body section contains the transactions that occurred since the previous block was added to the chain. A brief blockchain

structure is shown in Figure 1. The combination of these features creates a secure, decentralized, and immutable ledger that is ideal for storing transactional data without the need for intermediaries [5].

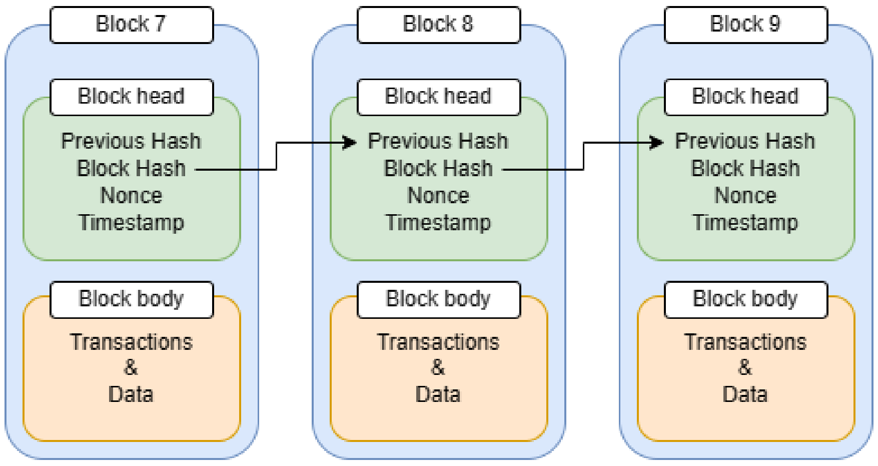


Figure 1: A brief blockchain structure

- **Blockchain Account**: In blockchain technology, an account is a unique identifier that represents a user or entity that can send or receive transactions on the network. Each account is associated with a public key, which is used to receive transactions, and a private key, which is used to sign and authorize transactions. The public key is generated from the private key using a mathematical algorithm, such as elliptic curve cryptography. The private key is typically generated randomly and should be kept secure and secret [8]. The private key is kept secret by the account owner and is used to prove ownership of the account and authorize transactions on the network. The account balance is the total amount of cryptocurrency or digital assets associated with the account. In some blockchain networks, accounts may also have additional attributes or metadata associated with them, such as smart contracts, tokens, or other custom data. The account structure is a key component of blockchain technology, as it enables secure, decentralized, and transparent transactions [5, 16].

2.1.2 Consensus algorithms

The attainment of consistency and accord among the various participants in a blockchain network, known as nodes, relies on the employment of consensus algorithms. Several well-established consensus algorithms include Paxos [7], Proof of Work (PoW) [5], Practical Byzantine Fault Tolerance (PBFT) [6], Proof of Stake (PoS) [8] and Raft [9]. These algorithms ensure that a consensus is reached within the network concerning the contents of the blockchain, notwithstanding the presence of malicious or faulty nodes. Consensus algorithms form the backbone of blockchain technology, ensuring the integrity, security, and reliability of data in a decentralized and trustless environment. This section provides an overview of the most common consensus algorithms used in blockchain networks and their key characteristics [16].

- **Proof of Work (PoW)**: PoW is the consensus algorithm employed by the first and most prominent blockchain, Bitcoin. It necessitates miners to solve complex mathematical problems, requiring significant computational resources. The first miner to solve the problem is granted the right to add a new block to the blockchain and receives a reward for their efforts. While PoW offers a high level of security and decentralization, it has been criticized for its energy consumption and potential centralization due to the computational power required for mining [5].

- **Proof of Stake (PoS)**: PoS is an alternative consensus mechanism that addresses some of the limitations of PoW. Instead of relying on computational power, PoS selects validators based on the number of tokens they hold (their stake) and other factors, such as the duration of holding those tokens. Validators are then chosen to create new blocks and confirm transactions, with the probability of being selected proportional to their stake. PoS is more energy-efficient than PoW, and it encourages long-term investment in the network. However, it may face issues related to initial distribution and concentration of wealth [17].
- **Practical Byzantine Fault Tolerance (PBFT)**: PBFT is a consensus algorithm designed to tolerate Byzantine faults, which are failures in a distributed system where nodes may exhibit arbitrary or malicious behavior. In PBFT, nodes in the network communicate and exchange information to reach a consensus on the state of the system. The algorithm can achieve consensus even if a certain fraction of nodes are faulty or malicious, provided that the majority of nodes are honest. PBFT is highly energy-efficient and can achieve fast transaction confirmations, but it may be less suited for large-scale networks due to its reliance on extensive communication between nodes [6].
- **Paxos**: Paxos is a consensus algorithm designed to achieve consensus among unreliable processors in distributed systems. It offers fault tolerance and data consistency even in the presence of node failures or communication delays. Paxos works through a series of communication rounds between nodes, consisting of two phases: Prepare and Accept. Proposers send prepare requests to acceptors, which, upon receiving a majority of responses, proceed to the Accept phase, where acceptors vote on the proposed value. Although Paxos provides strong safety and fault tolerance, its complexity and communication overhead make it challenging to understand and implement. It is more applicable to private or consortium blockchain networks with a limited number of nodes, focusing on data consistency and fault tolerance [18].
- **Raft**: Raft is a consensus algorithm designed to be more understandable and easier to implement than other distributed consensus mechanisms such as Paxos. Raft is commonly used in distributed systems to manage replicated logs and ensure consistency across the system. It achieves consensus by electing a leader among the nodes, who then manages the log replication process and coordinates with other nodes to maintain consistency. In the event of leader failure, the algorithm triggers a new election to select a replacement leader. While Raft may not be inherently designed for blockchain applications, its core concepts can be adapted and applied to blockchain systems to facilitate con-

sistency, fault tolerance, and decentralized decision-making. The simplicity and understandability of the Raft algorithm make it an appealing choice for developers and researchers seeking to develop novel consensus mechanisms tailored to specific use cases or to address the limitations of existing algorithms in blockchain technology [9].

In Table 1, a comparison of some of the most popular consensus algorithms used in blockchain is provided, including PoW, PoS, PBFT, Paxos, and Raft. The table compares these algorithms based on key factors such as decentralization, energy consumption, scalability, throughput, security, and fault tolerance. This comparison can be useful in selecting the optimal consensus algorithm for a particular blockchain application, taking into account factors such as network size, security requirements, and available resources [6,9,13,16].

Table 1: Comparison of Blockchain Consensus Algorithms

	PoW	**PoS**	**PBFT**	**Paxos**	**Raft**
Decentralization	High	High	Medium	Low	Low
Energy Consumption	High	Medium	Medium	Low	Low
Scalability	High	Medium	Medium	High	Low
Throughput	Low	Medium	High	High	High
Security	High	High	High	Medium	Medium
Fault Tolerance	High	High	Low	Medium	Medium

2.1.3 Categorization of blockchain networks

Blockchain networks can be categorized into three primary types based on their access permissions and participant restrictions: public, private, and consortium blockchains [16].

- **Public Networks**: Public Networks, exemplified by Bitcoin and Ethereum, are accessible to any interested party and operate within a permissionless environment [19]. They provide an elevated level of decentralization, security, and transparency. However, they may encounter challenges pertaining to scalability and energy consumption due to their open and unrestricted nature.
- **Private Networks**: Private Networks restrict participation to a select group of trusted entities and function within a permissioned environment [20]. While conceding a certain degree of decentralization, private blockchains can deliver enhanced efficiency, scalability, and privacy, rendering them ideal for applications necessitating access control and regulatory compliance. Be-DNS is designed for private networks.
- **Consortium Networks**: Consortium Networks, also known as federated blockchains, represent a hybrid approach between public and private blockchains [21]. They are governed by a predefined set of trusted entities or organizations, which share control over the network. Consortium blockchains

balance decentralization, security, and scalability by restricting access to a limited group while still distributing authority among multiple entities. This approach is suitable for applications where collaboration between different organizations is required while maintaining a controlled environment.

2.1.4 Advantages and Applications

Blockchain technology offers a myriad of advantages, which translate into a diverse range of potential applications spanning various sectors. The key benefits conferred by blockchain technology, along with their corresponding applications and use cases, are discussed below:

- **Decentralization**: The elimination of a central authority engenders a more distributed and resilient system devoid of single points of failure. This characteristic is particularly valuable in creating cryptocurrencies and digital payment systems, such as Bitcoin [5] and Ethereum [8], which facilitate secure and decentralized financial transactions.
- **Immutability**: The cryptographic safeguards and consensus mechanisms inherent in blockchain technology ensure that data, once recorded on the blockchain, remains impervious to tampering or alteration. This feature is crucial in supply chain management, where blockchain can deliver end-to-end visibility, traceability, and transparency, enhancing efficiency, curbing fraud, and guaranteeing product authenticity. For example, a copyright-aware Blockchain-enabled Knowledge Sharing platform is proposed to allow students to share their works without the worry of being plagiarized [22].
- **Transparency**: Transactions and data stored on a blockchain are visible to all network participants, fostering trust and accountability. This aspect is particularly advantageous in public record management, such as land registries and government record-keeping, where increased transparency can help prevent corruption and improve public trust. For instance, Shubham Chakraborty et al. proposed a carbon credit ecosystem using blockchain, guarantee the security, transparency, accessibility to the standardized carbon markets [23].
- **Security**: The decentralized architecture and cryptographic protections intrinsic to blockchain technology render it highly resistant to cyberattacks, fraud, and data breaches. This level of security is essential for digital identity and access management solutions, streamlining authentication and access control processes while ensuring user privacy and data protection.
- **Programmability**: Blockchain-based smart contracts enable the formulation of self-executing, programmable agreements capable of automating a myriad of processes and transactions. This capability has numerous applications, including decentralized finance (DeFi) platforms, insurance claim processing, and intellectual property management [24].

In conclusion, the unique combination of advantages offered by blockchain technology has facilitated its integration across a wide array of industries and use cases [25]. By leveraging the benefits of decentralization, immutability, transparency, security, and programmability, blockchain technology has the potential to address the limitations of traditional systems and foster a more interconnected, efficient, and trustworthy digital ecosystem.

2.1.5 Challenges and Future directions

Despite its numerous advantages and potential applications, blockchain technology also faces several challenges that need to be addressed to fully realize its potential [16]. Some of these challenges include:

- **Scalability**: Public blockchains, in particular, often struggle with scaling issues as the number of transactions and participating nodes increases, leading to network congestion and longer transaction times [26].
- **Energy Consumption**: Consensus algorithms such as Proof of Work (PoW) are known for their high energy consumption, which has raised environmental concerns and spurred the exploration of more energy-efficient consensus mechanisms [27].
- **Interoperability**: The growing number of distinct blockchain platforms has led to a lack of standardization and interoperability, making it difficult for different blockchain networks to communicate and exchange data seamlessly [28].
- **Regulatory and Legal Frameworks**: As a nascent technology, blockchain faces uncertainty regarding regulatory and legal frameworks, which could impact its adoption and integration across various sectors [29].
- **Privacy and Data Protection**: Although blockchain technology offers a certain degree of privacy through cryptographic techniques, ensuring compliance with data protection regulations and preserving user privacy in a transparent, decentralized environment remains a challenge [30].

To address these challenges, researchers and industry practitioners are continuously working on new solutions and approaches, such as layer-2 solutions [31], alternative consensus algorithms [17], and cross-chain communication protocols [32]. Additionally, the development of robust regulatory frameworks and the establishment of best practices for privacy and data protection will be crucial in facilitating the widespread adoption and integration of blockchain technology across various industries [25].

In light of these advancements and ongoing research, it is anticipated that blockchain technology will continue to evolve and mature, paving the way for innovative applications and solutions that can transform the digital landscape and create new opportunities for value creation and societal impact.

2.2 Domain name system

The Domain Name System (DNS) constitutes a vital element of the internet's infrastructure, serving the critical function of converting user-friendly domain names into computer-interpretable IP addresses [33].

This section presents an overview of DNS, delineating its core functions, inherent advantages, and potential challenges.

2.2.1 Overview

DNS is a hierarchical, distributed database that maps domain names to IP addresses, enabling users to access websites and other internet resources using easily memorable domain names rather than numerical IP addresses. DNS servers, distributed across the globe, store and maintain this mapping information, providing a reliable and efficient means to resolve domain names into IP addresses [33]. The domain name system is organized into a hierarchical structure, with each domain name consisting of a series of labels separated by dots. As Figure 2 shown, the top-level domain (TLD) is the highest level in the hierarchy and represents the generic or geographic category of the domain. Examples of TLDs include ".com", ".org", and country code TLDs like ".uk" or ".cn". The next level down is the second-level domain (SLD), which represents the specific organization or entity that owns the domain. For example, in the domain name "www.gla.ac.uk", ".ac" is the SLD and ".uk" is the TLD, the authoritative servers for "gla.ac.uk" would be responsible for providing information about the "gla.ac.uk" domain to other DNS servers, and the host server for "www.gla.ac.uk" would be responsible for hosting the actual web page associated with that domain. The hierarchy continues with subdomains, which can be used to further divide and organize the domain structure. The domain name hierarchy provides a way to organize and manage the vast number of domain names and resources on the internet, making it easier for users to find and access the content they need [34].

When a client attempts to access the website "www.gla.ac.uk", it sends a request to the DNS server for the domain name resolution. The DNS server first checks its own records to see if it has the IP address associated with the requested domain name. If it doesn't have the record, it forwards the request to other DNS servers in a hierarchical manner until a server with the requested record is found. Once the record is found, the IP address is returned to the client, which then uses it to access the corresponding web page or other internet resource. The brief process of accessing "www.gla.ac.uk" is shown as Figure 3.

2.2.2 Core functions

- **Registration**: Domain name registration is the process of acquiring a unique domain name for a website or online service. This involves registering the name with a domain name registrar, who is responsible for managing and maintaining the registration records for the domain name. The registration process typically involves selecting a domain name, checking its availability, and submitting the necessary information and payment to the registrar. Once registered, the domain name becomes the property of the registrant, who has the exclusive right to use it for a specified period, usually between one and ten years. Domain name registration is a crucial aspect of establishing an online presence,

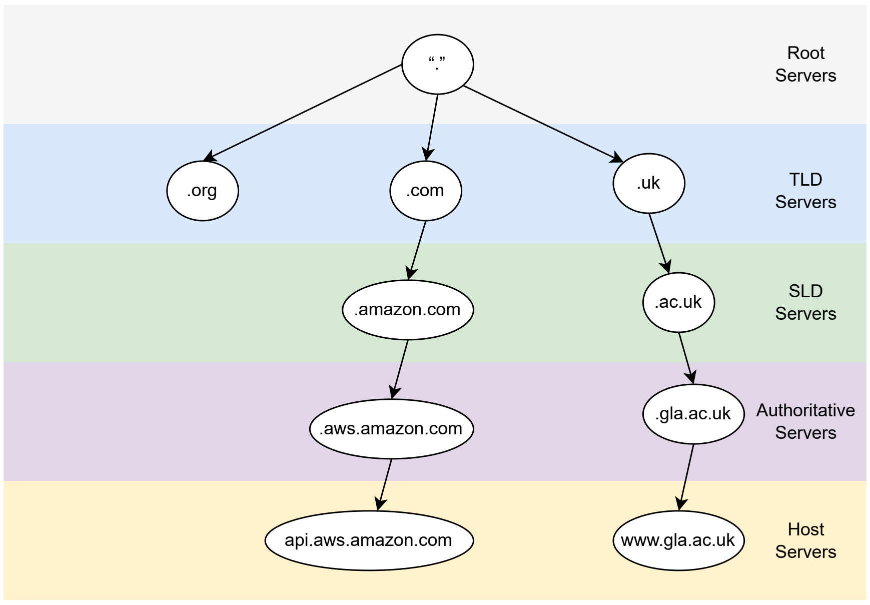


Figure 2: DNS hierarchical structure

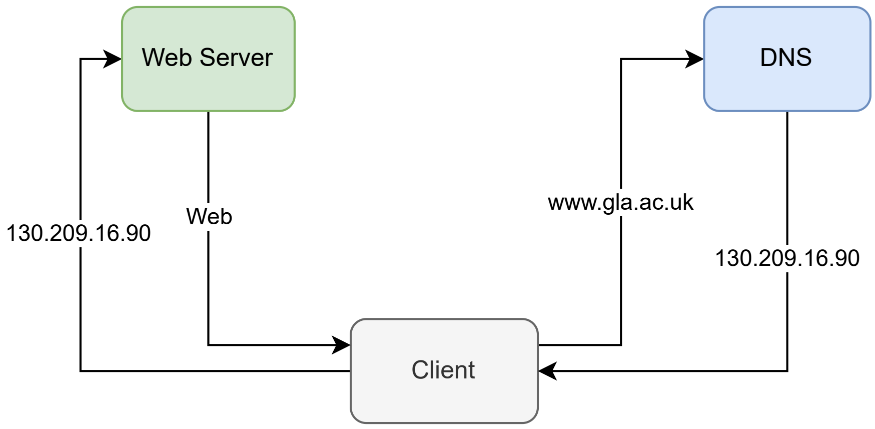


Figure 3: Brief process of accessing website with domain name

and the availability of a relevant and memorable domain name can significantly impact a website's success [33].

- **Resolution**: Domain name resolution is the process of converting a human-readable domain name, such as "www.gla.ac.uk", into an IP address that can be understood by machines. This process is essential for the functioning of the internet, as it allows users to access websites and other online resources using domain names, which are easier to remember than numerical IP addresses. The domain name resolution process involves a series of queries and responses between client devices, DNS servers, and authoritative name servers, which store and maintain information about the mapping of domain names to IP addresses. The efficient and reliable functioning of domain name resolution is critical for the proper functioning of the internet and the seamless access to online resources [34]. When a client attempts to access "www.gla.ac.uk" as shown in Figure 4, it first sends a request (Q1) to its local DNS resolver, which is typically provided by the client's internet service provider (ISP).

The local resolver then checks its cache to see if it has a record of the domain name and its associated IP address. If the record is not found, the local resolver forwards the request (Q2) to a root DNS server, which responds with a referral(A1) to the appropriate TLD server for the .uk domain. The TLD server which receives the query (Q3) then refers the local resolver to the SLD server responsible for the .ac.uk domain (A2). Then the SLD server responds the request (Q4) with the address of the authoritative server to local DNS resolver (A3). The authoritative server responds with the IP address of the web server hosting the www.gla.ac.uk website to the local resolver (A4), after being requested (Q5). Finally, the local resolver returns the IP address to the client (A5), which uses it to establish a connection with the web server and access the website.

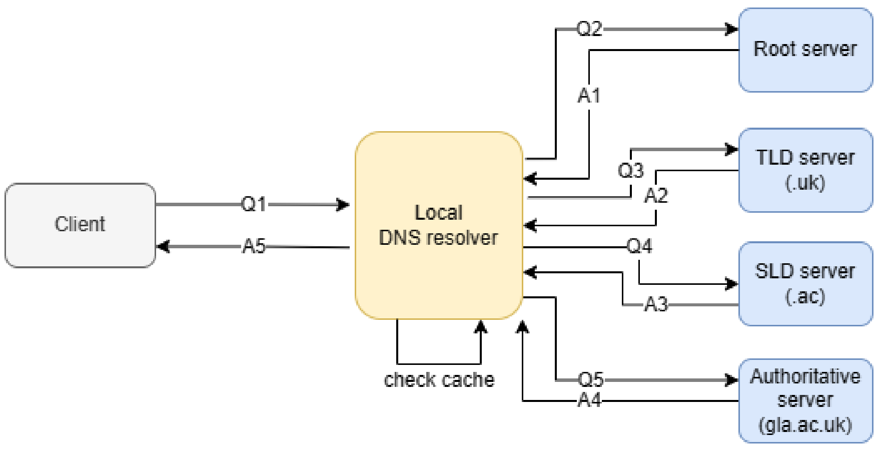


Figure 4: Domain name resolution process

2.2.3 Inherent advantages

DNS offer several inherent advantages that make them an essential component of the internet infrastructure. One of the most significant advantages is their ability to provide efficient and scalable domain name resolution services, which enable users to access internet resources using human-readable domain names. This translates into an improved user experience, as it is much easier to remember and use domain names than numerical IP addresses [35]. Additionally, DNS servers support the distribution of workload and network traffic, as they can handle a high volume of queries from clients and resolve domain names into IP addresses quickly and reliably. Another advantage of DNS is its hierarchical structure, which allows for delegation of authority and decentralization of domain name management, resulting in a more robust and resilient system with no single point of failure. Overall, the inherent advantages of DNS make it an essential technology for efficient and reliable internet communication, facilitating the growth and expansion of the internet and its associated services [34].

2.2.4 Potential challenges

DNS face various challenges that can affect their performance, security, and reliability. One of the main challenges is the potential for DNS cache poisoning attacks, which can lead to incorrect or malicious mappings of domain names to IP addresses. Another challenge is the possibility of DNS amplification attacks, where attackers exploit vulnerable DNS servers to flood a target system with an overwhelming amount of traffic [36]. DNS servers can also be targeted by distributed denial of service (DDoS) attacks, which can disrupt their availability and cause significant downtime for websites and internet services. In addition to these security threats, DNS servers also face challenges related to scalability, performance, and maintenance. As the volume of internet traffic continues to grow, DNS servers must be able to handle increasing levels of requests while maintaining high levels of performance and reliability. DNS administrators must also keep their servers up-to-date with the latest security patches and software updates to prevent vulnerabilities and ensure the ongoing security and integrity of the DNS [37].

3 System Design

3.1 Framework

The Be-DNS framework, as depicted in Figure 5, presents a coherent architecture designed to provide a decentralized and secure domain name resolution system. The framework begins with the blockchain network layer, where a collection of nodes participates in maintaining the distributed ledger. The blockchain design is composed of block design, account design, and data storage design to ensure the integrity and security of domain name bindings and associated information. The SyncRaft consensus algorithm is introduced to facilitate a decentralized and fault-tolerant mechanism for achieving network agreement on the blockchain's state. The algorithm's design consists of node design and algorithm design, effectively connecting the blockchain network layer to the functions layer. The functions layer encompasses four essential services that streamline user access to online resources via human-readable domain names: account creation, domain name registration, domain name resolution, and domain name modification. To deliver a user-friendly interaction platform, the web interface layer is built on HTML, CSS, and HTTP frameworks. This layer provides a seamless connection between personal devices and the Be-DNS network, simplifying user operations. Additionally, an HTTP API is available for IoT devices to interact with the system.

By integrating these layers and components, Figure 5 effectively highlights the cohesive architecture of the Be-DNS system, emphasizing its capacity for decentralized and secure domain name resolution.

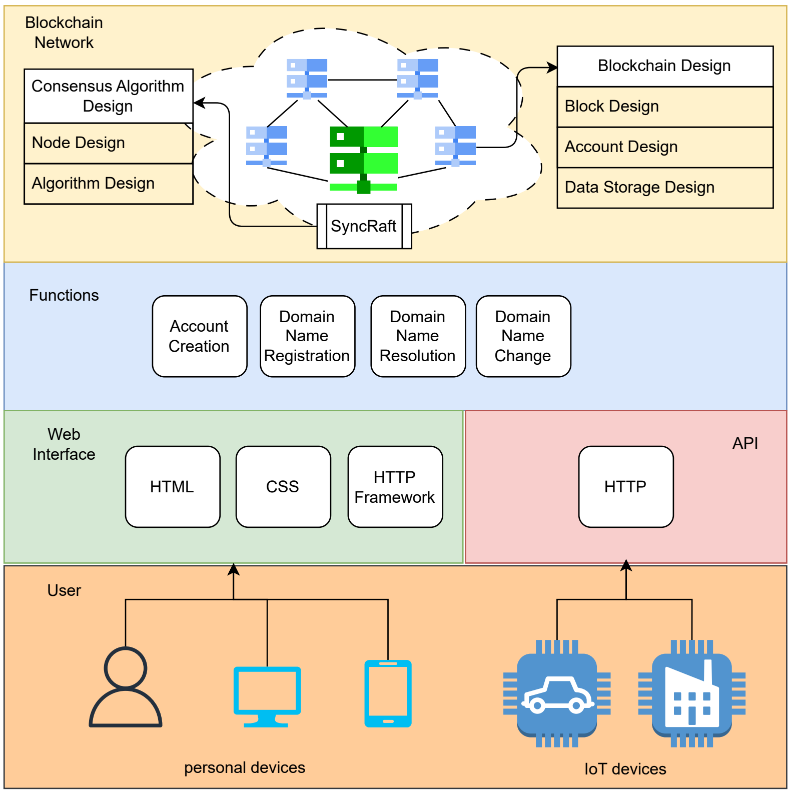


Figure 5: Blockchain-enabled DNS Framework

3.2 Blockchain design

The blockchain design is a critical component of the Be-DNS system, providing the infrastructure for data storage and management. The design of the blockchain includes block structure design , account design and data storage design.

3.2.1 Block design

The block design in Be-DNS serves as the fundamental building block for storing and maintaining data in a decentralized and tamper-resistant manner. As Figure 6 shown, Each block is composed of five primary components:

- **Index**: The index provides a unique identifier for each block in the blockchain.
- **Previous hash**: The previous hash value links each block to its preceding block.
- **Timestamp**: The timestamp records the date and time when the block was created.
- **Map hash**: The map hash value represents the current state of the mapping data.
- **Block hash**: The hash value of current block ensures the immutability of block.

The use of hash values ensures the immutability and integrity of the blockchain data. The previous hash value acts as a pointer to the preceding block, forming a chain of blocks, which prevents any unauthorized changes to the data. The timestamp and map hash values are used to verify the authenticity and consistency of the data stored in each block. Additionally, the block design enables the Be-DNS to maintain a secure, tamper-resistant, and decentralized system for storing and managing domain name resolution information.

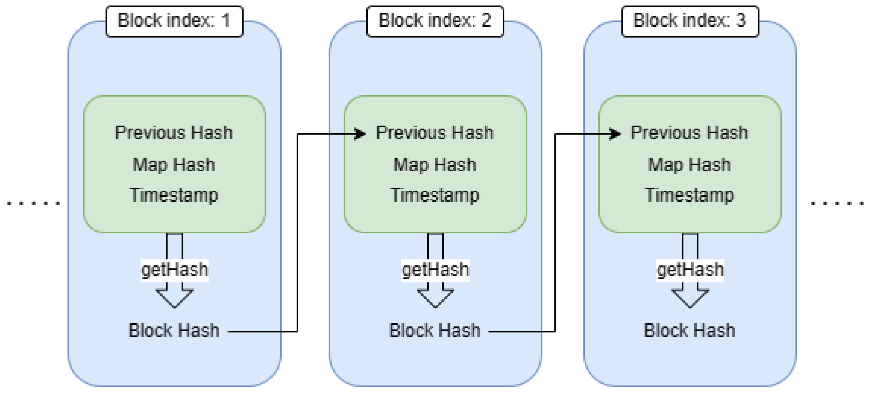


Figure 6: Block Structure of Be-DNS

3.2.2 Account design

In Be-DNS, the account architecture incorporates a two-step verification process involving account creation and domain name ownership validation. The initialization phase entails the user providing a distinct private key, coupled with a randomly generated number, or salt, to enhance the account's security. A public key is then derived through a cryptographic hash function, specifically SHA-256, by combining the private key and the salt.

The introduction of the salt during the account's inception guarantees the uniqueness and security of the generated public key. This is achieved by concatenating the salt with the private key, followed by hashing the resulting combination. Such an approach ensures that each account possesses a distinct and secure public key, serving as an authentication mechanism when incorporating or modifying DNS bindings within the blockchain.

3.2.3 Data storage design

Be-DNS maintains two primary mappings for managing domain names, IP addresses, and their corresponding owners. These mappings are critical to the operation of the system, as they enable nodes to resolve domain names into IP addresses and verify the ownership of domain names:

- **DomainNametoIpmap**: A mapping of domain names to their corresponding IP addresses. This mapping provides the fundamental functionality of the Be-DNS system, enabling nodes to resolve human-readable domain names into machine-readable IP addresses. When a new domain name and IP address binding is added or an existing domain name's IP address is changed, the NametoIpmap is updated to reflect the new information.
- **DomainNametoOwnermap**: A mapping of domain names to their respective owners, represented by their public keys. This mapping is essential for ensuring the security and authenticity of domain name management operations, as it allows nodes to verify that an individual attempting to add or change a domain name's IP address binding is the legitimate owner of the domain.

Be-DNS ensures the integrity of stored mappings through the use of cryptographic hashes and a consensus mechanism among participating nodes. When a new block is added to the blockchain, it must be verified by a majority of nodes before being accepted.

3.3 Blockchain consensus algorithm design

In a decentralized system like Be-DNS, it is crucial to have a consensus algorithm to maintain data consistency and integrity among nodes. Be-DNS employs a consensus algorithm, named SyncRaft, which is modified from Raft. It ensures the agreement of nodes on the state of mappings stored within the blockchain. SyncRaft is based on a synchronizer-follower model, where one node is elected as the synchronizer and is responsible for maintaining the agreement of the network. The other nodes act as followers and validate the proposed blocks before adding them to their local copies of the blockchain. The algorithm entails five key stages: initialization, synchronizer election, heartbeat detection, new block addition, and synchronizer failure handling. By utilizing heartbeat messages and synchronizer elections, SyncRaft ensures the proper functioning of the synchronizer, and thus, preserves consistency among network nodes. This algorithm allows individual nodes to add new blocks, while safeguarding network coherence. The brief process of SyncRaft is described in Algorithm 1, and detailed explanations will be given in the following subsections.

Algorithm 1: SyncRaft Algorithm

```
Input: Nodes in the network
Output: Consistent blockchain across the network
1  Initialization:
2  Nodes initialize state and timer set
3  Synchronizer Election:
4  if timer expires then
5      Broadcast candidacy
6      if majority votes received then
7          Become synchronizer
8          Send heartbeat messages
9  Heartbeat Detection:
10 Synchronizer sends heartbeats with block height and data hash
11 Nodes self-check and request synchronization if needed
12 Adding New Blocks:
13 if new block request then
14     Check block height consistency
15     if consistent then
16         Broadcast new block
17         Validate and add to blockchain
18 Synchronizer Failure Handling:
19 if synchronizer unresponsive then
20     Trigger new synchronizer election
21     if new synchronizer elected then
22         Resynchronize and resume operation
```

3.3.1 SyncRaft node design

The SyncRaft node design utilizes various parameters, which are divided into two categories: persistent and volatile. Persistent parameters maintain their values across node restarts, ensuring the consistency of crucial information such as the network structure and blockchain data. On the other hand, volatile parameters are designed to be reset upon node restarts or adapt to leadership changes and network events. The following lists provide detailed explanations of the functions and purposes of each parameter used in the SyncRaft node design.

Persistent Parameters of the SyncRaft Node:

- **nodes**: List of all nodes in the network. Persistent to maintain network structure and facilitate communication between nodes.
- **blockchain**: The instance of the blockchain class that represents the node's local blockchain. Persis-

tent to ensure that the blockchain data remains consistent across node restarts.

- **url**: The URL of the node for network communication. Persistent to uniquely identify the node and allow it to reconnect to the network.

Volatile Parameters of the SyncRaft Node:

- **term**: The term number representing the node's participation in synchronizer elections. Volatile as it needs to be reset upon node restarts to allow for new synchronizer elections.
- **leader**: The current synchronizer node of the network. Volatile to enable leadership changes and ensure proper functionality in case of leader node failures.
- **state**: The current state of the node, which can be 'candidate', 'synchronizer', or 'follower', adapting to leadership changes and other network events.
- **voted**: A boolean flag indicating if the node has voted in the current term, ensuring that the node can participate in new elections in case of a restart.
- **votesReceived**: The number of votes received by the node in the current term, allowing the node to participate in new elections after a restart.
- **received**: A boolean flag indicating if the node has received a heartbeat in the current term, which enables the node to detect synchronizer failures and start new elections.
- **majority**: The majority value, which is calculated as half of the total nodes minus one. Volatile to accommodate potential changes in the number of nodes.
- **timeout**: The timeout value for the node, chosen randomly in a range. Volatile to randomize node timeouts and avoid synchronization conflicts.
- **timer**: A timer object to trigger the node's runtime function based on the timeout value. Volatile to handle different timeouts and avoid conflicts between nodes.

3.3.2 Synchronizer election

The Synchronizer Election is a crucial phase in the SyncRaft algorithm, responsible for selecting the node that will act as the synchronizer to maintain consistency in the blockchain network. This process consists of the following steps, which are illustrated in Algorithm 2.

1. **Initialization:** Each node initializes its state with a random timeout and starts listening for heartbeat messages.
2. **Election Trigger:** Upon a node's timer expiration, it initiates the election process by broadcasting its candidacy to the other nodes in the network.
3. **Voting:** The other nodes receive the candidacy message and evaluate it based on certain criteria, such as the current block height. They then return their vote (yes or no) to the initiating node.

Algorithm 2: Synchronizer Election

```
Input: Nodes in the network
Output: Elected Synchronizer
1  foreach node do
2      Initialize timer with random timeout
3      if timer expires then
4          Broadcast candidacy to other nodes
5      end
6  end
7  foreach node do
8      if candidacy received then
9          Evaluate candidacy based on criteria
10         Return vote to initiating node
11     end
12 end
13 if initiating node receives majority of votes then
14     initiating node becomes synchronizer
15     Start sending heartbeat messages
16 end
```

4. **Synchronizer Selection:** If the initiating node obtains a majority of votes, it becomes the synchronizer and starts sending periodic heartbeat messages to maintain its status.

3.3.3 Heartbeat detection

The Heartbeat Detection process in the SyncRaft consensus algorithm is explained in Algorithm 3. The process can be described as follows:

1. **Periodic Heartbeat Messages**: The synchronizer node periodically sends heartbeat messages to all follower nodes in the network. These messages include the synchronizer's block height, map hash, term, and URL. The primary purpose of sending heartbeats is to demonstrate the synchronizer's liveliness and maintain its status in the network.
2. **Follower Self-Check and Synchronization**: Follower nodes continuously listen for incoming heartbeat messages from the synchronizer. Upon receiving a heartbeat message, follower nodes perform a self-check to determine whether their local block height and map hash match those of the synchronizer. If there is a discrepancy, the follower nodes request synchronization from the synchronizer to update their local copies of the blockchain and ensure consistency across the network.
3. **Synchronizer Failure Handling**: If a follower node does not receive a heartbeat message within a predetermined time interval, it assumes that the current synchronizer has failed or become unresponsive. In such cases, the follower node transitions to the candidate state and initiates a new synchronizer election. This mechanism guarantees that the network can recover from synchronizer failures and maintain the consistency of the blockchain.

Algorithm 3: Heartbeat Detection

```
Input: Synchronizer and Follower nodes
Output: Network consistency
1  while synchronizer is active do
2      Send heartbeat message to follower nodes
3      heartbeat message contains synchronizer's block height, map hash, and term
4      foreach follower node do
5          if heartbeat message received then
6              Perform self-check using received data
7              if inconsistencies found then
8                  Request synchronization from synchronizer
9              end
10         else
11             Set timer for heartbeat message timeout
12             if timer expires then
13                 Start new synchronizer election process
14             end
15         end
16     end
17     Wait for a predetermined interval
18 end
```

3.3.4 New block addition

Algorithm 4: New Block Addition

```
Input: Requesting Node $r$, Synchronizer Node $s$, New Block $B$
Output: Consensus on new block addition
1  $r \rightarrow s$: Send new block request;
2  $s$: Check block heights consistency among nodes;
3  if Block heights are consistent then
4      $s \rightarrow r$: Inform that the new block can be broadcasted; $r$: Broadcast new block to all nodes
5      foreach Node n in the network do
6          $n$: Validate the new block
7          if New block is valid then
8              $n$: Add new block to local blockchain
9          end
10     end
11 else
12     $s$: Reject the new block request;
13 end
```

The process of committing a new block to the blockchain network can be described as follows:

1. **New Block Request**: When a node in the network wants to add a new block, it sends a request to the synchronizer. This request contains the details of the new block, such as the updated mappings.
2. **Synchronizer Block Height Confirmation**: Before accepting the new block request, the synchronizer checks if the block heights of all nodes in the network are consistent. This step ensures that no

inconsistencies or forks exist in the blockchain before proceeding with the addition of the new block.

3. **Broadcasting the New Block**: Once the synchronizer confirms the consistency of block heights, it informs the requesting node that it can broadcast the new block. The node then broadcasts the new block to all other nodes in the network, who validate the block's contents.
4. **Block Validation and Addition**: Upon receiving the new block, each follower node validates it by checking the integrity of the data and ensuring that it adheres to the rules of the blockchain. If the validation is successful, the follower nodes add the new block to their local copies of the blockchain.

The New Block Addition process, as described in Algorithm 4, ensures that new blocks are only added to the blockchain when the block heights are consistent across the network, maintaining the integrity and consistency of the SyncRaft consensus algorithm.

3.4 DNS functionality design

The DNS Functionality Design is built on top of the blockchain, which utilizes the SyncRaft consensus algorithm to provide a secure, efficient, and decentralized domain name system. As shown in Figure 7, the DNS functions are implemented as an application layer above the blockchain, leveraging the robustness of the underlying consensus algorithm. These components work together to provide essential domain name services, as detailed below:

- **Domain Name Registration**: In the distributed DNS system built on the SyncRaft-based blockchain, users can register a domain name by submitting a registration request to one of the nodes. This request should contain the desired domain name, its associated IP address, the owner's identifying information, and the public key that corresponds to their user account. The node then verifies the authenticity of the public key to ensure the request is genuine. Once the key is verified, the server processes the registration request, and the domain name is registered in the blockchain. This approach ensures that domain names are registered securely and transparently.
- **Domain Name Update**: Domain name owners can modify their domain-to-IP address mappings by sending an update request to a node. This request must include the updated IP address, the domain name, the owner's identifying information, and their public key. Similar to the registration process, the server verifies the public key and checks the owner's rights to the domain to ensure that only authorized users can make updates. After the verification is successful, the server processes the update request, and the new IP address mapping is reflected in the blockchain. This design allows for a secure and decentralized way to manage domain name updates.
- **Domain Name Query**: To resolve a domain name to an IP address, users can send a query request to any node participating in the network. Each node maintains a local copy of the blockchain, and upon receiving a query request, the node searches its local copy for the domain name. If the domain name

is found, the node responds with the associated IP address. This decentralized approach to domain name resolution ensures that the system is resilient to failures and censorship while maintaining high query performance.

By leveraging blockchain technology and the robust SyncRaft consensus algorithm, the DNS system built on top of the blockchain guarantees the reliability and accuracy of the distributed DNS system, making it an attractive alternative to traditional, centralized DNS solutions.

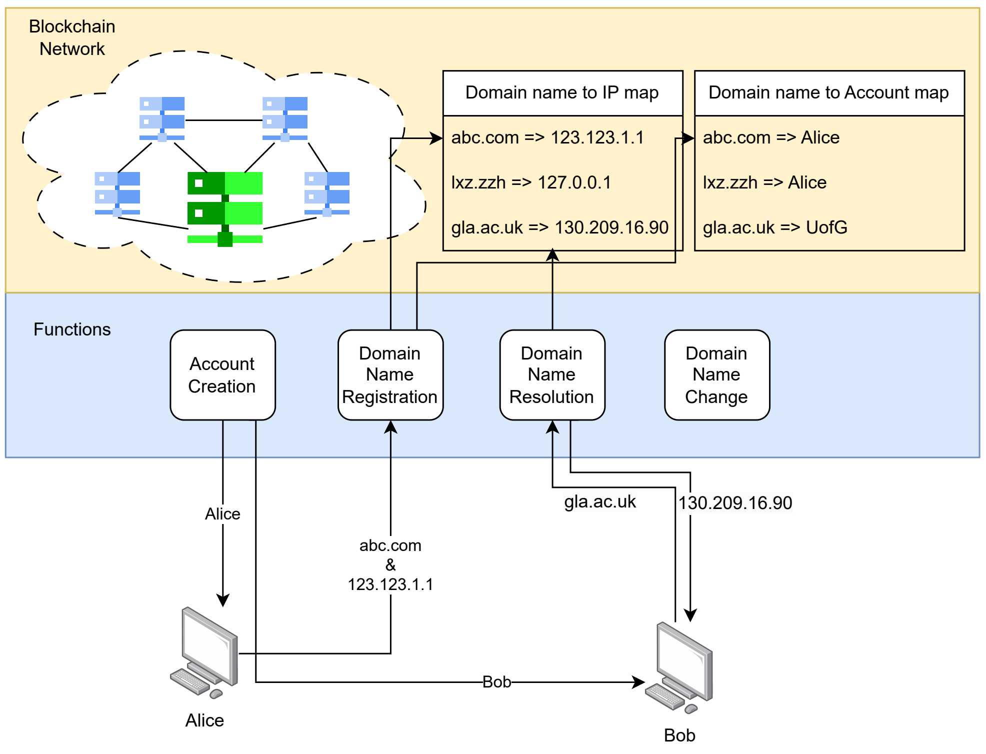


Figure 7: DNS functions of Be-DNS

3.5 Web interface design

The web interface design is essential for ensuring user-friendly interaction with the application. The main elements of the web interface design are:

- **Homepage**: The homepage of the application serves as the central hub from which users can access various features of the DNS system. It should provide clear navigation options to other pages, including query, account management, domain name registration and update, chain and node information.
- **Domain Query Interface**: The domain query interface allows users to search for IP addresses associated with specific domain names. It should provide a simple search box where users can enter the domain name and receive the corresponding IP address as a result.

- **Account Management Interface**: The account management interface is designed to facilitate the creation of new user accounts. It should include fields for users to enter their public keys and a button to create the account. Once the account is created, the interface should display the user's public key, private key, and a confirmation message.
- **Domain Registration and IP Address Update Interface**: The domain registration and IP address update interface should provide users with an intuitive way to register new domain names and update the IP addresses of existing domain names. It should include separate sections for registering a new domain and updating an existing one. Both sections should contain fields for users to enter the required information, such as domain name, IP address, public key, and a random number.
- **Chain and Node Information Interface**: This interface should provide users with an overview of the current state of the blockchain and the nodes participating in the network. It should display information such as the current leader, the height of the blockchain, and a list of participating nodes.
- **Error Handling and User Feedback**: The interface design should incorporate error handling and provide appropriate feedback to users. This includes displaying error messages when necessary, such as when users fail to provide all required information for account creation or domain registration.

4 Implementation

The author utilizes Python as the primary programming language for implementing the SyncRaft-based DNS system. The Flask web framework is employed for creating the web application, while the consensus algorithm and synchronization mechanisms are implemented using Python classes and functions. Additionally, the website's user interface is developed using HTML, CSS, and JavaScript to provide a user-friendly and responsive experience. To organize the code, the project is divided into three parts, focusing on different aspects of the system.

4.1 Backend implementation

This part involves the creation of the blockchain and node classes, as well as the implementation of the consensus algorithm and synchronization mechanisms. This ensures the proper functioning of the SyncRaft protocol and allows nodes to maintain a consistent view of the DNS data.

4.1.1 Basic functions

The basic functions include getTime(), getHash(), and verify(). These are helper functions that are used throughout the implementation.

- **getTime()** - Returns the current timestamp as a formatted string.
- **getHash(data)** - Computes the SHA256 hash of the given input data.
- **verify(key, ran, publickey)** - Checks if the provided key, random number, and public key are valid by recomputing the public key and comparing it with the provided one.

4.1.2 Classes

In the backend implementation, the main components of the SyncRaft-based DNS system are represented by Python classes. These classes encapsulate the functionality of the system and enable efficient organization of the code. The main classes involved in the backend implementation are as follows:

- **Block**: The Block class represents individual blocks within the blockchain. Each block contains an index, a previous hash, a timestamp, a map hash, and a hash of its own content. The class also includes methods for generating the hash of a block.
- **Account**: The Account class represents user accounts within the system, which are used for authentication when adding or changing domain-IP bindings. Each account has a public key, a private key, and a random number (ran). The class includes methods for creating new accounts and generating the necessary keys.
- **Blockchain**: The Blockchain class represents the entire blockchain structure, which consists of a chain of blocks and associated data maps (NametoIpmap and NametoOwnermap). This class con-

tains methods for adding new domain-IP bindings, changing existing bindings, and querying the blockchain for IP addresses. It also includes methods for validating the blockchain and generating the map hashes.

- **Node**: The Node class represents a participant in the SyncRaft network. Each node has its own instance of the Blockchain class and communicates with other nodes to maintain a consistent view of the DNS data. The Node class contains methods for handling various aspects of the consensus algorithm, such as leader election, heartbeat detection, and synchronization with other nodes. It also includes methods for broadcasting changes to the blockchain and recovering from inconsistent states.

These classes provide a solid foundation for the backend implementation of the SyncRaft-based DNS system, allowing for modular and maintainable code. By encapsulating the functionality within these classes, the system becomes easier to understand, debug, and extend as needed.

4.1.3 Block

The Block class represents individual blocks within the blockchain. Each block contains an index, a previous hash, a timestamp, a map hash, and a hash of its own content.

Attributes:

- **index** - The position of the block in the blockchain.
- **previousHash** - The hash of the previous block in the blockchain.
- **timestamp** - The time the block was created.
- **mapHash** - The hash of the current state of the domain name to IP address and domain name owner mappings.
- **hash** - The hash of the block's content, including its index, previous hash, timestamp, and map hash.

Methods:

No specific methods are present in the Block class, but the constructor is used to initialize the block and calculate the hash.

4.1.4 Account

The Account class represents an account in the system. It is responsible for generating and storing the private key, random number, and public key associated with a user.

Attributes:

- **key** - The private key of the user account.
- **ran** - A random number used to generate the public key.
- **publickey** - The public key of the user account, generated using the private key and random number.

Methods:

- **getKey()** - Returns the private key, random number, and public key of the user account.

4.1.5 Blockchain

The Blockchain class is responsible for maintaining the chain of blocks, as well as the mapping of domain names to IP addresses and domain name owners. It includes methods for adding new blocks, verifying the validity of the chain, and querying and modifying domain name bindings.

Attributes:

- **chain** - A list of blocks that make up the blockchain.
- **NametoIpmap** - A dictionary that maps domain names (hashed) to IP addresses.
- **NametoOwnermap** - A dictionary that maps domain names (hashed) to their corresponding owner's public key (hashed).

Methods:

- **getPreviousHash()** - Retrieves the hash of the most recent block in the blockchain.
- **getMapHash()** - Computes the hash of the current state of the domain name to IP address and domain name owner mappings.
- **checkVaild()** - Verifies that the most recent block in the chain is valid by comparing its map hash and previous hash to the current state.
- **addNewBlock()** - Creates and appends a new block to the blockchain.
- **addNewBinding()** - Adds a new domain name to IP address and domain name owner mapping to the dictionaries and updates the blockchain.
- **changeBinding()** - Updates an existing domain name to IP address mapping and the blockchain.
- **queryBinding()** - Retrieves the IP address associated with a given domain name from the mapping.
- **showBlock()** - Displays the content of a specified block in the blockchain.
- **showAllBlock()** - Displays the content of all blocks in the blockchain.

4.1.6 Node

The Node class is responsible for implementing the Raft consensus algorithm and maintaining the state of a node in the system. It includes methods for starting elections, sending heartbeats, and synchronizing the blockchain between nodes.

Attributes:

- **term** - The node's current term in the Raft consensus algorithm.
- **nodes** - A list of all nodes in the network.
- **blockchain** - The node's instance of the Blockchain class.
- **leader** - The current leader of the Raft consensus algorithm.

- **state** - The current state of the node ('candidate', 'synchronizer', or 'follower').
- **url** - The URL of the node.
- **voted** - A boolean indicating whether the node has voted in the current term.
- **votesReceived** - The number of votes received by the node in the current term.
- **received** - A boolean indicating whether the node has received a message in the current term.
- **majority** - The number of votes needed for a node to become a leader (more than half of the total nodes).
- **timeout** - A random timeout value used in the Raft consensus algorithm.
- **timer** - A timer object that triggers node actions based on the timeout value.

Methods:

- **runtime()** - The main loop that runs the node, handling state transitions and executing the appropriate actions for each state.
- **startElection()** - Initiates a new election round, sending vote requests to other nodes and counting received votes.
- **sendHeartbeat()** - Sends heartbeat messages to other nodes while in the 'synchronizer' state to maintain leadership and synchronize data.
- **recover()** - Retrieves the blockchain from the leader and updates the node's blockchain to match.
- **follower()** - Handles the 'follower' state logic, waiting for messages and transitioning to 'candidate' state if no messages are received.
- **broadcastBlock()** - Sends the most recent block to all other nodes in the network to synchronize data.
- **getData()** - Returns information about the node, including the current leader, the height of the blockchain, and the list of nodes.

4.2 Web application and routing

The web application serves as a user interface for the decentralized DNS system, enabling users to interact with the blockchain and nodes. The application uses Flask, a lightweight and flexible Python web framework, for creating routes, managing user input, and handling interactions with the backend components.

4.2.1 Initialization

Import the necessary libraries, such as Flask, and initialize the blockchain and node instances. Configure the Flask app by setting the IP address, port, and template folder for rendering HTML templates.

1. Import Flask, JSON, and other required libraries.
2. Create a Flask app instance and set the template folder, static folder, and static URL path.

3. Initialize the blockchain and node classes.
4. Configure the desired IP address and port.

4.2.2 Routes

The Flask app uses routes to handle user requests and render appropriate templates. The main routes include:

- **Main page**: A route for the main page, rendering the "index.html" template.
- **Query page**: A route for the query page, rendering the "query.html" template, which allows users to search for IP addresses associated with domain names.
- **Account page**: A route for the account page, rendering the "account.html" template. This page allows users to create new accounts.
- **Add & Change page**: A route for the add/change page, rendering the "add&change.html" template. Users can add new domain name bindings or update existing ones on this page.
- **Chain & Node page**: A route for the chain/node page, rendering the "chain&node.html" template. This page displays information about the blockchain, such as the current leader, height, and connected nodes.

4.2.3 POST Request Handling

Define routes for handling POST requests to create accounts, add new domain name bindings, and change existing bindings. These routes interact with the blockchain and node classes to update the system state and broadcast changes to other nodes in the network.

- **Create account**: A route to handle account creation requests.
- **Add domain name binding**: A route to handle requests for adding new domain name bindings.
- **Change domain name binding**: A route to handle requests for changing existing domain name bindings.

4.2.4 Consensus-Related Routes

Define routes for handling SyncRaft consensus-related actions, such as voting requests, heartbeats, and receiving new blocks from other nodes. These routes facilitate communication between nodes and ensure the proper functioning of the SyncRaft protocol.

- **Voting requests**: A route for handling voting requests among nodes.
- **Heartbeats**: A route for handling heartbeat messages among nodes.
- **Receiving new blocks**: A route for receiving new blocks from other nodes and updating the local blockchain accordingly.

4.2.5 Data Retrieval and Synchronization

Define routes to retrieve chain data and synchronize with other nodes. These routes are used when recovering from failures or joining the network.

- **Get chain data**: A route for retrieving the chain data, including the blockchain, domain name to IP address map, and domain name to owner map.
- **Synchronize**: A route for synchronizing with other nodes, ensuring that a new node joining the network or a recovering node can obtain the most recent blockchain state.

4.2.6 Web Application Deployment

Finally, deploy the web application by running the Flask app instance. This serves the web application to users, enabling them to interact with the decentralized DNS system and perform actions such as creating accounts, adding domain name bindings, and querying IP addresses associated with domain names.

1. Run the Flask app instance.
2. Configure the Flask app to listen on the desired IP address and port.

To summarize, the web application and routing component provide a user interface for interacting with the decentralized DNS system. The Flask framework is used to create application routes, manage user input, and handle interactions with the blockchain and node classes. The web application also plays a crucial role in the Raft consensus process, enabling nodes to communicate and synchronize with each other to maintain a consistent view of the DNS data.

4.3 Frontend user interface

The user interface is designed using HTML, CSS, and JavaScript to provide a visually appealing and user-friendly experience. This part includes creating templates for various web pages, ensuring responsiveness, and providing an intuitive interface for users to interact with the system. At a high level, the Frontend user interface consists of five main pages: the Home page, the Account page, the Query page, the Add & Change page, and the Chain & Node page. Each page serves a specific purpose and provides users with the necessary tools and information to perform their tasks.

- **Home**: The Home page is the starting point for users and provides them with an overview of the Be-DNS system. It includes information about the system's purpose and functionality, as well as links to the other main pages.
- **Account**: The Account page allows users to create and manage their Be-DNS accounts. Users can create new accounts, which can be used in IP binding.
- **Add & Change**: The Add & Change page allows users to add new domain names to the Be-DNS

system or modify existing ones. Users can enter the necessary information for the domain name, such as the name and IP address, and submit it to the system. The page also includes feedback mechanisms to inform users of any errors or issues with their submission.

- **Chain & Node**: The Chain & Node page provides users with information about the Be-DNS blockchain network, such as the current synchronizer and block height. Users can also view a list of current nodes on the network. The page is designed to provide users with transparency and visibility into the underlying blockchain technology that powers the Be-DNS system.
- **Query**: The Query page allows users to search for domain names and obtain their corresponding IP addresses. Users can enter a domain name into the search bar and receive the corresponding IP address as a result. The page also includes feedback mechanisms to inform users of any errors or issues with their search.

Overall, the Frontend user interface plays a critical role in the Be-DNS system, allowing end-users to interact with the system and perform tasks easily and efficiently. By providing clear and intuitive navigation and feedback mechanisms, the interface ensures a smooth and user-friendly experience for all users.

5 Analysis and Results

In this section, the performance of Be-DNS, which is built upon the SyncRaft consensus algorithm, is rigorously examined, concentrating on aspects such as latency, throughput, and fault tolerance. The evaluation is carried out on a 5-node Be-DNS network deployed on a personal laptop, with each node operating on ports within the range of 5000 to 5004. The latency and throughput of Be-DNS under specific conditions are assessed. A comparative analysis fault tolerance of SyncRaft is conducted against other prominent consensus algorithms, including PoW, PoS, PBFT, Paxos, and Raft. Furthermore, an in-depth illustration of the user experience while interacting with the Be-DNS service is presented.

5.1 Latency

The latency plot in Figure 8 presents the distribution of latencies observed during the experiment. The experiment consisted of sending 1000 requests to add new blocks to the blockchain. For each request, the author measured the time elapsed between sending the request and receiving the response, which represents the latency. The mean latency was found to be 2046.256 ms, with a median of 2047.577 ms and a standard deviation of 8.862 ms. The plot indicates that the latencies are relatively consistent, with a small standard deviation. This consistency is an essential factor for maintaining network stability and ensuring timely updates to the blockchain.

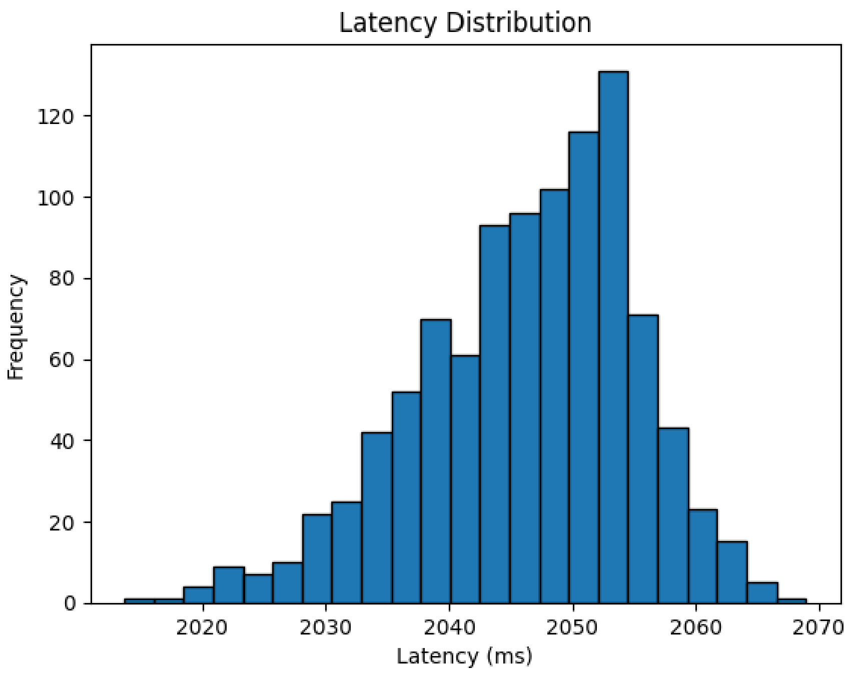


Figure 8: Latency distribution of requests in a 5 nodes Be-DNS

5.2 Throughput

In the context of consensus algorithms, throughput is commonly defined as the number of transactions or operations processed per unit of time. Within the scope of the SyncRaft implementation used in Be-DNS, each operation, such as adding or modifying a record, is encapsulated within a block. As a result, the throughput is calculated based on the inverse relationship with latency, which is quantified in milliseconds. This approach enables a comprehensive assessment of the algorithm's performance by evaluating the number of block operations completed within a specific time frame.

The general equation for throughput can be represented as:

$$Throughput = \frac{Number\ of\ transactions}{Time} \tag{1}$$

The modified equation for SyncRaft is as follows:

$$Throughput_{SyncRaft} = \frac{1}{Latency\ (ms) \times 10^{-3}} \tag{2}$$

Various quantities of requests are supplied to the system, ranging from 500 to 5000 in increments of 500. The latency of each request is recorded, and the average latency for each request quantity is calculated. The throughput for each request quantity is computed using Equation 2. The throughput plot depicted in Figure 9 reveals that the throughput of Be-DNS remains consistent at approximately 0.49, irrespective of the request quantity. These throughput results demonstrate that the SyncRaft consensus algorithm can efficiently manage a moderate number of requests within a specified timeframe, delivering stable and consistent responses.

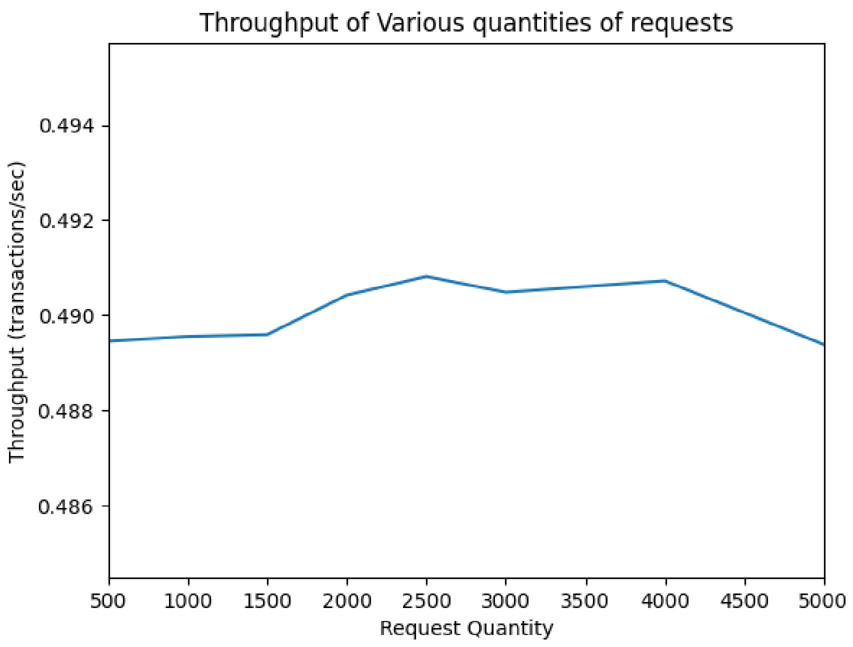


Figure 9: Throughput of various request quantities in a 5 nodes Be-DNS

5.3 Fault tolerance

Figure 10 showcases the fault tolerance of various consensus algorithms as the number of nodes in the network increases. In the context of consensus algorithms, fault tolerance is a measure of the percentage of nodes that can fail or become compromised while the system remains operational. A higher fault tolerance percentage indicates better resilience to failures.

The fault tolerance of SyncRaft, Paxos, and Raft is identical, as they all tolerate the failure of up to $\frac{n-1}{2}$ nodes, where n is the total number of nodes in the network. As the number of nodes increases, the fault tolerance percentage for these algorithms increases, approaching 50%. On the other hand, PoW and PoS both exhibit a constant fault tolerance of 50%, regardless of the number of nodes in the network. PBFT shows a different trend, with a fault tolerance of $\frac{n-1}{3}$ nodes, resulting in a increases fault tolerance percentage as the number of nodes increases, eventually approaching 33.3%.

These results suggest that while SyncRaft provides a similar level of fault tolerance to Paxos and Raft, it outperforms PoW and PoS in terms of consistency and efficiency, making it a viable alternative for applications requiring a balance between performance and fault tolerance.

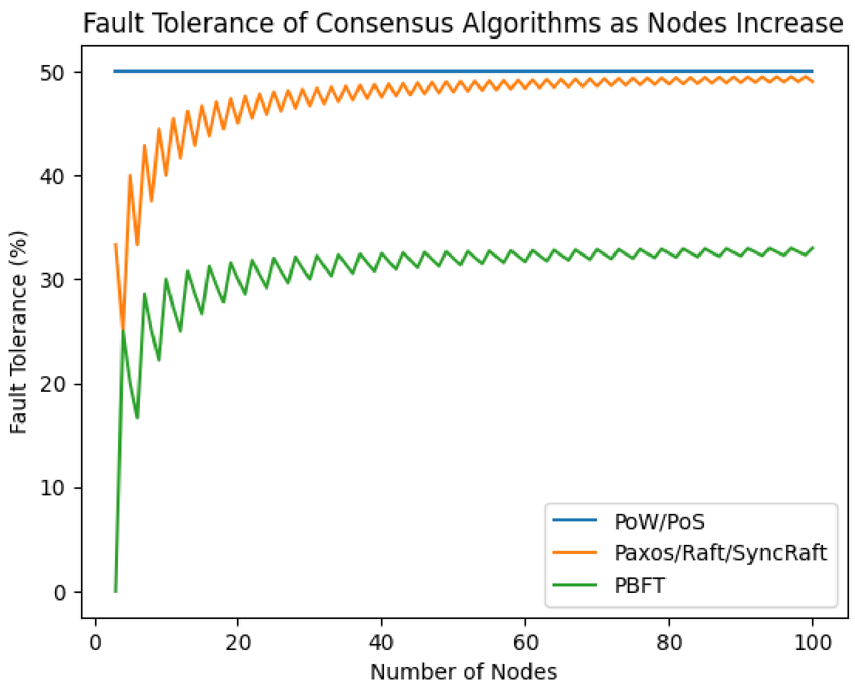


Figure 10: Fault tolerance of various consensus algorithms

5.4 Comparison of Consensus Algorithms

In order to evaluate the performance and characteristics of SyncRaft in relation to other popular consensus algorithms, a comprehensive comparison has been conducted, taking into account several key factors that influence consensus performance. Table 2 illustrates the key differences among PoW, PoS, PBFT, Paxos, Raft, and SyncRaft in terms of decentralization, energy consumption, scalability, throughput, security, and fault tolerance.

Analyzing the results, SyncRaft exhibits a medium level of decentralization, which is an improvement over the Raft algorithm and offers a better balance between decentralization and efficiency. Its energy consumption is low, making it more environmentally friendly compared to PoW and on par with PoS, PBFT, Paxos, and Raft. In terms of scalability, SyncRaft performs at a medium level, indicating its potential to handle a moderate number of nodes without compromising performance. The throughput of SyncRaft is also medium, reflecting its capacity to process transactions at a reasonably fast pace. Regarding security, SyncRaft has a medium level of protection against attacks, making it less vulnerable than traditional Raft but not as secure as PoW or PoS. Lastly, SyncRaft demonstrates medium fault tolerance, indicating its ability to maintain network functionality in the face of node failures or other issues.

This comparison aims to provide insights into the relative strengths and weaknesses of SyncRaft, as well as to identify potential areas of improvement for the Be-DNS system. By understanding these performance characteristics, it becomes possible to make informed decisions about the suitability of SyncRaft for various applications and to guide future development efforts.

Table 2: Comparison of Blockchain Consensus Algorithms

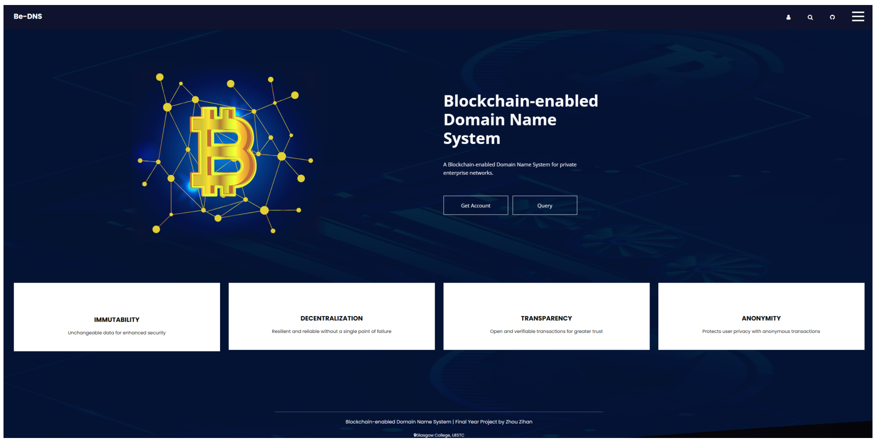
	PoW	**PoS**	**PBFT**	**Paxos**	**Raft**	**SyncRaft**
Decentralization	High	High	Medium	Low	Low	Medium
Energy Consumption	High	Medium	Medium	Low	Low	Low
Scalability	High	Medium	Medium	High	Low	Medium
Throughput	Low	Medium	High	High	High	Medium
Security	High	High	High	Medium	Medium	Medium
Fault Tolerance	High	High	Low	Medium	Medium	Medium

5.5 Be-DNS Interface and Functionality

The Be-DNS system provides a user-friendly interface, consisting of several pages with specific functions, to simplify the domain registration and management process. These pages are visually represented in a series of screenshots, highlighting the key features and functionality of Be-DNS.

5.5.1 Homepage

Figure 11 showcases the homepage of the Be-DNS system, which provides an overview of the system and its functionalities and features. The homepage includes a brief introduction to Be-DNS, emphasizing its decentralized nature and robust security. Additionally, it highlights the key functions that users can access, such as account creation and querying. By this homepage, users can easily navigate the Be-DNS system and take advantage of its various features and capabilities.



Blockchain-enabled Domain Name System

A Blockchain-enabled Domain Name System for private enterprise networks.

IMMUTABILITY
Unchangeable data for enhanced security

DECENTRALIZATION
Resilient and reliable without a single point of failure

TRANSPARENCY
Open and verifiable transactions for greater trust

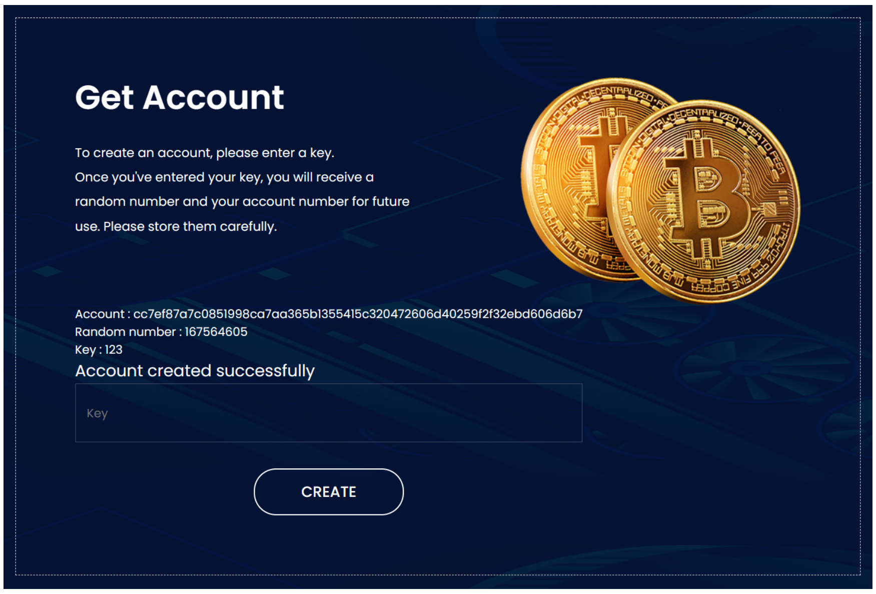
ANONYMITY
Protects user privacy with anonymous transactions

Blockchain-enabled Domain Name System | Final Year Project by Zhou Zihan
Glasgow College, UESTC

Figure 11: Homepage of Be-DNS

5.5.2 Account Creation

The user journey begins with account creation, as shown in Figure 12, where the user is guided through a simple process to set up their unique account by giving a personal key and receiving an account, a random nonce.



Get Account

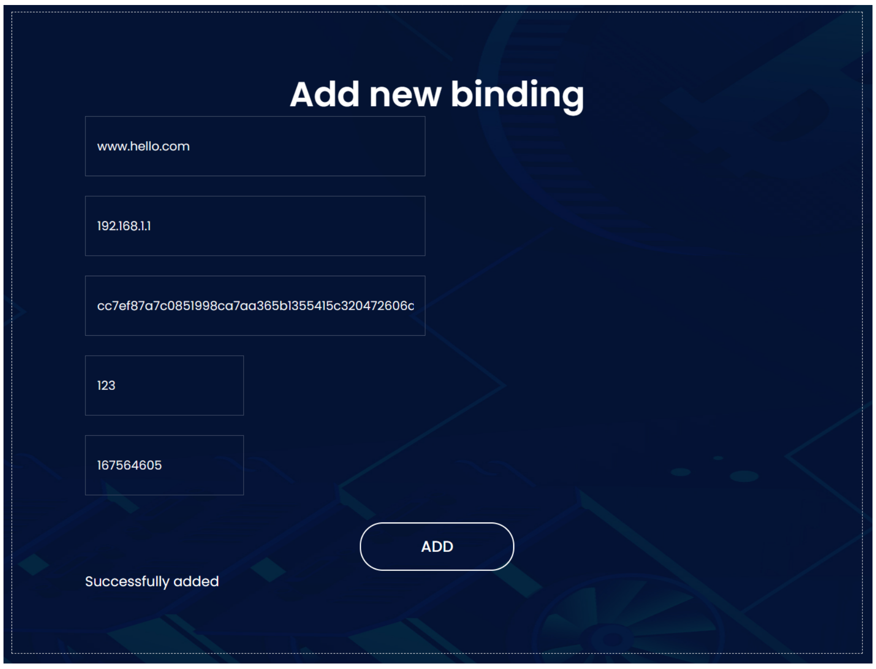
To create an account, please enter a key. Once you've entered your key, you will receive a random number and your account number for future use. Please store them carefully.

Account : cc7ef87a7c0851998ca7aa365b1355415c320472606d40259f2f32ebd606d6b7
Random number : 167564605
Key : 123
Account created successfully

Figure 12: Account creation in Be-DNS

5.5.3 Domain Name Registration

Following successful account creation, the user can proceed to the domain name registration stage, as displayed in Figure 13. They can input their desired domain name and corresponding IP address, along with the necessary ownership details.



www.hello.com

192.168.1.1

cc7ef87a7c0851998ca7aa365b1355415c320472606c

123

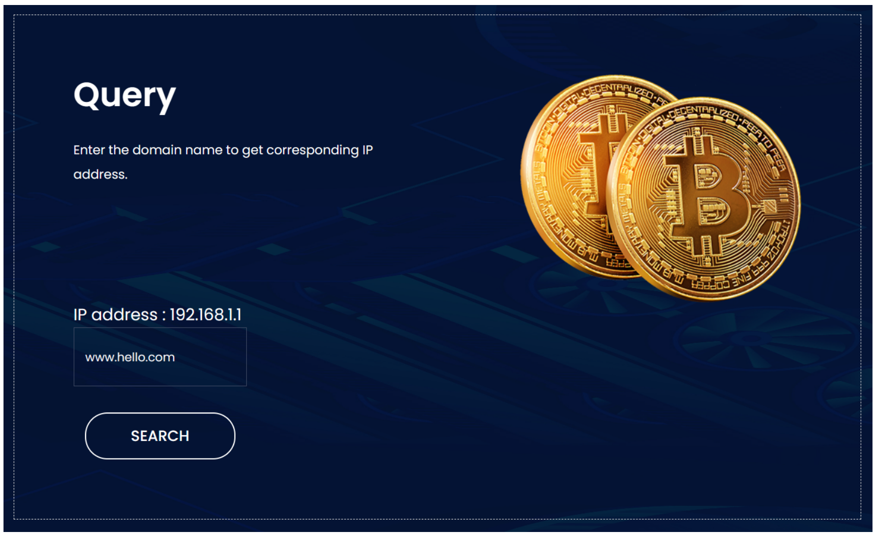
167564605

Successfully added

Figure 13: Domain name registration in Be-DNS

5.5.4 Domain Name Query

Users can perform query for the domain name and obtain the associated IP address, as demonstrated in Figure 14. This is the one of the major functions of Be-DNS.



Enter the domain name to get corresponding IP address.

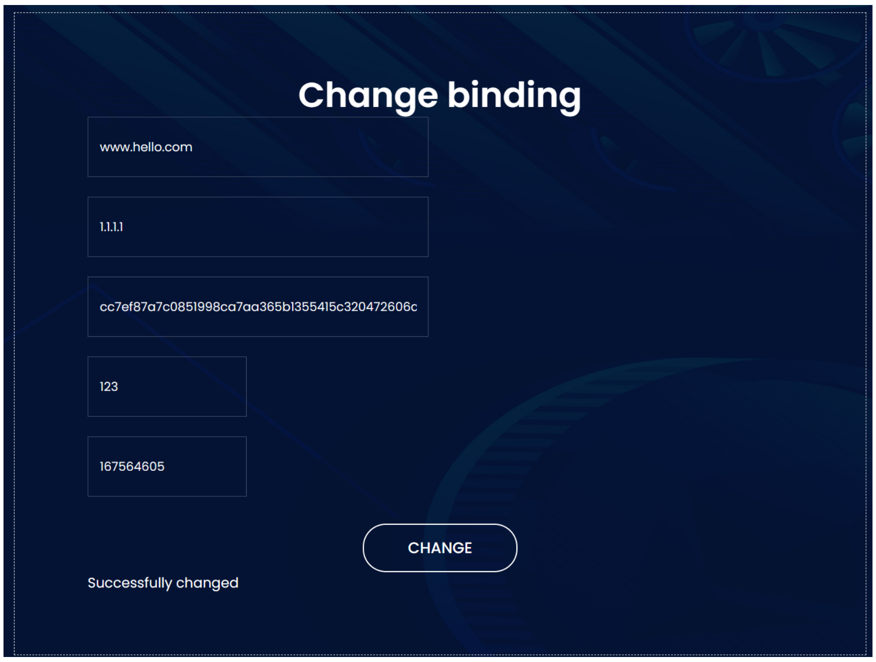
IP address : 192.168.1.1

www.hello.com

Figure 14: Domain name query in Be-DNS

5.5.5 Domain Name Update

The user can update the IP address associated with a domain name through the domain name update interface, as shown in Figure 15. It allows user to update the new IP address and submit the changes after the account verification.



www.hello.com

1.1.1.1

cc7ef87a7c0851998ca7aa365b1355415c320472606c

123

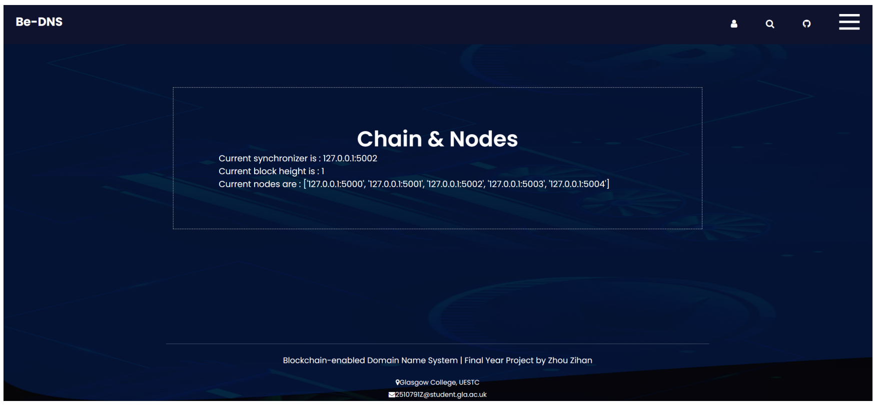
167564605

Successfully changed

Figure 15: Domain name update in Be-DNS

5.5.6 Blockchian and Network information

Offering information about synchronizer, block hight and current nodes, this page, as shown in Figure 16, allows users to better understand the decentralized nature of the Be-DNS system and how it maintains the integrity and security of domain name data.



Current synchronizer is : 127.0.0.1:5002
Current block height is : 1
Current nodes are : ['127.0.0.1:5000', '127.0.0.1:5001', '127.0.0.1:5002', '127.0.0.1:5003', '127.0.0.1:5004']

Blockchain-enabled Domain Name System | Final Year Project by Zhou Zihan

Glasgow College, UESTC
2510791Z@student.gla.ac.uk

Figure 16: Chain and Node information in Be-DNS

5.6 Summary

The Analysis and Results section examines the performance of the Be-DNS system, which utilizes the SyncRaft consensus algorithm, in terms of latency, throughput, and fault tolerance. A 5-node network is used for evaluation, and the results show consistent latencies and a moderate throughput of approximately 0.49. SyncRaft's fault tolerance is compared with other consensus algorithms, demonstrating a similar level of resilience to Paxos and Raft. Additionally, a comprehensive comparison of PoW, PoS, PBFT, Paxos, Raft, and SyncRaft is presented based on several key factors. The Be-DNS system's user experience is demonstrated through a series of screenshots showcasing the registration, management, and querying of domain names. Overall, the results suggest that SyncRaft is a viable alternative for applications requiring a balance between performance and fault tolerance, and Be-DNS offers a user-friendly interface with robust security and decentralization features.

6 Conclusions

In this paper, Be-DNS, a novel domain name system based on the SyncRaft consensus algorithm is proposed. The primary objective of Be-DNS is to enhance the security, reliability, and performance of traditional DNS by leveraging the strengths of blockchain technology. This section discusses the implications of the findings, potential limitations, and future research directions in the context of Be-DNS.

The performance analysis of Be-DNS focused on three crucial aspects: latency, throughput, and fault tolerance. In terms of latency, Be-DNS achieves relatively low latency, which is crucial for maintaining responsiveness in domain name resolution processes. The throughput of Be-DNS, calculated as the inverse of latency, indicates the system's ability to process a significant number of operations per unit of time. The fault tolerance analysis shows that Be-DNS maintains a reasonable level of fault tolerance, ensuring the system's robustness against node failures.

One of the primary advantages of Be-DNS is its decentralized nature, which enhances security and reduces the likelihood of single points of failure. Be-DNS exhibits medium decentralization levels, which is a balance between the high decentralization of PoW and PoS systems and the low decentralization of Paxos and Raft. This balance ensures that Be-DNS remains resilient against various attacks, such as DDoS attacks, while not sacrificing performance.

Another critical aspect of Be-DNS is its security. Compared to traditional DNS, the decentralized nature of Be-DNS offers several advantages in terms of security and resilience. Traditional DNS systems are vulnerable to attacks such as DDoS, cache poisoning, and man-in-the-middle, which can result in disruption of services and theft of sensitive data. By leveraging the power of blockchain technology, Be-DNS creates a distributed, tamper-proof, and transparent system that is inherently more secure.

In terms of security, Be-DNS is comparable to PoW, PoS, and PBFT, offering a medium level of security. This is achieved by combining the security features of blockchain technology with the SyncRaft consensus algorithm. Be-DNS maintains network consistency through a single synchronizer node while allowing each node to add new blocks, ensuring the proper functioning of the synchronizer and maintaining consistency among nodes in the network.

6.1 Suggestions for further work

Despite the promising results obtained from Be-DNS, there are some limitations to consider. First, the Be-DNS system was tested on a small-scale network of 5 nodes deployed on a personal laptop. While this provided valuable insights into the performance of Be-DNS, it is essential to evaluate the system on larger networks with more diverse node configurations and under various network conditions to validate the robustness and scalability of Be-DNS.

Second, the SyncRaft consensus algorithm, which forms the basis of Be-DNS, may be vulnerable to certain types of attacks, such as Sybil attacks. To mitigate these risks, future research should explore possible improvements to the SyncRaft consensus algorithm or investigate the integration of additional security mechanisms.

Finally, Be-DNS is a relatively new concept that has not yet been extensively tested in real-world scenarios. Future research should focus on developing practical implementations of Be-DNS and evaluating its performance in real-world settings, such as enterprise networks and large-scale domain name resolution systems.

In conclusion, Be-DNS represents a promising alternative to traditional DNS systems by leveraging the strengths of blockchain technology and the SyncRaft consensus algorithm. The performance analysis indicates that Be-DNS achieves competitive latency, throughput, and fault tolerance while offering medium levels of decentralization, security, and scalability. Future research should address the limitations identified in this paper and explore practical implementations of Be-DNS to validate its potential as a reliable, secure, and efficient domain name system.

References

[1] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, “What will 5g be?” *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[4] S. Al-Saqqar, O. Maennel, and R. Bush, “Blockchain for dns security: A survey,” *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 16–21, 2017.

[5] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 1999, pp. 173–186. [Online]. Available: https://dl.acm.org/doi/10.5555/296806.296824

[7] L. Lamport, “The implementation of reliable distributed multiprocess systems,” *Computer Networks (1976)*, vol. 2, no. 2, p. 95–114, 1978.

[8] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” in *Proceedings of the 2014 ACM conference on Computer and communications security*, 2014, pp. 97–107. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2810103.2813685

[9] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, 2014, pp. 305–320. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[10] W. Liu, Y. Zhang, L. Liu, S. Liu, H. Zhang, and B. Fang, “A secure domain name resolution and management architecture based on blockchain,” in *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1–7.

[11] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, “Blockchain challenges and opportunities: a survey,” *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[12] F. Schneider and S. Merz, “A consensus algorithm survey and classification,” in *2017 IEEE International Conference on Computer and Information Technology (CIT)*. IEEE, 2017, pp. 313–320.

[13] P. Vigna and M. J. Casey, *The age of cryptocurrency: How bitcoin and digital money are challenging the global economic order*. St. Martin's Press, 2015.

[14] W. Mougayar, *The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology*. John Wiley & Sons, 2016.

[15] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," *Fast Software Encryption*, pp. 371–388, 2011.

[16] S. M. H. Bamakan, A. Motavali, and A. Babaei Bondarti, "A survey of blockchain consensus algorithms performance evaluation criteria," *Expert Systems with Applications*, vol. 154, p. 113385, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420302098

[17] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 357–379, 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3133956.3133986

[18] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998. [Online]. Available: https://dl.acm.org/doi/10.1145/279227.279229

[19] M. Swan, *Blockchain: Blueprint for a New Economy*. O'Reilly Media, Inc., 2015.

[20] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," pp. 557–564, 2017.

[21] V. Buterin. (2015) On public and private blockchains. [Online]. Available: https://ethereum.stackexchange.com/questions/135/what-is-the-difference-between-a-private-and-a-public-blockchain

[22] C. Guo, Z. Zhou, H. Xu, Y. Fan, X. Zhang, and L. Zhang, "Besharing: A copyright-aware blockchain-enabled knowledge sharing platform," in *2022 4th Conference on Blockchain Research and Applications for Innovative Networks and Services (BRAINS)*, 2022, pp. 49–50.

[23] S. Chakraborty, A. Anand, D. Kalash, and A. Srivastava, "Transparency in carbon credit by automating data-management using blockchain," in *2022 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*, 2022, pp. 1–5.

[24] A. Abuhashim and C. C. Tan, "Smart contract designs on blockchain applications," in *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1–4.

[25] D. Tapscott and A. Tapscott, *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world*. Penguin, 2016.

[26] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," *Financial Cryptography and Data Security*, vol. 9604, pp. 106–125, 2016.

[27] C. Mora, R. Rollins, D. D. Deheyn, and T. A. White, "Bitcoin's energy consumption-a shift to more sustainable options is needed," *Nature Sustainability*, vol. 1, no. 12, pp. 756–758, 2018.

[28] A. Zohar, "Interoperability in blockchain networks," *Communications of the ACM*, vol. 62, no. 7, pp. 60–68, 2019.

[29] A. Walch, "The path of the blockchain lexicon (and the law)," *Review of Banking & Financial Law*, vol. 36, no. 713, pp. 713–765, 2017.

[30] A. Zohar, "Privacy in blockchain networks," *Communications of the ACM*, vol. 62, no. 6, pp. 56–63, 2019.

[31] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," *Technical Report, Draft Version*, vol. 0, no. 9, 2016.

[32] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. ACM, 2018, pp. 245–254.

[33] P. Mockapetris, "Domain names–concepts and facilities," *RFC-1034*, 1987.

[34] M. Dooley and T. Rooney, *Introduction to the Domain Name System (DNS)*, 2017, pp. 17–29.

[35] P. Albitz and C. Liu, *DNS and BIND*. O'Reilly Media, Inc., 2001.

[36] V. Ramasubramanian, "Understanding the domain name system," *IEEE Internet Computing*, vol. 9, no. 4, pp. 73–76, 2005.

[37] M. H. Jalalzai, W. B. Shahid, and M. M. W. Iqbal, "Dns security challenges and best practices to deploy secure dns with digital signatures," in *2015 12th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2015, pp. 280–285.

A Appendices

```python
# Import required libraries
from dataclasses import dataclass
from hashlib import sha256
from datetime import datetime
import random
import threading
import time
import requests
import json

# Define the list of nodes
NODES =
    [’127.0.0.1:5000’,’127.0.0.1:5001’,’127.0.0.1:5002’,’127.0.0.1:5003’,’127.0.0.1:5004’]

def getTime():
    now = datetime.now()
    time = now.strftime("%Y-%m-%d, %H:%M:%S")
    return time

def getHash(data):
    return sha256(data.encode()).hexdigest()

def verify(key, ran, publickey):
    pk = getHash(str(ran) + str(key))
    if(pk == str(publickey)):
        return True
    else:
        return False

class blockchain:
    def __init__(self):
        self.chain = []
        self.NametoIpmap = {}
        self.NametoOwnermap = {}
        genesisBlock = block(0, 0x0, "hello world", 0x0)
        self.chain.append(genesisBlock)

    def getPreviousHash(self):
        return self.chain[-1].hash

    def getMapHash(self):
        mapHash = getHash(str(self.NametoIpmap) + str(self.NametoOwnermap))
        return mapHash

    def checkVaild(self):
        check = (self.chain[-1].mapHash ==
            self.getMapHash())&(self.chain[-1].previousHash ==
            self.chain[-2].hash)
        return check

    def addNewBlock(self):
        newBlock = block(len(self.chain), self.getPreviousHash(), getTime(),
            self.getMapHash())
        self.chain.append(newBlock)

    def addNewBinding(self, domainName, ip, owner, key, ran):
        if (verify(key, ran, owner)):
            if (self.NametoIpmap.get(getHash(domainName), 'not exist') !=
                'not exist'):
                return 'Domain Name used'
            self.NametoIpmap[getHash(domainName)] = str(ip)
            self.NametoOwnermap[getHash(domainName)] = getHash(owner)
            self.addNewBlock()
            return 'Successfully added'
        else:
            return 'incorrect key or account'

    def changeBinding(self, domainName, Newip, owner, key, ran):
        if (verify(key, ran, owner)):
            if (self.NametoOwnermap.get(getHash(domainName), 'not exist') ==
                'not exist'):
                return 'Domain Name not exist'
            if (self.NametoOwnermap.get(getHash(domainName), 'not exist') !=
                getHash(owner)):
                return 'invaild user'
            self.NametoIpmap[getHash(domainName)] = str(Newip)
```

```
                self.addNewBlock()
                return 'Successfully changed'
            else:
                return 'incorrect key or account'

    def queryBinding(self, domainName):
        # debug 20230321
        print(self.NametoIpmap, type(self.NametoIpmap))
        if (self.checkVaild == False):
            return 'invaild blockchain log'
        if self.NametoIpmap.get(getHash(domainName), 'not exist') == 'not
           exist':
            return 'Corresponding IP address does not exist'
        return self.NametoIpmap[getHash(domainName)]

    def showBlock(self, index):
        selected_block = self.chain[index]
        print('{')
        print('index: ' + str(selected_block.index))
        print('previousHash: ' + str(selected_block.previousHash))
        print('timestamp: ' + str(selected_block.timestamp))
        print('mapHash: ' + str(selected_block.mapHash))
        print('} => Hash: ' + str(selected_block.hash))
        print('-'*80)

    def showAllBlock(self):
        for i in range(len(self.chain)):
            self.showBlock(i)

class block:
    def __init__(self, index, previousHash, timestamp, mapHash):
        self.index = index
        self.previousHash = previousHash
        self.timestamp = timestamp
        self.mapHash = mapHash
        self.hash =
           getHash(str(index)+str(previousHash)+str(timestamp)+str(mapHash))

class account:
    def __init__(self, key):
        ran = random.randint(1,1000000000)
        publickey = getHash(str(ran) + str(key))
        self.key = key
        self.ran = ran
        self.publickey = publickey

    def getKey(self):
        return self.key, self.ran, self.publickey


class node:
    def __init__(self, blockchain, url):
        self.term = 0
        self.nodes = NODES
        self.blockchain = blockchain
        self.leader = None
        self.state = 'candidate'
        self.url = url
        self.voted = False
        self.votes_received = 0
        self.received = True
        self.majority = ((len(self.nodes) - 1) // 2)
        self.timeout = random.uniform(1, 5)
        self.timer = threading.Timer(self.timeout, self.runtime)
        self.timer.start()

    def runtime(self):
        time.sleep(5)
        while True:
            if self.state == 'candidate':
                self.start_election()
            elif self.state == 'leader':
                self.send_heartbeat()
            elif self.state == 'follower':
                self.follower()
            time.sleep(1)

    def start_election(self):
```

```python
        self.term += 1
        election = {
            'term':self.term,
            'url':self.url,
        }
        for node in self.nodes:
            if node == self.url:
                continue
            response = requests.post(f'http://{node}/vote_request',
                json=election)
            if response.status_code == 200:
                if response.json()['vote'] == 'yes':
                    self.votes_received += 1

        if self.votes_received >= self.majority:
            self.leader = self.url
            self.state = 'leader'
        else:
            self.state = 'follower'

    def send_heartbeat(self):
        while self.state == 'leader':
            heart_received = 0
            time.sleep(5+random.uniform(1, 2))
            leader = {
                'height': len(self.blockchain.chain),
                'hash': self.blockchain.getMapHash,
                'term': self.term,
                'url':self.url
            }
            for node in self.nodes:
                if node == self.url:
                    continue
                response = requests.post(f'http://{node}/receive_heartbeat',
                    data=leader)
                if response.status_code == 200:
                    heart_received += 1

            if heart_received < self.majority:
                self.state = 'candidate'

    def recover(self):
        response = requests.get(f'http://{self.leader}/get_chain')
        if response.status_code == 200:
            self.blockchain.chain = response['chain']
            self.blockchain.NametoIpmap = response['NametoIpmap']
            self.blockchain.NametoOwnermap = response['NametoOwnermap']

    def follower(self):
        self.voted = False
        while True:
            self.received == False
            time.sleep(10+random.uniform(1, 2))
            if self.received == False:
                self.state = 'candidate'

    def broadcast_block(self):
        newblock = self.blockchain.chain[-1]
        newNametoIpmap = self.blockchain.NametoIpmap
        newNametoOwnermap = self.blockchain.NametoOwnermap
        data = {
            'url':self.url,
            'index':newblock.index,
            'previousHash':newblock.previousHash,
            'timestamp':newblock.timestamp,
            'mapHash':newblock.mapHash,
            'hash':newblock.hash,
            'newNametoIpmap': json.dumps(newNametoIpmap),
            'newNametoOwnermap': json.dumps(newNametoOwnermap),
        }
        for node in self.nodes:
            if node == self.url:
                continue
            response = requests.post(f'http://{node}/receive_block',
                data=data)

    def getData(self):
        if self.leader != None:
            data = {
                'leader': self.leader,
                'height':len(self.blockchain.chain),
                'nodes':self.nodes,
            }
        else:
            data = {
                'leader': 'in election',
                'height':len(self.blockchain.chain),
                'nodes':self.nodes,
            }
        return data
```

Listing 1: Blockchain classes

```
# Import necessary libraries
from flask import Flask, render_template, request, url_for, jsonify
import json
from blockchain import*
# Create Flask app and initialize blockchain and node
app = Flask(__name__, template_folder =
    'templates',static_folder='static',static_url_path='')
selfChain = blockchain()

# Input the desired IP address and port
host = '127.0.0.1'
port = 5000
ip = f'{host}:{port}'
selfNode = node(selfChain, ip)

# Route to the main page
@app.route('/', methods=['GET'])
def Index():
    return render_template("index.html")

# Route to the query page
@app.route('/query', methods=['GET'])
def Query():
    return render_template("query.html")

```

```
# Route to the account page
@app.route('/account', methods=['GET'])
def Account():
    return render_template("account.html")

# Route to the add & change page
@app.route('/add&change', methods=['GET'])
def add_change():
    data = selfNode.getData()
    leader = data['leader']
    return render_template("add&change.html", var5=leader)

# Route to the chain & node page
@app.route('/chain&node', methods=['GET'])
def chain_node():
    data = selfNode.getData()
    leader = data['leader']
    height = data['height']
    nodes = data['nodes']
    return render_template("chain&node.html", var1=leader, var2 = height,
        var3 = nodes)

# Route to create an account
@app.route('/create', methods=['POST'])
def createAccount():
    key = request.form['key']
    if(key == ''):
        message = 'Please enter a key'
        return render_template("account.html", var4=message)
    else:
        accountcreated = account(key)
        address = accountcreated.publickey
        ran = accountcreated.ran
        message = 'Account created successfully'
        return render_template("account.html", var1=address, var2=ran,
            var3=key, var4=message)

# Route to add a new domain name and IP address binding
@app.route('/add', methods=['POST'])
```

```python
def add():
    domain = request.form['domain']
    ip = request.form['ip']
    owner = request.form['owner']
    key = request.form['key']
    ran = request.form['ran']
    if((domain == '')|(ip == '')|(owner == '')|(key == '')|(ran == '')):
        message = 'Please enter the information'
        return render_template("add&change.html", var1 = message)
    else:
        message = selfChain.addNewBinding(domain, ip, owner, key, ran)
        if message == 'Successfully added':
            selfNode.broadcast_block()
        return render_template("add&change.html", var1 = message)

# Route to change the IP address for an existing domain name
@app.route('/change', methods=['POST'])
def change():
    domain = request.form['newdomain']
    newip = request.form['newip']
    owner = request.form['newowner']
    key = request.form['newkey']
    ran = request.form['newran']
    if((domain == '')|(newip == '')|(owner == '')|(key == '')|(ran == '')):
        message = 'Please enter the information'
        return render_template("add&change.html", var2 = message)
    else:
        message = selfChain.changeBinding(domain, newip, owner, key, ran)
        if message == 'Successfully changed':
            selfNode.broadcast_block()
        return render_template("add&change.html", var2 = message)

# Route to search for an IP address associated with a domain name
@app.route('/search', methods=['POST'])
def search():
    domain = request.form['domain']
    ip = selfChain.queryBinding(domain)
    return render_template("query.html", var1 = ip)

```

```
# Route to handle voting requests among nodes
@app.route('/vote_request', methods=['POST'])
def vote_request():
    data = request.get_json()
    term = data['term']
    url = data['url']
    if url != selfNode.url:
        if selfNode.voted is False:
            if term > selfNode.term :
                response = {
                    'term': term,
                    'vote': 'yes'
                }
                selfNode.term = term
                selfNode.leader = url
                selfNode.voted = True
                selfNode.votes_received = 0
                selfNode.state = 'follower'
                return jsonify(response), 200
            else :
                response = {
                    'term': selfNode.term,
                    'vote': 'no'
                }
                return jsonify(response), 200
        if selfNode.voted is True:
            response = {
                    'term': selfNode.term,
                    'vote': 'no'
                }
            return jsonify(response), 200

# Route to handle heartbeats among nodes
@app.route('/receive_heartbeat', methods=['POST'])
def receive_heartbeat():
    height = request.form.get('height')
    hash = request.form.get('hash')
    term = request.form.get('term')
    url = request.form.get('url')
    height_check = (height == len(selfChain.chain))
    hash_check = (hash == selfChain.getMapHash)
    term_check = (term == selfNode.term)
    if url != selfNode.url:
        if url == selfNode.leader:
            if (height_check and hash_check and term_check):
                response = 'correct'
                selfNode.received = True
                return response, 200
            else:
                response = 'incorrect'
                selfNode.received = True
                selfNode.recover()
                return response, 200

# Route to retrieve the chain data
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {
            'chain': selfChain.chain,
            'NametoIpmap': selfChain.NametoIpmap,
            'NametoOwnermap': selfChain.NametoOwnermap,
        }
    return response, 200

# Route to receive new blocks from other nodes
@app.route('/receive_block', methods=['POST'])
def receive_block():
    url = request.form.get('url')
    index = request.form.get('index')
    previousHash = request.form.get('previousHash')
    timestamp = request.form.get('timestamp')
    mapHash = request.form.get('mapHash')
    hash = request.form.get('hash')
    newNametoIpmap = json.loads(request.form.get('newNametoIpmap'))
    newNametoOwnermap = json.loads(request.form.get('newNametoOwnermap'))
    if url != selfNode.url:
        newBlock = block(index, previousHash, timestamp, mapHash)
        if newBlock.hash == hash:
            selfChain.chain.append(newBlock)
            selfChain.NametoIpmap = newNametoIpmap
            selfChain.NametoOwnermap = newNametoOwnermap
            response = 'Block received'
        else:
            response = 'Block rejected'
        return response, 200
```

Listing 2: Application

```
import requests
import time
from datetime import datetime
import statistics
import matplotlib.pyplot as plt
import numpy as np

RequestNumber = 1000
API_URL = "http://localhost:5002/add"

latencies = []

for i in range(RequestNumber):
    data = {
        'domain': f'{i}',
        'ip': f'{i}',
        'owner':
            'cc7ef87a7c0851998ca7aa365b1355415c320472606d40259f2f32ebd606d6b7',
        'key': '167564605',
        'ran': '123'
    }

    start_time = time.time()
    response = requests.post(API_URL, data=data)
    end_time = time.time()

    latency = (end_time - start_time) * 1000
    latencies.append(latency)

mean_latency = statistics.mean(latencies)
```

```python
median_latency = statistics.median(latencies)
std_dev_latency = statistics.stdev(latencies)

print(f"\nMean latency: {mean_latency:.3f} ms")
print(f"Median latency: {median_latency:.3f} ms")
print(f"Standard deviation: {std_dev_latency:.3f} ms")

plt.hist(latencies, bins='auto', edgecolor='black')
plt.xlabel('Latency (ms)')
plt.ylabel('Frequency')
plt.title('Latency Distribution')
plt.show()

throughputs = [1000 / latency for latency in latencies]

plt.plot(throughputs)
plt.xlabel('Request Number')
plt.ylabel('Throughput (transactions/sec)')
plt.title('Throughput Over Time')
plt.show()

mean_throughput = statistics.mean(throughputs)
median_throughput = statistics.median(throughputs)
std_dev_throughput = statistics.stdev(throughputs)

print(f"\nMean throughput: {mean_throughput:.3f} requests per second")
print(f"Median throughput: {median_throughput:.3f} requests per second")
print(f"Standard deviation: {std_dev_throughput:.3f} requests per second")

# Define the number of nodes range
nodes = np.arange(3, 101, 1)

# Define fault tolerance functions for each consensus algorithm
def pow_pos_fault_tolerance(num_nodes):
    return 50

def paxos_raft_syncraft_fault_tolerance(num_nodes):
    return (num_nodes - 1) // 2 / num_nodes * 100

```

```
def pbft_fault_tolerance(num_nodes):
    return (num_nodes - 1) // 3 / num_nodes * 100

# Calculate fault tolerance values for each algorithm
pow_pos_values = [pow_pos_fault_tolerance(n) for n in nodes]
paxos_raft_syncraft_values = [paxos_raft_syncraft_fault_tolerance(n) for n
    in nodes]
pbft_values = [pbft_fault_tolerance(n) for n in nodes]

# Create a line chart
plt.plot(nodes, pow_pos_values, label='PoW/PoS')
plt.plot(nodes, paxos_raft_syncraft_values, label='Paxos/Raft/SyncRaft')
plt.plot(nodes, pbft_values, label='PBFT')

# Set chart labels and title
plt.xlabel('Number of Nodes')
plt.ylabel('Fault Tolerance (%)')
plt.title('Fault Tolerance of Consensus Algorithms as Nodes Increase')
plt.legend()

# Display the chart
plt.show()
```

Listing 3: Test

B University's Plagiarism Statement

The University's degrees and other academic awards are given in recognition of a student's personal achievement. All work submitted by students for assessment is accepted on the understanding that it is the student's own effort.

Plagiarism is defined as the submission or presentation of work, in any form, which is not one's own, without acknowledgement of the sources. Plagiarism includes inappropriate collaboration with others. Special cases of plagiarism can arise from a student using his or her own previous work (termed autoplagiarism or self-plagiarism). Autoplagiarism includes using work that has already been submitted for assessment at this University or for any other academic award.

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism. Work may be considered to be plagiarised if it consists of:

- a direct quotation;
- a close paraphrase;
- an unacknowledged summary of a source;
- direct copying or transcription.

With regard to essays, reports and dissertations, the rule is: if information or ideas are obtained from any source, that source must be acknowledged according to the appropriate convention in that discipline; and any direct quotation must be placed in quotation marks and the source cited immediately. Any failure to acknowledge adequately or to cite properly other sources in submitted work is plagiarism. Under examination conditions, material learnt by rote or close paraphrase will be expected to follow the usual rules of reference citation otherwise it will be considered as plagiarism. Departments should provide guidance on other appropriate use of references in examination conditions.

Plagiarism is considered to be an act of fraudulence and an offence against University discipline. Alleged plagiarism, at whatever stage of a student's studies, whether before or after graduation, will be investigated and dealt with appropriately by the University.

The University reserves the right to use plagiarism detection systems, which may be externally based, in the interests of improving academic standards when assessing student work.