EVALUATING BLOCKCHAIN NETWORKS THROUGH REAL-TIME SIMULATION

SAHAND SANIEE MONFARED

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF APPLIED SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING & COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

AUGUST 2023

Abstract

The massive scale and distributed nature of Internet-of-Things (IoT) presents challenges in realizing practical and effective security solutions. Blockchain-empowered platforms and technologies have been proposed to address aspects of this challenge. In order to realize a practical Blockchain deployment for IoT, there is a need for a testing and evaluation platform for the performance and security of Blockchain applications and systems. This research involves conducting a series of experiments through real-time simulation of the Algorand blockchain platform to assess its efficiency across different configurations. These experiments provide valuable insights into the behavior of the Algorand blockchain platform. The findings contribute to a better understanding of the platform's performance, and resource utilization under different conditions.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet of Things (IoT) is a recent technology that uses smart connected systems to create a global network of physical devices that exchange and communicate data with each other. Despite the widespread success and popularity of IoT technologies in diverse industries, they still need a faster and more secure approach for communication and data transfer among them. This is due to the hardware limitations of IoT devices, which make it challenging to implement secure protocols with limited processing power and low-energy usage. Blockchain technology has recently gained a lot of recognition as a tool for secure and transparent data storage and management. Blockchain is essentially a system for recording data that makes it much more difficult to tamper with. Blockchain uses a distributed networking system of machines that replicate and create a chain of data. This chain of data can be considered a ledger, with each of these becoming a block. This chain of data is turned into a block that is linked to the previous block creating a chain of blocks, hence the name blockchain.

Every block is further reinforced with blocks that come after it. This is due to the fact that the blocks that come after the previous one will contain information on the last block. Additionally, each block is propagated within the network, allowing each machine to view the entire chain and all of its data allowing for multiple verifications to happen. This ensures that the data is authentic and that the block's integrity is not compromised [1].

Blockchain technology offers several potential benefits that make it valuable in various industries. Such as:

- Decentralization: The decentralized nature of blockchain technology means that it is not controlled by a single entity or government, making it more democratic and resistant to censorship.
- Security: Blockchain technology uses cryptographic algorithms to create a secure and tamper-proof record of transactions. The decentralized nature of the technology ensures that data is not controlled by a single entity, making it more resistant to attacks and data breaches.
- Transparency: Blockchain technology allows for transparent and auditable transactions, making it easier to verify the authenticity of data and ensure that all parties involved are held accountable.
- Efficiency: Blockchain technology enables faster and more efficient transactions by eliminating the need for intermediaries, reducing transaction costs, and improving the speed of settlement.

- Traceability: Blockchain technology allows for the tracking of assets and transactions throughout their entire lifecycle, enabling greater visibility and accountability [2].

The distributed nature of blockchain allows it to be a decentralized transaction system that is transparent and secure. By being independent of a centralized network, blockchain technology has become an increasingly promising foundation for the future development of the internet and data processing. As the world becomes more interconnected and data-driven, the need for a trustworthy and efficient means of data storage and processing becomes increasingly important. In this context, the decentralization aspect of blockchain technology offers a valuable solution.

By leveraging blockchain's distributed and immutable structure, connected systems can achieve a higher level of trust and security. This is particularly relevant in the context of the Internet of Things (IoT), where data is generated and processed across a vast network of devices. Incorporating blockchain within IoT can significantly enhance its security and resilience to tampering. The decentralized nature of blockchain means that it is highly resistant to attacks and data breaches. Blockchains have been successfully used in IoT applications such as smart homes, smart cities, and smart stores. Blockchain can provide data privacy, prevent fraud, improve transparency, and ease record keeping [3].

In order to test the effectiveness and security of the blockchain being implemented in IoT, the most cost-effective way is to use simulation. Blockchain simulators have been developed which are capable of analyzing the different layers of blockchain, as well as applications in IoT. A simulator can be used to test a blockchain network beforehand and to make specific adjustments prior to implementation.

In this project, I worked on simulating and evaluating Algorand blockchain implementations. Algorand is a blockchain-based cryptocurrency platform that aims to be secure, scalable, and decentralized [4]. The Algorand platform supports smart contract functionality, and its consensus algorithm is based on proof-of-stake and a Byzantine Agreement protocol. Through simulating the Algorand, I empirically study the performance of this platform.

My investigation is aimed at answering the following questions:

- Workload speed: How does the Algorand blockchain perform under different transaction loads? Different scenarios like low traffic workload, high traffic, sudden increase, sudden decrease, and oscillating workload.

- Workload Size: What is the impact of varying the transaction size on the performance of the Algorand blockchain?

- Number of Nodes: How does the resource utilization of the Algorand blockchain vary under different numbers of nodes?

- Block Generation: What is the impact of varying the block generation policy on the performance of the Algorand blockchain?

This is the list of contributions that I made in my research.

- Development of a simulator for Algorand: I developed a simulator for the Algorand blockchain that allows me to configure the network and the workload. It also measures performance metrics and resource utilization.

- Analysis of resource utilization metrics: I used the simulation results to analyze resource utilization metrics such as CPU usage, memory usage, Network Input, network output, disk input, and disk output. This can help identify areas where the blockchain network may be over-utilizing resources, and suggest potential optimizations.

- Performance evaluation: By simulating the Algorand blockchain, I evaluated its performance in terms of transaction throughput, and confirmation times. This can provide insights into how the blockchain can be optimized for better performance.

- Adding Algorand to the BlockCompass simulator: BlockCompass simulator already supports several blockchain platforms like Ethereum and Hyperledger Fabric. I added the Algorand private network to the BlockCompass simulator. This helps us to use the workload generator and resource monitoring module of BlockCompass to run experiments on the Algorand network and test it in that condition.

The rest of this work is organized as follows: In chapter 2 the necessary information for understanding the ideas are provided. It also introduces blockchain simulators including BlockCompass. Chapter 3 describes the experimental setup, and steps that are needed to run the experiment. Next, in chapter 4 I show the experiment results and provide a comparison among them. Chapter 5 concludes the project.

Chapter 2

Background

Section 2.1 discusses the concept of blockchain and compares it to conventional methods of handling transactions. Next, in section 2.2 we discuss different types of consensus protocols and mention the consensus protocol that Algorand uses. Section 2.3 provides an introduction to the Algorand blockchain platform. The section outlines several key properties that are used to evaluate blockchain platforms and explains how Algorand performs in each of these areas.

2.1 Blockchain

A blockchain is a public ledger of transactional data, distributed across multiple computers ("nodes") in a network [5]. All of these nodes work together, using the same set of software and rules, to verify transactions to add to the finalized ledger. This approach is different than the traditional ledger of transactional data that may live in a single database on a few computers that only certain people have access to.

The "block" part of "blockchain" refers to a set of transactions that are proposed and verified by the other nodes and eventually added to the ledger. The "chain" part refers to the fact that each block of transactions also contains proof (a cryptographic hash) of what was in the previous block. This pattern of capturing the previous block's data in the current block continues all the way back to the start of the network (the genesis block) creating a publicly verifiable and tamperproof record of all transactions.

Practically, this means that changing even a single record, anywhere in the history of a blockchain, will be evident and rejected by the network nodes. This is different than a traditional ledger where a change in a database is entrusted to a limited group and can easily be manipulated either through malicious intent or simply an error.

In a blockchain, each node runs software that instructs them how to verify transactions and add new blocks to the chain [6]. These instructions are collectively referred to as the "consensus protocol". The nature of these instructions is one of the main distinguishing factors of different blockchains.

2.2 Consensus

A public Blockchain maintains a decentralized system on a worldwide scale. To ensure the integrity of these openly shared ledgers, an effective, ongoing, practicable, dependable, and secure consensus mechanism is required. Consensus refers to the process by which a decentralized network of nodes or participants reaches an agreement on the state of the system or a transaction. In the context of blockchain technology, consensus is a crucial mechanism that enables trust among participants and ensures the integrity and immutability of the data stored on the network. To ensure these criteria, it is important to maintain a consensus method, that has the Byzantine Fault Tolerance (BFT) characteristics. BFT is a system's ability to continue operating even if some of its nodes fail or act maliciously [7].

Blockchain consensus mechanisms can be divided into two categories of Proof-based consensus protocols and Voting-based consensus protocols.

2.2.1 Proof-based consensus protocols

Consensus protocols that rely on proof of effort or resource consumption are referred to as proof-based protocols. This type of consensus has the advantage of being able to scale to a large number of nodes, as they allow any node that can demonstrate a certain level of effort or resource consumption to participate in the consensus process. This means that there is no need for permission or trust between nodes to participate in the network. As a result, proof-based protocols can enable a large number of nodes to join the network and participate in the consensus process, which can enhance the security and decentralization of the network [8]. However, this technique can also result in a problem called forking, where two nodes propose blocks at the same time, causing the system to spend time determining which branch of the fork is the correct one. This delay in resolving the fork can result in longer completion times. Examples of proof-based consensus protocols include Proof of Work and Proof of Elapsed Time which we discuss next [9].

Proof of Work Consensus Proof of Work (PoW) is a consensus protocol that is widely used in various blockchain networks, including Bitcoin [10]. This consensus method is utilized to both, verify and validate transactions as well as mine cryptocurrency. The miner must solve a cryptographic puzzle after creating the transaction data. Solving this puzzle means that the miner has to find a hash for the last block that has a specific characteristic [11]. For instance, the first forty bits of the hash output should be zero. In order to achieve this, there is no other way rather than calculating the hash function with different parameters so many times until we find what we were looking for. This is the concept of the hashcash [12] which means that the miner had put so much time and computational power to achieve this output. The concept of hashcash could also be used in sending an email as proof that the sender is not an attacker or spammer since he had to perform a certain amount of computational work, such as performing a cryptographic hash function. If the miner finds the hash for the last block, they will earn the permission of adding a new block to the block-chain and earn a reward. Solving this cryptographic puzzle can have different difficulty levels. This difficulty determines how long it takes a miner to solve a block. The miner must additionally take the hash of the previous block as input, in addition to transaction data. Every block is connected to the next block in this manner, making a chain. Miners compete with their transaction data in order to solve the puzzle for a specific block. When a miner discovers a solution, it broadcasts it to the network, where it is validated by other miners. The block is added to the network after validation, and the miner who solved it is rewarded.

Proof of Elapsed Time PoET, an alternative to PoW, was released by Intel in 2016. PoET implements a competitive scheme based on a previously used random back-off mechanism in local area networks. In general, participating nodes request to the trusted party [13] a random waiting time, and the node with the shortest waiting time is chosen as the block leader [14]. In addition to publishing the new block, the block leader must show that it has the shortest waiting time and that it has not broadcast its block before the waiting time has expired. There are many other proof-based consensus protocols such as Proof of Burn, Proof of Capacity, and Proof of Activity that are not in my scope here.

2.2.2 Voting-based consensus protocols

We need network participants to share their results of validating new blocks or transactions before reaching a final choice on which node is permitted to add a new block to the chain. The voting-based algorithms have the advantage of providing low-latency finality [15]. Finality refers to the irreversible and permanent state of a transaction or a block once it has been added to the blockchain. It ensures that the recorded data cannot be reversed or modified, providing a high level of trust, security, and immutability. Consensus and finality are achieved when a majority of nodes validate a transaction or block. Since voting-based algorithms often need nodes to send messages to every other node on the network, the more nodes there are on the network, the longer it takes to establish consensus. As a result, scalability and speed are sacrificed. Proof Of Authority (PoA), Practical Byzantine Fault Tolerance (PBFT), and Raft are examples of this type of consensus.

Proof Of Stake Consensus Proof of Stake (PoS) is a consensus protocol used in blockchain technology to achieve consensus and validate transactions. In this protocol, validators are chosen based on the amount of cryptocurrency they hold, rather than the amount of computational power they contribute, as in the case of Proof of Work (PoW). Validators who put their cryptocurrency holdings as "stake" are selected to create and validate new blocks on the blockchain, with the probability of being chosen to be directly proportional to the amount of cryptocurrency they have "staked". PoS aims to solve some of the issues associated with PoW, such as high energy consumption and centralization of mining power, by enabling more efficient and environmentally friendly consensus [16].

Proof Of Authority Consensus The PoA protocol is an alternative to the Proof of Stake (PoS) protocol. Sealers, or trusted nodes, are chosen to validate transactions and blocks in PoA. New blocks can be created by authorized 'signers' only. The network has a high throughput, is extremely scalable, and has almost no processing fees due to the small number of validators required. In addition, unlike with PoS, a sealer does not need to stake any of its assets, only its reputation [17]. The main problem with PoS is that the staking of wealth leads to affluent members receiving the majority of the network's incentives. This is overcome by staking reputation (something that must be earned over time by engaging in a network).

Practical Byzantine Fault Tolerance PBFT is a consensus protocol based on replication between known parties that can tolerate a failure of up to one-third of the parties. It is an algorithm for resolving a Byzantine fault caused by the Byzantine Generals Problem's (GBP) failure to achieve consensus. In general, a primary node is elected to construct an ordered list of transactions that is broadcast to other validating nodes, who subsequently execute them. Validation nodes generate the hash code for the new block after transactions are executed, which is subsequently broadcast to their peers. The block is committed to the node's local copy of the Blockchain if two-thirds of the received hash codes are the same. With a tiny increase in waiting time, PBFT ensures network fault tolerance and supports thousands of operations per second [18].

Raft Raft is a consensus method that tolerates crashes. It is inspired by the famous consensus algorithm Paxos. Raft is predicated on the presumption that $[n/2 + 1]$ of the total nodes work normally at all times. Verifying nodes can play one of three roles to execute the Raft consensus algorithm: follower, candidate, or leader. RequestVote for voting a leader node and AppendEntries for forwarding requests to other nodes are the two major types of messages that these nodes send to each other. Essentially, there would be some sequentially numbered time periods during which a node in the leader role would keep its responsibilities until the end of the term, after which it would transition back to the follower role and a new election round begins [19].

2.3 Algorand

2.3.1 Features of Algorand

There are many different ways by which a blockchain network can be implemented, based on design decisions of their creators. Thus different networks exhibit different properties and aspects such as the native currency, the fee aspect, etc. We discuss this below and compare it with the network that is of interest to our project, Algorand.

The native currency Each blockchain has its own native currency that plays a critical role in incentivizing good network behavior. Algorand's native currency is called the Algo. The Algo also acts as a utility token. When someone is building an application, they need algos to pay transaction fees and to serve as minimum balance deposits if they want to store data on the blockchain. The cost of these fees and minimum balances is very low, fractions of a penny in most cases [20].

Fees Fees are calculated based on the size of the transaction and a user can choose to augment a fee to help prioritize acceptance into a block when network traffic is high and blocks are consistently full. There is no concept of gas fees in Algorand. The minimum fee for a transaction is only 1,000 microAlgos or .001 Algos [21].

Openness Earlier, we compared a blockchain ledger that is distributed, to a traditional ledger that is owned by a single entity. Technically, a blockchain ledger could be owned and operated by just a few entities, but this wouldn't be a very good blockchain since such a centralized set of nodes could easily manipulate the state of the blockchain. Algorand is completely open and permissionless [22]. Anyone, anywhere in the world, who owns Algos can participate in consensus.

Transparency Since all of the code for the core protocol for Algornad is open source, it is easy to check that anything that the Algorand team is telling us is true [23]. Anyone can review it and contribute to it.

Forking (or lack of) Forking is when a blockchain diverges into two separate paths. Sometimes this forking is intentional, like when a significant part of the community wants to change the fundamentals of the protocol. Other times this forking is accidental and occurs when two miners find a block at almost the same time. Eventually, one of the paths will be abandoned, which means that all transactions that occurred since that fork on the abandoned path (the orphaned chain) will be invalid. This has important implications for transaction finality, which will be mentioned below.

Forking is impossible in Algorand since Algorand is pure proof-of-stake and uses a voting mechanism to validate blocks [24]. In a worst-case scenario if the committee is taking longer to reach an agreement, the blockchain will slow down or temporarily stall.

Performance The speed at which blocks are produced, the number of transactions that can fit into a block, and when those transactions are considered final are important factors to consider when choosing a blockchain. For Algorand, performance is a key focus area.

Throughput We want to choose a blockchain that can scale and handle high throughput so that users do not experience long wait times when interacting with the application.

On Algorand, blocks are produced every 4.5 seconds and can hold up to 5,000 transactions, which results in a throughput of about 1,000 transactions per second (1000 TPS) [25].

Finality In proof-of-work blockchains, since forking is a possibility, transactions cannot be considered final until a certain amount of time passes and the likelihood of the transaction being on an orphaned chain is practically zero. This means that the actual throughput of this type of blockchain is caveated by a delay in finality.

As mentioned, Algorand uses pure proof of stake. So, Algorand does not have forking. therefore transactions are final as soon as they are confirmed in a block [26]. A throughput of 1,000 TPS for Algorand actually means 1,000 finalized transactions per second.

2.3.2 Algorand consensus protocol

Algorand utilizes its own Pure Proof of Stake (PPoS) consensus protocol. Algorand's consensus protocol works by selecting a block proposer and a set of voting committees at each block round, to propose a block and validate the proposal, respectively. The proposer and committees are randomly chosen from the pool of all token holders (which are the accounts that hold algos), and the likelihood of being chosen is proportional to the account's stake in the network (i.e. how many algos it has relative to the whole). The cryptographic algorithm that goes into this process is called "cryptographic sortition" to ensure that the vote is fair, no one can collude, and that the overall system is highly secure [27]. The protocol that Algorand uses to reach consensus among users is a new form of Byzantine Agreement (BA).

Gossip protocol Algorand network contains nodes and users. Nodes are responsible for participating in the consensus protocol, and propagating transactions and blocks throughout the network. Users interact with the Algorand network by sending transactions and accessing blockchain data. Algorand nodes communicate through a gossip protocol. Nodes use the gossip protocol to submit new transactions. Each node collects a block of pending transactions that they hear about, in case they are chosen to propose the next block, as shown in Figure 2.1 Algorand uses BA to reach a consensus on one of these pending blocks.

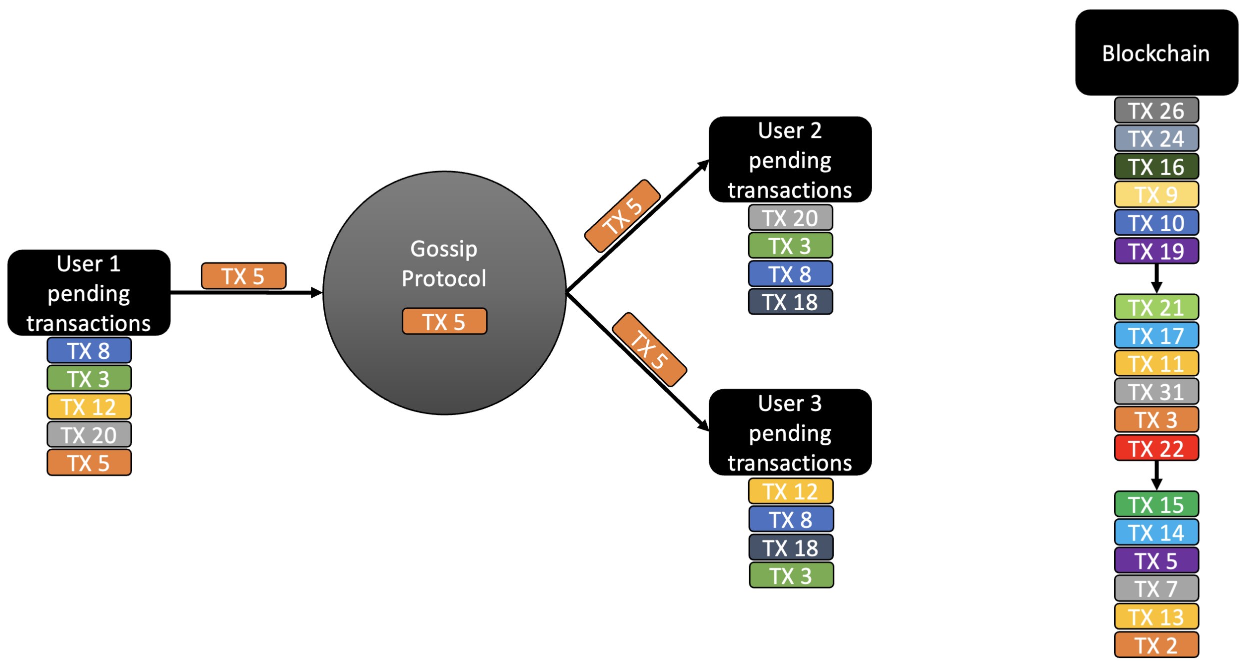

Figure 2.1: Transaction flow in Algorand

Transaction flow Each node in Algorand has a pair of public and private keys. For each transaction in the network, a user will sign that transaction using his own private key and will send it to another node's public key. These transactions will be added and generate a new block. Each block will be added after the last block and the chain of these blocks is called the blockchain. In Algorand the blockchain will grow in asynchronous rounds. In each round, some of the newly added transactions, which make a new block, will be added to the previous block. We also need to keep the pointer to the previous block in each block.

In algorand's network, a node will sign a message using his private key and send it to a small random set of peers. This method is called gossip network. The receiver node will check if the signature is valid using the sender's public key, then he will pass it around.

Final consensus and tentative consensus Algorand uses two kinds of consensus protocols to achieve Byzantine agreement: final consensus and tentative consensus.

Final consensus Final consensus is the protocol used to confirm the final state of the blockchain. It ensures that all honest nodes in the network agree on the current state of the blockchain, even in the presence of up to one-third of malicious nodes. Final consensus in Algorand is achieved through a process called BA or Byzantine agreement with an adaptive round. In the BA protocol, nodes first propose a block and then participate in a series of rounds of voting. In each round, nodes exchange messages with each other to determine the majority decision. Nodes can adaptively adjust the number of rounds based on the network's behavior, which helps to reduce the time needed to reach consensus. Once a block has been confirmed through final consensus, it is considered final and cannot be reverted. This ensures that the blockchain's history is immutable and cannot be tampered with.

Tentative consensus Tentative consensus is the protocol used to confirm the ordering of transactions within a block. It ensures that all honest nodes agree on the order in which transactions are included in a block, even in the presence of up to one-third of malicious nodes. Tentative consensus in Algorand is achieved through a process called Block Proposal, Soft Votes, and Certification. In the Block Proposal phase, nodes propose blocks and broadcast them to the network. In the Soft Votes phase, nodes cast "soft votes" for the block they believe should be included in the blockchain. In the Certification phase, nodes cast "hard votes" to confirm the block's validity and order transactions within it. Once a block has been certified through tentative consensus, it is considered tentative and can be reverted if necessary. However, the likelihood of this happening is low, as the final consensus process provides an additional layer of security. Overall, Algorand's use of final and tentative consensus protocols ensures that the blockchain is secure, reliable, and resistant to attacks from malicious nodes.

Verifiable Random Functions (VRFs) Verifiable Random Functions (VRFs) are cryptographic building blocks that allow the generation of a deterministic random output while providing proof of the correctness of the output. A VRF takes a secret key and an input and produces a verifiable output. The output appears random but is determined entirely by the key and the input.

Byzantine Agreement Algorand employs a fresh form of the Byzantine Agreement (BA) for its consensus protocol to achieve agreement among users about the upcoming transactions. To extend consensus to a large number of users, Algorand has developed a unique mechanism based on Verifiable Random Functions, which allows users to verify privately if they have been selected to participate in the Byzantine Agreement. They can also include proof of their selection in their network messages.

Block proposal In order to check if a user has been selected for proposing a block, they will run the cryptographic sortition in each round. sortition ensures that a small fraction of users are selected at random, weighed by their account balance, and provides each selected user with a priority, which can be compared between users, and a proof of the chosen user's priority. Users who are selected in the cryptographic sortition will send out their suggested block of transaction through the gossip protocol. These block proposals are prioritized based on the proposing user's priority.

Agreement using BA As mentioned, different nodes receive different blocks. BA can help us to reach consensus on a single block. Each user initializes BA with the highest-priority block that they received. BA executes in repeated steps, illustrated in Figure 2.2 Algorand uses BA to reach a consensus on one of these pending blocks.

Each step begins with sortition, where all users check whether they have been selected as committee members in that step. Committee members then send a message which includes their proof of selection. These steps repeat until, in some step of BA, enough users in the committee reach consensus. When a user notices that the previous step had ended, he will check if he has been selected for the next round or not in an asynchronous way. This means that the selected users can change each round so it gets harder for attackers to target some

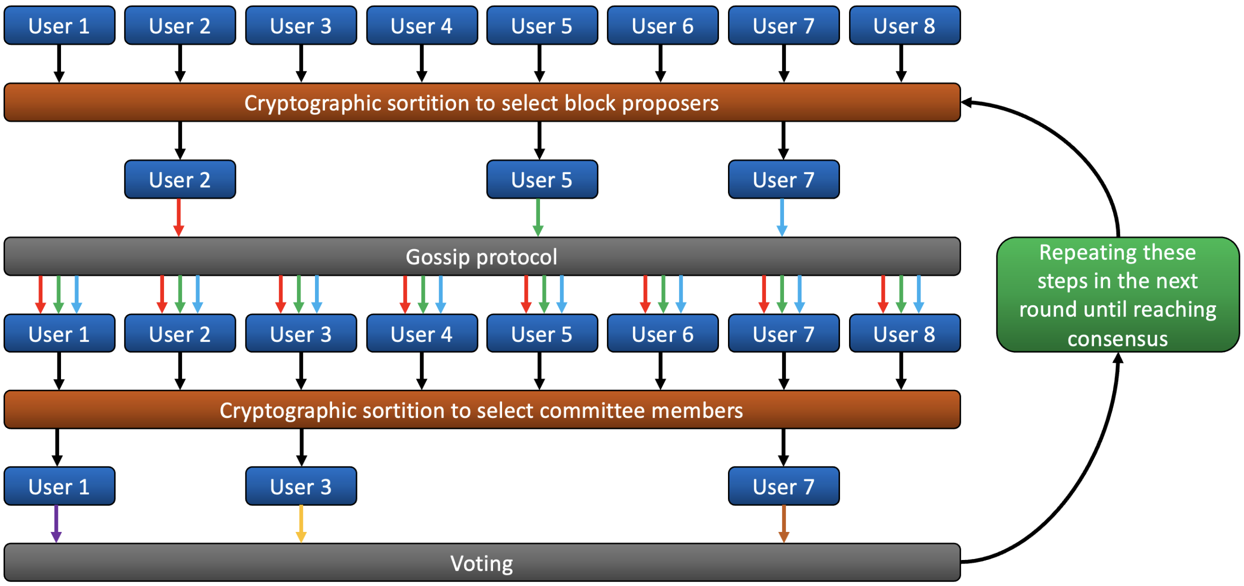

Figure 2.2: One round of BA

of the nodes.

Efficiency Reaching consensus depends if we have adversaries in the system or not. Also, the number of algos that the adversary has and his luck would change the number of rounds that are required to reach consensus. In case the highest priority block proposer is honest as well as the rest of the nodes, BA can reach consensus in four rounds. This would be the best-case scenario for BA. In the worst-case scenario, if the adversary is lucky, all of the honest users reach consensus on the next block in 13 steps.

Cryptographic Sortition In order to choose a random subset of users in the network for each round to participate in the block proposal we use a mechanism that lets us select users based on their weights. This mechanism is called Cryptographic Sortition. Here we have the weight for each user w_i and we have the total weights which is the summation of weights $W = \sum_i w_i$. the probability that user i is selected is proportional to w_i/W. The sortition algorithm uses Verifiable Random Functions (VRFs) that take three parameters as input. One of them is the random seed which enables the sortition algorithm to have randomness. Each user in the network has a public key/private key pair (pk_i, sk_i) that can be used as proof that the particular user has been selected for the block proposal or the agreement. On any input string x, $VRF_{sk}(x)$ returns two values: a hash and a proof. The hash has a constant length and can be uniquely determined by having the private key (sk) and the input x to the VRF. This is a one-way function, meaning that using the private key, we can verify that the hash is valid. But without having the private key, the hash is not distinguishable from a random string. Rather than the hash, there is another output from the VRF which we call the proof (π). The proof (π) enables anyone that knows pk to check that the hash indeed corresponds to x, without having to know sk.

Selection procedure Algorand implements cryptographic sortition using the VRFs. As shown in Algorithm 1, cryptographic sortition requires a role parameter that specifies the role that a user might have after he has been selected. for example, the user may be selected to propose a block in some round, or they may be selected to be a member of the committee at a certain step of BA. Algorand specifies a threshold τ that determines the expected number of users selected for that role. As mentioned earlier, cryptographic sortition selected users based on how many algos they own in the network. This is why cryptographic sortition is defending against Sybil attack. One subtle implication is that the users who have more algos may be chosen more than once by sortition. Sortition addresses this by returning the j parameter, which indicates how many times the user was chosen. Being chosen j times means that the user gets to participate as j different "sub-users." To select users in proportion to their money, we consider each unit of Algorand as a different "sub-user." If user i owns w_i (integral) units of Algorand, then simulated user (i, j) with $j \in 1, ..., w_i$ represents the jth unit of currency i owns, and is selected with probability $p = \frac{\tau}{W}$, where W is the total amount of currency units in Algorand.

Algorithm 1 cryptographic sortition algorithm

procedure Sortition $(sk$, seed, τ, role $, w, W)$:

$\langle$ hash, $\pi\rangle \leftarrow \mathrm{VRF}_{\mathrm{sk}}$ (seed $\|$ role)

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

while $\frac{\text{hash}}{2^{\text{hashlen}}} \notin \left[\sum_{k=0}^{j} B(k; w, p), \sum_{k=0}^{j+1} B(k; w, p)\right)$ do

  j_{++}

return $\langle$hash, $\pi, j\rangle$

As shown in Algorithm 1, a user performs cryptographic sortition by computing $\langle$ hash, $\pi\rangle \leftarrow$ VRF$_{sk}$(seed $\|$ role), where sk is the user's secret key. The pseudo-random hash determines how many sub-users are selected, as follows. The probability that exactly k out of the w (the user's weight) sub-users are selected follows the binomial distribution,

$$B(k;w,p)=\left(\begin{array}{c} w \\ \\ k \end{array}\right)p^k(1-p)^{w-k}$$

where

$$\sum_{k=0}^{w}B(k;w,p)=1$$

Since

$$B\left(k_1;n_1,p\right)+B\left(k_2;n_2,p\right)=B\left(k_1+k_2;n_1+n_2,p\right),$$

If a user splits his weight among different accounts but with the same owner, there will be no change in the number of sub-users that are going to be selected from his side. This is because the weight that he has, is just distributed among different accounts. But the overall is the same. This is how we cope with Sybil attack. To determine how many of a user's w sub-users are selected, the sortition algorithm divides the interval $[0,1)$ into consecutive intervals of the form

$$I^j=\left[\sum_{k=0}^{j}B(k;w,p),\sum_{k=0}^{j+1}B(k;w,p)\right) \text{ for } j\in\{0,1,\ldots,w\}$$

If $\frac{\text{hash}}{2^{\text{hashlen}}}$ falls in the interval I^j, then the user has exactly j selected sub-users. The number of selected sub-users is publicly verifiable from the VRF using the proof π.

Sortition provides two important properties. First, given a random seed, the VRF outputs a pseudo-random hash value, which is essentially uniformly distributed between 0 and $2^{\text{hashlen}}-1$. As a result, users are selected at random based on their weights. Second, an adversary that does not know sk_i cannot guess how many times user i is chosen, or if i was chosen at all (more precisely, the adversary cannot guess any better than just by randomly guessing based on the weights). The pseudocode for verifying a sortition proof, shown in Algorithm 2, follows the same structure to check if that user was selected (the weight of the user's public key is obtained from the ledger). The function returns the number of selected sub-users (or zero if the user was not selected at all).

Algorithm 2 Verifying sortition of a user with public key pk

procedure VerifySort (pk, hash, π, seed , τ, role, w, W):

if $\neg VerifyVRF_{pk}$ $(hash, \pi, seed\|role)$ $then\ return$ 0;

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

$while\ \frac{hash}{2^{hashlen}} \notin \left[\sum_{k=0}^{j} B(k;w,p), \sum_{k=0}^{j+1} B(k;w,p)\right)\ do$

$\quad j{+}{+}$

$return\ j$

Choosing the seed As shown in the Algorithm 1 Sortition requires a parameter named seed which will be chosen randomly and is publicly known. In each round of Algorand, we need a seed that is known by everyone in the network. The reason that we need this seed to be chosen randomly is that an adversary could select or suggest a seed that could lead to selecting some of the corrupted users. But if we select it randomly, we will not let the adversary benefit from it.

In each round of Algorand, a new seed is published. The seed published at Algorand's round r is determined using VRFs with the seed of the previous round $r-1$. More specifically, during the block proposal stage of round $r-1$, every user u selected for block proposal also computes a proposed seed for round r as $(seed_r, \pi) \leftarrow VRF_{sk_u}(seed_{r-1} \| r)$. Algorand requires that sk_u be chosen by u well in advance of the seed for that round being determined. This ensures that even if u is malicious, the resulting $seed_r$ is pseudo-random. This seed (and the corresponding VRF proof π) is included in every proposed block, so that once Algorand reaches an agreement on the block for round $r-1$, everyone knows $seed_r$ at the start of round r.

2.3.3 Attacks against Algorand and defense Tactics

Bitcoin In this section, I will illustrate a common method of defense tactics against blockchain attacks using the Bitcoin example. Additionally, I will explain how Algorand addresses and mitigates these types of attacks. Cryptographic currencies such as Bitcoin enable us to avoid trusted centralized authorities. However, in the cryptocurrencies that we already have, either we cannot completely make sure that our transaction has been confirmed on the blockchain and the payer cannot double-spend, or we have to wait a long time to get the confirmation. In Bitcoin's case, it is suggested that we wait for the next six blocks to be added to the blockchain so that we can trust that our block is confirmed and will not be erased in forking. This waiting for the next six blocks would happen approximately an hour. So, we either have to wait an hour and spend a lot of computational power to make sure that our transaction is confirmed, or we cannot be confident about our transaction confirmation.

Double-spending Double-spending in cryptocurrencies refers to the act of spending the same cryptocurrency twice. It occurs when someone tries to use the same cryptocurrency for multiple transactions simultaneously or in rapid succession. This can be done by creating a fake transaction or spending the same cryptocurrency in two different transactions before the first transaction gets recorded in the blockchain. Double-spending is a significant challenge in the world of cryptocurrencies, as it undermines the integrity of the blockchain network, and can lead to financial losses for users and businesses.

Cope with double-spending In order to cope with this problem, Bitcoin uses the concept of blockchain. In blockchain we have the record for each transaction from the beginning until now. So when a user spends his bitcoin, we can find the record of him spending bitcoin on the blockchain. That is why he cannot spend it later on. Although there is a possibility that the user tries to double-spend the Bitcoin right after he spend it somewhere else. That is why it is recommended to wait at least for six blocks, about an hour, before accepting a Bitcoin transfer. Otherwise the buyer may have spend the same Bitcoin in a different fork shortly after he has spent it the first time.

Double-spending is a possible threat in Bitcoin due to the way the consensus protocol, Proof of Work (PoW), operates. Miners participating in the PoW protocol aim to solve a cryptographic puzzle by finding a nonce that, when concatenated with the hash of the previous block, results in an output that satisfies the Bitcoin condition, which requires the first forty bits of the output to be zero. However, the hash function used in Bitcoin works in a way that changing one bit in the input will change every character in the output. So attackers cannot gain any data from their previous experiments. Solving the puzzle requires linear search of a vast space of candidate solutions, and, is as such, time consuming but not impossible.

BA However, Algorand uses a Byzantine agreement protocol called BA that scales to many users, which allows Algorand to reach consensus on a new block with low latency and without the possibility of forks. A key technique that makes BA suitable for Algorand is the use of verifiable random functions (VRFs) to randomly select users in a private and non-interactive way.

Denial of Service (DoS) A Denial of Service (DoS) attack is a type of cyber attack where someone tries to make a website or service unavailable by overwhelming it with traffic or other types of requests. An example could be when a large number of people try to get through a door at the same time and no one can get in or out.

The goal of a DoS attack is to make it impossible for legitimate users to access the website or service by using up all of its resources like memory, processing power, or bandwidth. This can make the website slow or unresponsive, or even cause it to crash entirely.

There are a few different types of DoS attacks, including flood attacks where the attacker floods the website or service with a huge amount of traffic or requests, and application-layer attacks where they target specific vulnerabilities in the website or service to overload it.

To protect against DoS attacks, organizations can implement a range of strategies, such as increasing the capacity of their servers and network resources, filtering out malicious traffic with firewalls and intrusion detection systems, distributing traffic across multiple servers to prevent one from becoming overwhelmed, and using cloud-based services to absorb sudden spikes in traffic.

Sybil attack Sybil attack is a type of security attack where someone creates multiple fake identities or accounts to gain control or influence over a network or system. This can be a big problem because the attacker can use their fake identities to manipulate the network or system and achieve their own goals.

For example, in case that we are part of an online community that makes decisions based on a voting system. If someone launches a Sybil attack and creates multiple fake accounts, they could use those accounts to vote multiple times and change the outcome of the vote in their favor. This can lead to a lack of trust in the community and cause problems down the line.

There are different ways to prevent Sybil attacks, such as requiring users to perform some type of computational work to participate in the network. One of them is proof-of-X protocols such as proof-of-work. Algorand is not using the PoW. The second way is by relying on reputation scores to limit the influence of users with low reputations. The other way to cope with Sybil attack is to have an organization or person which every node in the network trusts and then that node can control the creation of multiple fake accounts. This could be helpful in centralized applications but one of the main points of using blockchain is to avoid having a centralized application and a trusted third party. So this cannot help us in the blockchain.

Addressing the challenges Algorand addresses these challenges using several techniques, as follows.

Weighted users To prevent Sybil attacks, Algorand assigns a weight to each user. BA is designed to guarantee consensus as long as a weighted fraction (a constant greater than 2/3) of the users are honest. In Algorand, users will be weigh based on the amount of algos in their account. Thus, as long as more than some fraction (over 2/3) of the money is owned by honest users, Algorand can avoid forks and double-spending.

Consensus by committee BA achieves scalability by choosing a committee to run each step of its protocol. A committee is a small set of representatives randomly selected from the total set of users. All other users observe the protocol messages, which allows them to learn the block that has been agreed on. BA chooses committee members randomly among all users based on the users' weights. This allows us to ensure that a sufficient fraction of committee members are honest. However, relying on a committee creates the possibility of targeted attacks against the chosen committee members.

Cryptographic sortition As mentioned earlier, BA selects committee members in a private and non-interactive way. This method prevents an adversary from targeting committee members. This means that every user in the system can independently determine if they are chosen to be on the committee, by computing a function (a VRF) of their private key and public information from the blockchain. If the function indicates that the user is chosen, it returns a short string that proves this user's committee membership to other users, which the user can include in his network messages. Since membership selection is non-interactive, an adversary does not know which user to target until that user starts participating in BA.

Participant replacement Finally, an adversary may target a committee member once that member sends a message in BA. BA mitigates this attack by requiring committee members to speak just once. Thus, once a committee member sends his message (exposing his identity to an adversary), the committee member becomes irrelevant to BA. BA achieves this property by avoiding any private state (except for the user's private key), which makes all users equally capable of participating, and by electing new committee members for each step of the Byzantine agreement protocol.

Byzantine agreement lets us to eliminates the possibility of forking while using Algorand. As a result, transactions are confirmed on the order of a minute. To make the Byzantine agreement robust to Sybil attacks, Algorand associates weights with users according to the money they hold.

Proof of Stake (PoS) In many of the blockchain platforms that use Proof of Stake for their consensus, a malicious node that wants to assemble a new block can create a fork in the network. In case this node is caught, he will lose his money on the network. People can find out about this since two versions of the new block are signed with his key. Users in Algorand have been assigned weights depending on how much money they have in the blockchain. The weights in Algorand, however, are only to ensure that the attacker cannot amplify his power by using pseudonyms; as long as the attacker controls less than 1/3 of the monetary value, Algorand can guarantee that the probability for forks is negligible.

Because proof-of-stake eliminates the computational burden of proof-of-work, transaction confirmation times can be shortened. However, It is difficult to implement proof-of-stake in reality. A malicious leader might announce one block and then show another block to isolated users. Attackers may also distribute their credits among numerous "users," some of whom may be chosen as leaders, in order to minimize the penalty when a dishonest leader is discovered. In order to reduce forks, certain proof-of-stake cryptocurrencies ask for a master key to regularly sign the proper branch of the ledger. This requires the trust of the users in the node that holds the master key and is in opposition to decentralization. This idea may also lead to unintentional forks. One of the benefits of not having forking in Algorand is that even if the leader is malicious, he still will not be able to fork and double-spend.

2.4 Blockchain Simulators

2.4.1 Overview

We now switch our focus to systems developed for simulating and analyzing blockchain protocols. Blockchain simulators let us test and optimize blockchain applications in a controlled environment. By simulating different scenarios, we can identify potential bottlenecks in the network and optimize our applications for performance. By using simulators, we can test our applications without having to deploy them to a real blockchain network. For instance, after writing a smart contract we need to test it to see how it would work under certain circumstances [28]. We can run our tests by sending different types and rates of transactions. This helps us to make sure that our application is going to work as expected in real life.

Another benefit of blockchain simulators is that they can help us identify security risks in our blockchain applications. By simulating various attack scenarios, we can identify potential vulnerabilities and optimize our applications to be more secure. We can also simulate different types of transactions, and test different consensus algorithms. This could help to study the behavior of blockchain networks under different scenarios [29].

Simulators can help us study the behavior of large-scale blockchain networks that would be difficult or costly to replicate in real life. Large-scale blockchain networks are difficult to replicate exactly as they are in real life due to the sheer size and complexity of the network. For example, in the case of Bitcoin, there are millions of users and thousands of nodes spread across the globe. Simulating such a massive network requires significant computational resources and can be very time-consuming. Furthermore, there are many unpredictable factors that can affect the network's performance, such as node failures, network congestion, and malicious attacks. Therefore, it is challenging to create a perfect replica of the real network, and simulation tools are needed to approximate the real-world behavior of these networks.

Using simulation, I want to answer the following questions:

- How does varying the speed of transaction loads (the traffic rate for sending transactions to the blockchain) impact the Algorand blockchain's resource utilization and performance metrics?

- How does varying the size of transactions affect performance metrics and resource utilization?

- How does varying the count of transactions that will be sent to the network impact performance metrics and resource usage?

- How does varying the number of nodes in the blockchain impact performance metrics and resource usage?

- How does varying the size of the block impact performance metrics and resource usage?

Simulators can be compared based on their simulation approach, such as discrete-event or real-time. Another way to compare them is by the networks they support, measurements they provide, and their fidelity.

Simulation approaches There are two main approaches to simulating a blockchain platform: discrete-event simulation (DES) and real-time simulation (RTS).

Discrete-event simulation (DES) is a type of simulation that models systems that change their state at specific points in time. It is used to simulate systems where events occur in a discrete manner, such as the arrival of a customer in a queue or the completion of a manufacturing process. DES simulators use mathematical models to describe various parameters of the system and simulate the events that occur in the system over time. The simulation progresses by advancing the time to the next event, and the state of the system is updated accordingly [30].

Real-time simulation (RTS) on the other hand is a type of simulation that is used to simulate systems that operate in real-time. RTS simulators use models that incorporate both the behavior of the system and the environment in which it operates. This type of simulation is used in fields such as robotics, aviation, and military simulations [31].

One key difference between DES and RTS is the timing of events. In DES, events occur at specific points in time, whereas in RTS, events occur in response to external stimuli, such as a sensor reading or a user input. Both DES and RTS have their strengths and weaknesses, and the choice of which simulator to use depends on the specific application and the requirements of the system being modeled.

Fidelity Fidelity in blockchain simulation refers to the degree to which a simulation accurately represents the behavior and performance of a real-world blockchain network. Fidelity is important because it allows researchers and developers to test and evaluate the performance and security of blockchain protocols and applications under various conditions without risking real-world assets. The fidelity of a blockchain simulation depends on various factors such as the accuracy of the underlying models, the level of detail in the simulation, and the complexity of the network being simulated. For example, a simulation that only considers the basic consensus mechanism of a blockchain network may have low fidelity, while a simulation that considers additional factors such as network latency and transaction fees may have higher fidelity. There are different blockchain simulators with varying degrees of fidelity. Some simulators use simple models to simulate the behavior of a blockchain network, while others use more complex models that can capture the details of a real-world network [32]. High-fidelity simulators are often used in research to analyze the performance of new blockchain protocols or to test the security of existing protocols against various attack scenarios. Fidelity can affect the accuracy of the simulation results [33].

2.4.2 Blockchain Simulator Examples

BlockSim BlockSim is a blockchain simulator that allows developers and researchers to simulate and analyze blockchain networks in a controlled environment. It supports various consensus algorithms and provides a way to analyze the performance and security of blockchain networks under different scenarios. BlockSim includes an interface that allows developers to easily set up and run simulations. BlockSim is a good choice for developers who want a general-purpose simulator that supports various consensus algorithms and provides a user-friendly interface for setting up and running simulations [34].

VIBES VIBES (Virtual Blockchain Simulation) is a blockchain simulator developed by IBM. It provides a way to simulate different types of blockchain networks, including private, public, and hybrid networks. VIBES includes a user-friendly interface for setting up and running simulations, as well as tools for monitoring network performance and analyzing data. VIBES is a good choice for developers who want a tool specifically designed for simulating private, public, or hybrid blockchain networks [35].

SimBlock SimBlock is a blockchain simulator developed at the University of Illinois. It is a discrete-event simulator that models the behavior of a blockchain network based on the actions of individual nodes. SimBlock allows developers to simulate various types of blockchain networks, including permissionless and permissioned networks. It can simulate various network conditions and attacks to study the behavior of the network. SimBlock is a good choice for researchers who want a more detailed simulation of blockchain networks, as it models the behavior of individual nodes and can simulate various types of network conditions and attacks [36].

BLOCKBENCH BLOCKBENCH is a benchmark suite for private and consortium blockchains. It provides a set of benchmarks to evaluate the performance and scalability of different blockchain platforms, including Ethereum, Hyperledger Fabric, and Corda. BLOCKBENCH includes a set of micro-benchmarks and macro-benchmarks that simulate various types of workloads and transactions. BLOCKBENCH is a good choice for developers who want a set of benchmarks to evaluate the performance and scalability of different blockchain platforms [37].

Bitcoin Simulator Bitcoin Simulator is a simulator that models the behavior of the Bitcoin network. It allows developers to simulate various network conditions and study the behavior of the network under different scenarios. Bitcoin Simulator provides a way to analyze the performance and security of the Bitcoin network and identify potential improvements. Bitcoin Simulator is a good choice for developers who are specifically interested in simulating the behavior of the Bitcoin network and identifying potential improvements. It is a relatively simple simulator that is focused on the Bitcoin network, rather than supporting a wide range of consensus algorithms and network types [38].

In table 2.1, I mentioned the simulation approach, fidelity, and a list of the blockchain platforms that each simulator is capable of simulating. Additionally, I provide information on the parameters that can be measured by each simulator.

Simulator	**Discrete-Event/ Real-Time**	**Networks Supported**	**Measurements Supported**	**Fidelity**
BlockSim	Discrete-Event	Bitcoin-like networks	Block propagation, mining power, network latency	High
VIBES	Discrete-Event	Permissioned networks, Bitcoin-like networks	Network traffic, network delay, consensus protocols	High
SimBlock	Real-Time	Bitcoin-like networks, Ethereum-like networks	Network latency, block propagation, mining power, transaction throughput	Low
BLOCKBENCH	Discrete-Event	Permissioned networks, Ethereum-like networks	Transaction throughput, confirmation latency, resource utilization, scalability	High
Bitcoin Simulator	Real-Time	Bitcoin-like networks	Mining power, transaction propagation, block size, network latency	Low

Table 2.1: Comparison of Blockchain Simulators

There were some terms in the table that I elaborate on them here. **Bitcoin-like networks** refer to blockchain networks that use a Proof of Work (PoW) consensus mechanism, similar to the one used in the Bitcoin blockchain. These networks typically require miners to perform computationally-intensive calculations in order to validate transactions and create new blocks. **Permissioned networks** A permissioned network is a type of blockchain network where access to the network and its functionalities are restricted to authorized participants only. In other words, permissioned networks require participants to have permission or authorization to join and participate in the network, usually granted by a central authority or administrator. Permissioned networks are also sometimes called private blockchain networks. Unlike public blockchain networks like Bitcoin and Ethereum, which are decentralized and open to anyone, permissioned networks are centralized and require permission to join [39]. **Ethereum-like networks** Ethereum-like networks refer to blockchain networks that use a smart contract platform, similar to the Ethereum blockchain. These networks typically use a different consensus mechanism than Bitcoin-like networks, such as Proof of Stake (PoS) or Proof of Authority (PoA), and allow developers to create decentralized applications (dApps) on top of the blockchain using smart contracts. **Blockchain performance** Blockchain performance refers to the efficiency and effectiveness of a blockchain network in terms of its ability to process transactions, validate blocks, and maintain the overall health of the network. **Mining power** Mining power refers to the computational power or hash rate that is required to mine new blocks in a Proof of Work (PoW) blockchain network. The more hash rate a miner has, the greater their chances of solving the puzzle and mining a new block, and the greater their share of the block reward. Therefore, mining power is a crucial factor in the security and stability of a PoW blockchain network, as a higher mining power makes it more difficult for any single entity to gain control of the network.

2.5 BlockCompass

BlockCompass is a Longitudinal Benchmark Tool to Evaluate the Performance of Different Blockchain Platforms [40].

Blockchain is still a widely misunderstood and misconstrued concept. There are several public and private Blockchain implementations, each with its own set of benefits and drawbacks. As a result, we need to empirically, systematically, and critically examine each Blockchain implementation in order to assist industries and practitioners in selecting the best platform based on their needs. BlockCompass is a Benchmark Tool to Evaluate the Performance of Different Blockchain Platforms. BlockCompass is a real-time, and high fidelity simulator that supports simulation of Hyperledger Fabric, Hyperledger Sawtooth, and Ethereum. BlockCompass supports multiple performance indicators including resource consumption (CPU, Memory, Network Input, and Network Output), and performance metrics (latency, transmission rate, throughput, and error rate). It also provides a real-time chart and the possibility to export a detailed report. BlockCompass comes with a custom workload generator that simulates concurrent users. This workload generator is used to simulate load for performing performance testing [40].

Overview The goal of creating BlockCompass was to have a flexible and extensible blockchain platform benchmarking tool. BlockCompass provides a fair means of comparing different platforms and enables a deeper understanding of different system design choices, while also assisting developers in identifying bottlenecks and improving their platforms. It assesses overall and component-level performance. BlockCompass creates complete instances of blockchain nodes and adds an instrumentation layer whereby measurements of the resource consumption (CPU, Memory, Network Input, and Network Output), and performance metrics (latency, transmission rate, throughput, and error rate) can be taken. Since our simulator is based on Blockcompass we provide a more detailed description of the tool in the next chapter.

2.6 Docker

Docker is a platform that enables us to create, deploy, and run applications in a container. A container is an executable package of software that includes everything needed to run the application. These are the reasons I use Docker to simulate the Algorand blockchain.

- Isolation: Docker containers provide a level of isolation between the host system and the container. This means that any changes made to the container are contained within the container and do not affect the host system. This is particularly useful when testing and experimenting with different configurations or setups.

- Portability: Docker containers are portable and can be run on any system that supports Docker. This means we can build our Algorand sandbox environment once and then run it on any system, regardless of the operating system or configuration.

- Efficiency: Docker containers are lightweight and require minimal resources to run. This means that we can run multiple containers on a single system without impacting performance or resource utilization.

- Consistency: Docker containers ensure that the environment in which our Algorand sandbox is running is consistent across different systems. This can help to avoid issues that may arise due to differences in system configurations or dependencies [41].

Here I present some docker basic notions.

Definition Docker allows users to package and run an application inside of a container, which is a loosely isolated environment. The user can run multiple containers on a single host because of the isolation and security which simplify the delivery of the distributed application [42].

Docker image A Docker image is a read-only template that gives instructions for building a Docker container. It allows users to package programs and pre-configured server settings in an easy-to-use manner. A Docker image is a collection of files that contain all of the necessary components, such as installations, application code, and dependencies, to set up a fully functional container environment [43]. One of two ways can be used to construct a Docker image:

- Interactive: By starting a container from a pre-existing Docker image, manually altering the container environment in real-time, and storing the resulting state as a new image.

- Dockerfile: By producing a Dockerfile, a plain-text file that contains the instructions for creating a new Docker image.

Docker Compose In the Docker paradigm, each container contains a single piece of software. As a result, when developing a multi-component application with Docker, it is unavoidable to incorporate multiple software components (containers) into a complex workflow. To address this issue, containers must be instantiated and properly integrated. Docker compose, as known, can be used to reduce this difficulty by offering a uniform setup method that deploys many containers using a YAML configuration file. [44]

Chapter 3

Algorand Simulation

3.1 Simulating Algorand with Blockcompass

3.1.1 Architecture

So far, we discussed the Algorand blockchain platform and its features, and we mentioned the BlockCompass simulator which can be used to evaluate different blockchain platforms. We propose to utilize BlockCompass in order to simulate Algorand aimed at analyzing its performance compared to other popular networks. To achieve that we perform the following main research activities:

- Development: Develop a Blockcompass plug-in for allowing the simulation of Algorand.
- Performance test: Perform performance test and resource utilization analysis of the Algorand implementation.

Overview We detail the architecture for simulating Algorand with BlockCompass in the following paragraphs and show the framework components. As you can see in Figure 3.1, the first component is the workload generator which is responsible for generating a key-value pair and sending it to the next component which is the client adapter. The client adapter receives the data from the workload and adjusts it to the format that is readable for the targeted blockchain network. The target network is the blockchain platform that we aim to simulate and analyze. It consists of some nodes which represent the users in the network. It also consists of the smart contract that has been written and deployed on the blockchain. The next component is resource monitoring which is working in parallel with the workload. It is responsible for constantly monitoring the resource utilization of the network during the experiment. The front-end component is for showing the analyzed result that we got. The mongoDB component saves all the data from the workload and resource consumption so that we can use them for analyzing the results.

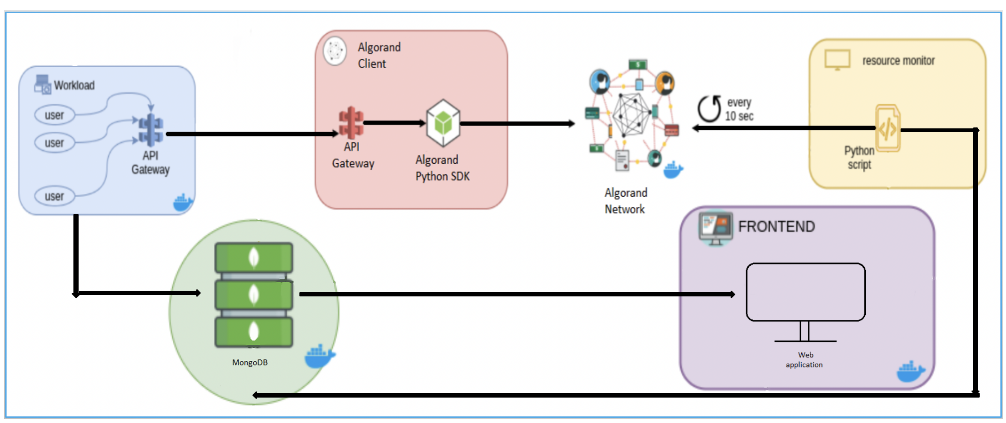

Figure 3.1: Showing the architecture of integration of Algorand and BlockCompass

Workload First, we have the workload component that simulates concurrent users and submits data packets of varied sizes to the Blockchain network of choice. The size of the packets can be determined in the configuration file and the data itself is generated randomly. The workload generator follows a closed-model approach, which means that users are completely independent of one another. So, different users can send packets in parallel but each user only submits the next request after the Blockchain client has returned a successful/Reject response code. The Workload keeps track of the sending and receiving time stamps so that we can calculate the latency, emit rate, throughput, and error rate. The collected data will be inserted into a MongoDB collection. The data that is generated in the workload generator will be sent to the server through an HTTP request. Later, the client adapter of each blockchain platform can read the data from the server and send it to the blockchain.

I conducted the experiments using two distinct methodologies. In the first approach, the workload generator was embedded within the same Docker environment as the blockchain network. Conversely, in the second scenario, the workload generator operated external to the Docker environment to establish communication with the network. My objective was to assess the system's reliability when the workload generator was situated separately from the blockchain network. In both instances, the network successfully received 100% of the transmitted packets. It is worth noting that placing the workload generator in Docker, lets us use the resource monitoring module to measure the consumption of computation resources used for the workload generator as well.

The process of calculating response time involved several steps. Initially, data generation was initiated by the workload generator. Subsequently, this generated data was submitted to the network. Finally, the response time was determined by evaluating the time taken for the network to process the transaction and produce a response. Within the workload generator, data could be generated at varying rates, with different intervals between the generation of successive data. This interval had an impact on the response time, which is not caused by network-related factors. Following the execution of each experiment, I recorded both the count of transactions sent and the time intervals between their submissions. Multiplying these parameters allowed me to isolate the delay introduced solely by the workload generator. In situations where the objective is to quantify network delay, it is possible to deduce the contribution of the workload generator's delay from the overall delay observed.

RESTful API After the workload generator generates the data, it will send it to the local host on port 3000. As a result, I need a server to run on port 3000 to receive the data from the workload generator and pass it to the client adapter. I have implemented this server using Flask and I wrote a RESTful API to do this. Flask is a lightweight web framework for Python that can be used to build RESTful APIs. It provides a simple and flexible way to create web applications and services. Flask allows developers to easily map URL endpoints to Python functions, which can then be used to handle incoming HTTP requests and generate responses.

RESTful API, or Representational State Transfer API, is a type of web service that uses HTTP requests to access and manipulate data [45]. RESTful APIs are based on a set of architectural principles and constraints that define how the API operates. Using RESTful

API we can have different HTTP methods such as GET and POST. We use GET to send the request for reading some information and POST to send the request for writing some information. In our case, I needed a POST method to send the key-value pair in my request and then after the request has been processed, I will send back the response. The response could be success, 200, or fail, 400, depending on if sending the data to the blockchain has been successful or not. I am providing the Python code for the RESTful API in appendix 6.1.

Client Adapter Each Blockchain platform provides its own unique methods to allow applications to interact with the network. In order to support multiple blockchain platforms, the workload will send requests to the client adapter that will be specified in the configuration file, and the client adapter will be responsible for sending the request to the Blockchain Network.

We use the Python SDK to interact with the blockchain network by connecting to one of the nodes. Through this node, we can send a transaction to the blockchain and wait for the nodes in the blockchain network to achieve consensus.

Blockchain Network The next component is the Algorand blockchain private network. We need this blockchain network to submit new transactions, and read blockchain data. There are different options to do this. In this work, we will use the Algorand Sandbox which is the most-used option. The sandbox allows developers to create local, private networks. Moreover, we can quickly remove a network, reset its state, or spin up a new network.

After running a private network for Algorand, we need to write a smart contract in PyTeal language and then deploy it to the blockchain. A smart contract is a self-executing contract with the terms of the agreement between buyer and seller being directly written into lines of code. The code and the agreements contained therein exist across a distributed, decentralized blockchain network. Then we need to add some features to the smart contract to be able to change a variable so that we can evaluate its performance of it later on. This can be done by sending a request to the smart contract to increment or decrement a variable that has been defined.

Resource Monitor The Resource Monitor component is responsible for monitoring the CPU, Memory, Network Input, and output consumption. Docker lets us access real-time resource monitoring data. Since we have the workload generator and we created our private blockchain network inside Docker, the logs that Docker shows us are regarding the resource utilization for those containers inside Docker. This helps us to monitor the resource that is being utilized only for our experiment. No other running processes on our computer.

Database The data collected by the Workload and Resource Monitor components are saved in different collections within the database. We use a database since it enables efficient retrieval and management of information. It also allows applications to access real-time data updates. We can use this feature to create a real-time chart stream.

Frontend Client The Frontend Client is an Angular 10 Web Application that displays real-time charts of all the collected metrics. The charts show the average of each metric as well as statistics per node. Aside from the charts, the Frontend displays each metric's average, standard deviation, variance, minimum, and maximum values. After getting the results in the front end, we can export the charts or save the result as a PDF file to use this information later on when we want to analyze Algorand or compare it with other blockchain platforms.

3.1.2 Evaluation metrics

- Emit rate: The workload generator runs concurrent users. Each user sends the next request only when he receives a response to the previous one. The emit rate indicator represents the number of requests sent per second.

- Throughput: The rate at which valid transactions are received by the blockchain in a second. When a transaction is initiated by a user on the blockchain, it is broadcasted to all the nodes in the network. According to the definition given by BlockCompass, the blockchain is said to have received a transaction as soon as the first node on the network receives it. This rate is expressed as transactions per second (TPS) at a network size. The total number of invalid transactions should be subtracted from the total transactions to yield the total received transactions.

- Error rate: Measure all transactions that have not been received by another node in the blockchain per second.

- Latency: Transaction Latency is a network-wide view of the amount of time taken for a transaction's effect to be received by the network. The measurement includes the time from sending a transaction from the workload generator to the point that the first node on the blockchain receives it. When a blockchain node receives a transaction, it first generates a transaction hash and then returns a response. Generating a transaction hash, however, does not imply that the transaction is added to a block.

- Resource utilization: CPU consumption, memory consumption, as well as network usage.

Running Ethereum-clique experiment as an example In order to get familiar with the BlockCompass tool and get an idea of what should we expect from implementing the Algorand private network, I ran the Ethereum blockchain on the BlockCompass as a sample project before implementing and running Algorand. I configured this experiment with five nodes in the blockchain. After running the experiment I can access the real-time charts containing resource utilization and performance metrics. In figure 3.2 and 3.3 we are providing an example of the result of the resource utilization and performance metrics.

Summary

#	CPU utilization (%)	Memory (%)	Network Input (MB)	Network Output (MB)
Minimum	0.564	0.99	0.464	0.43
Maximum	7.59	4.262	37.22	35.24
Mean	1.9	2.675	1,395,629.524	1,323,988.571
Variance	2.434	1.804	1,462,813,395,584.762	1,310,712,059,062.857
Std. Dev	1.56	1.343	1,209,468.228	1,144,863.336

Figure 3.2: Resource Utilization

So far we have the tool for evaluation of the blockchain platforms which is called Block-

Summary

#	Users #	Emit rate (transaction/sec)	Throughput (transaction/sec)	Latency (ms)	Error Rate
Minimum	10	16	16	0	0
Maximum	80	172	172	1,433.397	0
Mean	42.273	85	85	19.056	0
Variance	789.827	3,348.286	3,348.286	332.992	0
Std. Dev	28.104	57.864	57.864	588.401	0

Figure 3.3: Performance Metrics

Compass, and we have the Algorand private network and the smart contract that has been deployed on it. Now It is time to add an interface that can be implemented with Python SDK and can work as the client adapter to help connecting these two parts together. After having all of the required components it is time to adjust the monitor script and as a next step, asses overall and component-level performance in terms of throughput, latency, and resource consumption.

3.2 Simulating Algorand with YorkBlockSim

During this experiment, my primary objective was to configure the workload generator with different rates, sizes, and counts. Furthermore, I aimed to develop a tool that would process the acquired results and generate diagrams to facilitate analysis. To accomplish these goals, I created a custom blockchain simulator named YorkBlockSim [46], designed specifically for conducting experiments with various configurations and providing useful tools for result analysis. The subsequent sections outline the steps taken to create and set up

YorkBlockSim for simulating the Algorand network. The source code for the YorkBlockSim and the instruction on how we can run the experiment using this simulator can be found on YorkBlockSim's GitHub [46].

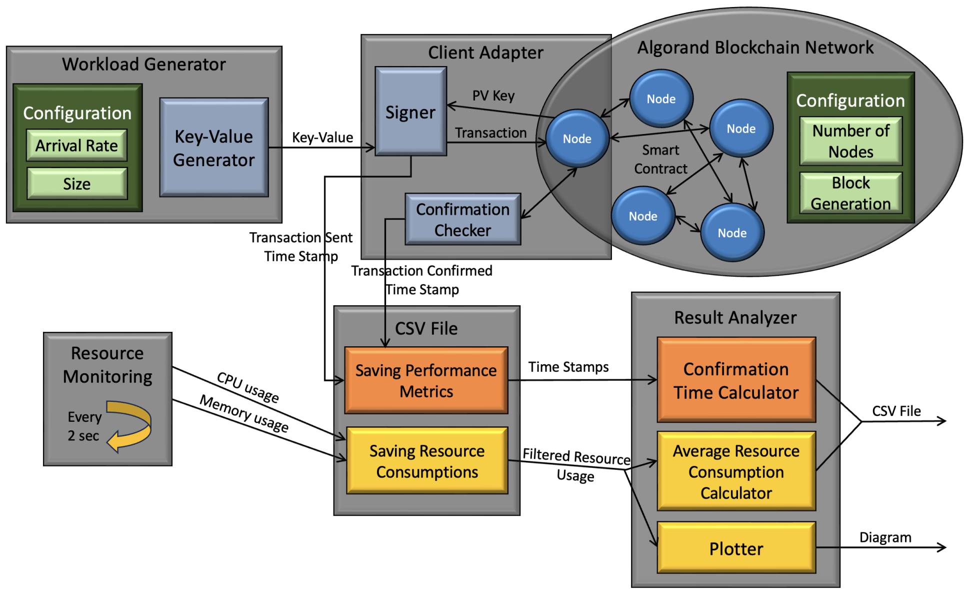

Figure 3.4: YorkBlockSim components

3.2.1 Creating an Algorand blockchain network

I used Algorand sandbox in order to create a network for testing purposes. Sandbox is an isolated virtual machine in which potentially unsafe software code can execute without affecting network resources or local applications. In order to create the sandbox for the Algorand blockchain network, I had to take these steps:

- Setting up the Algorand sandbox: We have different configuration files for setting the private or public Algorand network. We need to select the type of network that we want

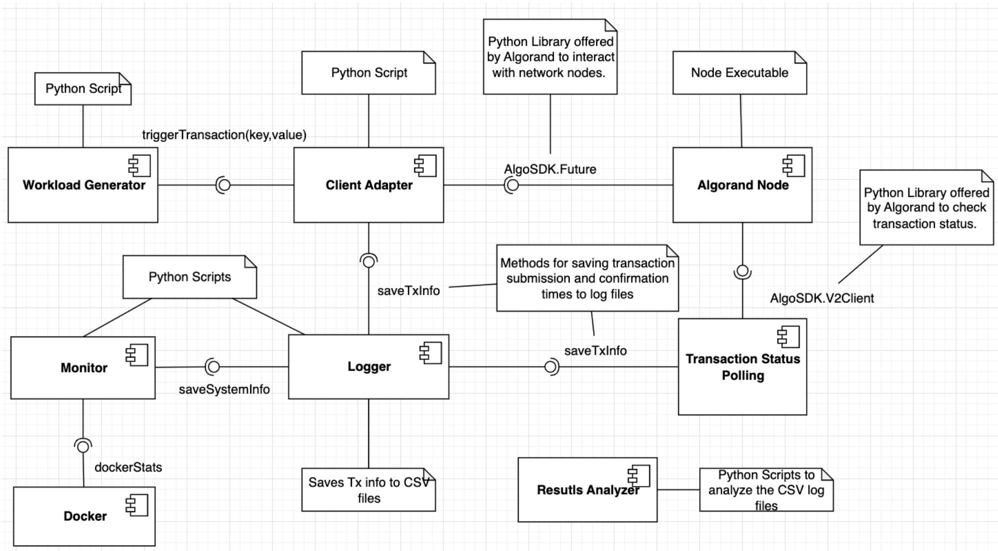

Figure 3.5: UML Component diagram

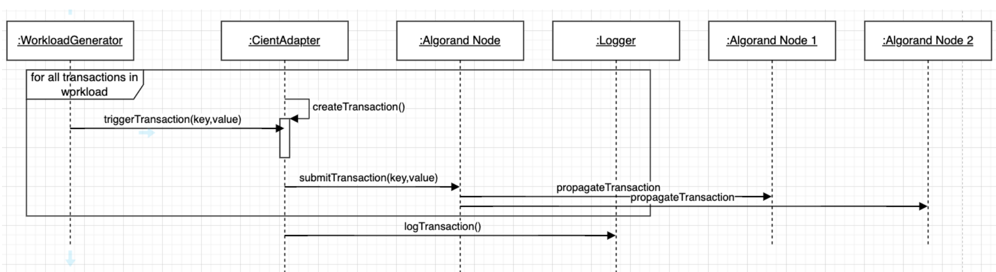

Figure 3.6: UML Sequence diagram

to create and then add all of them to the project folder including the docker file. In order to create the Algorand network we will run the bash file containing the required scripts to start the network inside the docker.

- Customize the settings: We can set the type of network that we want to run our experiment on it. We can also set the number of nodes and policy for adding a new block. Transactions could be saved in batches over a constant period of time, or we could have a new block for each transaction that we receive on the blockchain.

- Creating an account: Whether we are working with a private network or a public network, we need to create an account to communicate with other nodes in the blockchain. In a public network, we can run a specific command that will create a new account for us and generates pair of a public and private key and a wallet. By having the private key, we can access our mnemonic. We can use mnemonics later to prove that we are the owner of the account and we can send our transactions to the blockchain. The account we create in a private network is local and we cannot use it to communicate with other networks. We can set characteristics for this local account in the config file.

- Funding the accounts: In order to participate in the Algorand network, we need some Algos, the native cryptocurrency of the Algorand blockchain. If we are working on a public blockchain, we can acquire Algos on a cryptocurrency exchange or get them from the Algorand faucet. The Algorand Faucet Dispenser is a tool that allows developers and users to request 5 Algos, for free. The Algorand Faucet Dispenser is used for testing and experimenting with the Algorand blockchain network. It is designed to provide Algos to developers and can be used to experiment with applications on the Algorand network. We can provide our Algorand address and the faucet will load 5 algos to our account.

3.2.2 Implement adequate Smart Contract

A smart contract is an agreement between two people or entities in the form of computer code programmed to execute automatically [47]. The idea of the smart contract is to be independent of a trusted third party like a bank. A smart contract contains code that defines the rules and conditions of the agreement, as well as the actions that should be taken if those conditions are met. Once the conditions are met, the smart contract automatically executes the specified actions, which will be recorded on the blockchain network. Smart contracts are immutable, meaning that they cannot be altered once they have been deployed to the blockchain network. This ensures that the terms and conditions of the agreement cannot be changed without the agreement of all parties involved.

For Algorand, we are using PyTeal language to implement the smart contract. PyTeal is a Python-based language that provides a syntax for defining the logic and conditions of a smart contract. The reason for this name is that, first we need to write our smart contract in Python language using the required libraries and then we will compile this code to get our Teal code. The Teal code is the low-level language that can be used by the Algorand blockchain.

In our PyTeal smart contract, first, we need to check if the transaction is valid according to the rule of the smart contract. For instance, we check if the user has the minimum algos to pay for the transaction fee. In this experiment, we are going to generate a pair of key-value and we will save this on our blockchain to measure the performance metrics and resource consumption. As a result, we need a smart contract that supports saving a key-value pair. So, we have to define two variables in our smart contract and then we need an application to pass these two parameters and save on our variables.

Algorand smart contracts are implemented using two programs:

Approval Program The Approval program is responsible for processing all application calls to the contract, with the exception of the clear call (described in the next paragraph). This program is responsible for implementing most of the logic of an application. This program specifies the conditions under which a transaction can be considered valid and approved by the network. This program will succeed only if one nonzero value is left on the stack upon program completion.

Clear State Program The clear state program is used to handle accounts using the clear call to remove the smart contract from their balance record. This program will pass or fail the same way the approval program does.

I am providing the PyTeal code for the smart contract in appendix 6.2. This code has been compiled to get the Teal code which is understandable for the Algorand blockchain.

3.2.3 Implementing the workload generator

In the workload generator, I am generating random keys and values and placing them in a JSON format to send them to the localhost on port 3000 so that my server can receive them to process them further. In the workload generator, I am able to set the length of the keys and values. Also, I can determine the rate at that I want the workload generator to generate those tuples. Furthermore, I can set the number of key-value pairs that I want to be generated before the experiment is over.

3.2.4 Implement corresponding client adapter

The workload will send data to a client adapter that will be responsible for submitting the transaction to the Blockchain network in the correct format. The payload format sent by the workload is a JSON format as follows ”function”: ”Write”, ”args”: args. Where args is the table format holding the pair of data to store in the Blockchain network. I have implemented a client adapter that runs on port 3000 and has an API endpoint named invoke that will be invoked by the workload generator. The client adapter needs to call the correspondent smart contract and pass the received data, then after sending the data to the Blockchain network must return a response with status 200 if successful and 400 in case of an error. By default, the workload sends the data to a service (client adapter) that runs on localhost:3000 and calls an API named invoke. In my client adapter, first, I am going to compile the smart contract that is written to get the Teal file. After that, I will create an application and submit it on the blockchain and I wait for it to be confirmed on the blockchain. From now on, I can call my application by having the application Id and then I can pass the user's private key and the key-value pair to send the request to save them on the blockchain.

Following the submission of a request, I do not await the confirmation of the transaction before initiating the next one. This methodology mirrors real-world network behavior, where users independently submit transactions without waiting for each other's confirmations. Instead of pausing to receive confirmations, I commence sending transactions with predefined time intervals and track transaction IDs within a list. Concurrently, I implement a parallel process to monitor and identify transactions that have successfully been written onto the blockchain. These transactions are subsequently removed from the list. Consequently, even if a single transaction experiences delays and takes minutes to be included in the blockchain, this approach safeguards the uninterrupted submission and recording of other transactions. It is worth noting that the system's response time remains unaffected by this approach; it just enables the system to continuously submit transactions, irrespective of the completion status of preceding ones.

Client adapter have been implemented in Python as the other modules are written in Python. This makes the setup and running of the project very easy. I am providing the Python code for the client adapter in appendix 6.3.

3.2.5 Calculating the success rate and response time

After generating the key-value pair and sending them as an HTTP request, we will receive them on the server and pass them to the client adapter. In the client adapter, we will create a new transaction and add the key-value that we have. Then we will sign this transaction using the user's private key and send it to the blockchain. As one of the metrics for our experiment, we need to have the number of transactions that have been successfully submitted on the blockchain. Also, we need to have the time that it takes from sending the data to the blockchain to the time that we receive the confirmation. This confirmation means that our transaction has been written in one of the blocks in our blockchain. So, calculating the success rate means the number of transactions that have been sent to the blockchain and we received the confirmation for them. And the response time means the difference between the timestamps when we send the transaction to the blockchain and the timestamp when we receive the confirmation.

3.2.6 Writing the resource monitor script

In this step, we are going to measure the resource consumption of the system including CPU usage, and memory usage. We want to know how much computational power is required to write a specific number of transactions. Also, we want to compare the CPU and Memory utilization for different modes of experiments that we run to see which one is more efficient. We can get these data by monitoring the Docker using the "docker stats" command. This would output the percentage of the CPU and memory that is being used in each timestamp. I wrote a Python code, which would run this command regularly and filter and save the result in a CSV file. Before start submitting the transactions to the network, I run this Python code to start the monitoring. After all of the transactions have been submitted and confirmed on the blockchain, it is time to stop the monitoring and start the next step which is processing the results like calculating the average.

3.2.7 Running the experiment and receiving the results

Performance comparison: Compare performance and resource utilization for Algorand with different rates of data generation and the number of nodes and size of data.

There are different parameters that we can set before running our experiments. We can set the rate that the workload generator will generate the key-value pairs. We can set the size of the data that the workload generator will make. Also, we can set the number of tuples that we want to have in our experiment. Rather than the workload generator, we can change some settings in the Alogrand network as well. We can change the number of nodes that we have in our blockchain. This could change the result since it could affect the consensus protocol as more or fewer nodes are participating in the consensus. Also, the initial amount of algos that each node has, is a factor that can play a role in the consensus protocol. I can also set my private Algorand network to generate a new block for each transaction that I add or generate a new block for a batch of transactions. These two could have their own use in real-life based on our needs.

In this step, I will run my experiment based on the different parameters and modes that I have mentioned and then I will get the result for each case. Getting the result means that I am going to monitor my CPU, Memory, and Network consumption over the time that the experiment is running. Also, I will measure the success rate and the response time for each of the cases so that I can compare the results and conclude which set of parameters works better for which scenario.

Chapter 4

Experiments

4.1 Overview

In this experiment, I aim to measure the resource utilization and performance metrics of Algorand under different workload conditions and network configurations. As mentioned in chapter 1, I analyze the effect of four parameters on resource utilization and performance metrics of the Algorand blockchain. The parameters are as follows:

- Transaction arrival rate
- Transaction size
- Number of nodes
- Block generation policy

For each of these parameters, I conducted experiments to measure resource utilization and performance metrics. In order to measure the resource utilization, I monitored Docker which displays real-time statistics for each running container. I monitored and saved the CPU and memory usage during the experiment. Algorand blockchain network is constantly working on generating a new block by reaching a consensus on the content of the block. The content of a block is the list of transactions that will be included in that block. So, by generating a new block, we are moving to the next round. After receiving the confirmation for a transaction, I also get the number of rounds that the transaction was written on. Number of rounds is one of the performance metrics in this experiment. We can find out how many blocks were needed to write all of the transactions that we submitted to the blockchain network during an experiment using the number of rounds metrics.

For every transaction in the network, there are two different timestamps. The first one is when we submit the transaction to the network and the second one is when we receive the confirmation that the transaction has been included in a block and was written on the blockchain. To determine the response time, I recorded the timestamp when I submitted the first transaction to the blockchain and also recorded the timestamp when I received the confirmation for all of the transactions that were submitted during that experiment. For example, in the case of submitting 1000 transactions to the network, I save the timestamp that I submit the first transaction and I wait to receive all 1000 transactions' confirmation. By subtracting the first timestamp from the second, and subsequently dividing the result by the total number of transactions sent, I calculated the average response time. It should be mentioned that after each experiment, I save the time interval that I intentionally add between sending transactions. So, at the end of the experiment, I will be able to deduct this time period from the total time so that it would not affect the response time. Response time is the other performance metrics in the experiment.

After getting these metrics, I provided a comparison between different modes of running the experiment and analyzed the effect of that parameter on the network. Here is what each of these metrics means:

- CPU usage: This metric shows the percentage of CPU time that a container is currently using.
- Memory usage: This metric shows the amount of memory that a container is currently using. It includes both the container's working set (i.e., the memory that is currently in use) and its memory limits (i.e., the maximum amount of memory that the container is allowed to use). It is important to note that Docker does not provide individual data for each node in the blockchain network. Instead, it provides information on the total memory usage across all nodes in the blockchain network.
- Average response Time: To determine the response time, I recorded the timestamp when I submitted the first transaction to the blockchain and also recorded the timestamp when I received the confirmation for all of the transactions that were submitted during that experiment. By subtracting the timestamp that the first transaction was submitted from the timestamp that all of the transactions were confirmed and dividing the result by the number of transactions sent, we get the average response time. These timestamps

are recorded by the client adapter which submits the transaction to the blockchain network and checks for confirmation.

- Number of Rounds: This metric gives us the number of blocks that were created to save all the transactions that were sent to the blockchain. In Algorand in every round of consensus, a new block will be created that lets us save the newly submitted transactions to it.

4.2 Sanity Check

These tests are intended to demonstrate the simulator's ability to provide reasonable measurements. I conducted these tests on a private Algorand network consisting of 5 nodes.

Comparison of High and Low transaction arrival rate In my first comparison, I ran the experiment with two different transaction arrival rates. I am providing the details for all of the experiments in table 4.1. I want to point out that in these experiments, I utilized a 4-core CPU. When you see a CPU usage of 400%, that indicates the system is running at its maximum capability. Similarly, a CPU usage of 150% signifies that one and a half CPU cores are actively in use.

Here, I am comparing and analyzing the initial two experiments that I conducted. In the experiment with a high transaction arrival rate, from the first time stamp to almost the last ones, the CPU utilization is higher than in the experiment with a low transaction arrival rate. This is because there are more transactions per second in our network that need to

Workload traffic	**High**	**Low**	**Sudden Increase**	**Sudden Decrease**	**Oscillating**
Transaction arrival rate (TPS)	100	5	5, 100	100, 5	100, 5, 100, 5
Avg CPU %	139.45	83.26	109.8	107.36	106.91
Avg Response time (s)	0.3589	0.5007	0.4371	0.4301	0.4389

Table 4.1: Specifications and results for the experiments with different workload traffic

be processed, and the CPU is responsible for this processing. Although we submitted the same number of transactions (100) in both experiments, we expect the first experiment to finish sooner since we are sending transactions at a higher rate. As a result, the blockchain receives the transactions earlier and has more transactions in the queue to add to a block at the time it wants to add a new block. This is different from the low-traffic workload scenario, where the blockchain could have more transactions in a block, but it has not received enough transactions when creating a new block. As a result, in the low arrival rate experiment, the blockchain has to generate more blocks for the same number of transactions, which takes more time. This is why the response time of the high arrival rate experiment is shorter than that of the low arrival rate experiment.

Comparison of Sudden Increase and Sudden Decrease in transaction arrival rate

Here, I set the workload to submit the first half of the packets at a low rate and the second half of the requests at a high rate. This experiment is labeled as the sudden increase in workload. Although, in the next experiment, I did the reverse. The workload begins at

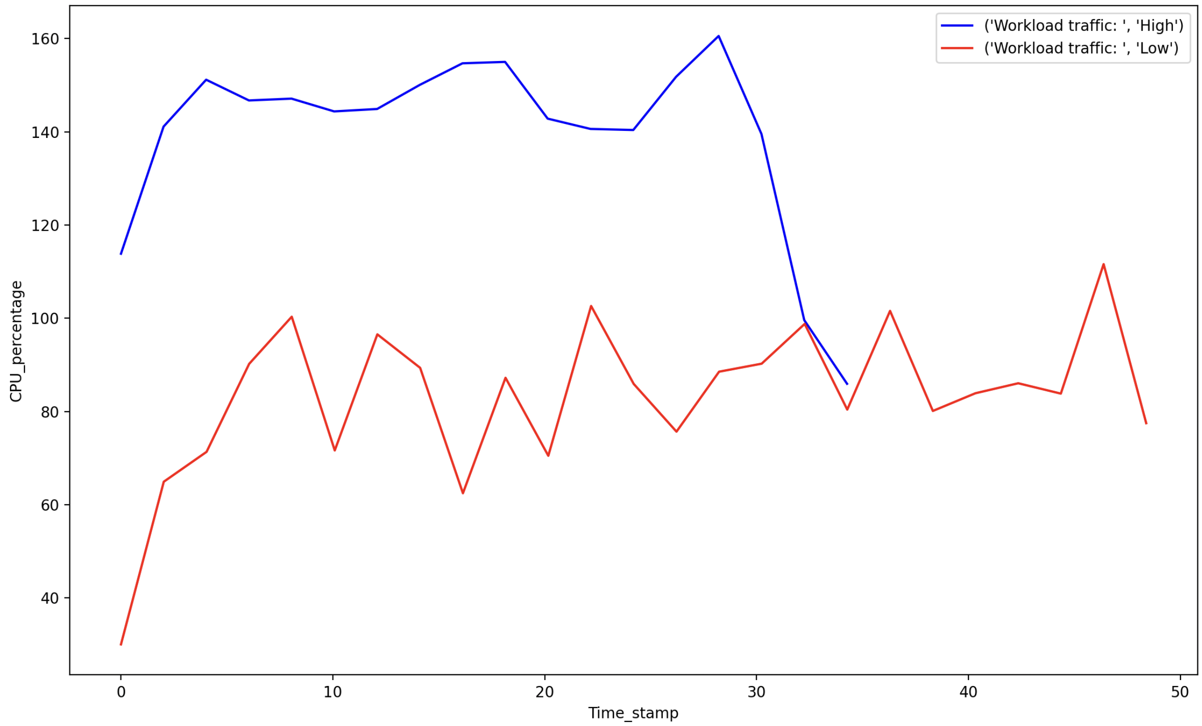

Figure 4.1: CPU consumption for high and low traffic workloads

high traffic and continues with low traffic for the second half of the data. We labeled this experiment as the sudden decrease in workload.

Submitting transactions to the Algorand blockchain at a higher rate can lead to higher CPU and memory utilization because of the consensus algorithm used by the blockchain. Algorand's consensus algorithm, Pure Proof-of-Stake (PPoS), requires a large number of participating nodes to validate each transaction. When transactions are submitted to the Algorand network at a faster rate, the number of transactions that must be validated and processed by the participating nodes also increases. The validation process involves complex cryptographic operations that require computational resources, including the CPU and memory. As the rate of transactions submitted to the Algorand blockchain increases, the workload on the CPU and memory also increases proportionally to process and validate these transactions. Additionally, since Algorand's consensus algorithm aims to achieve consensus on a large number of transactions in real-time, the CPU and memory must work intensively to process transactions within a very short period. Thus, submitting transactions to the Algorand blockchain at a higher rate can lead to higher CPU and memory utilization due to the consensus algorithm's demands for computational resources to validate and process transactions.

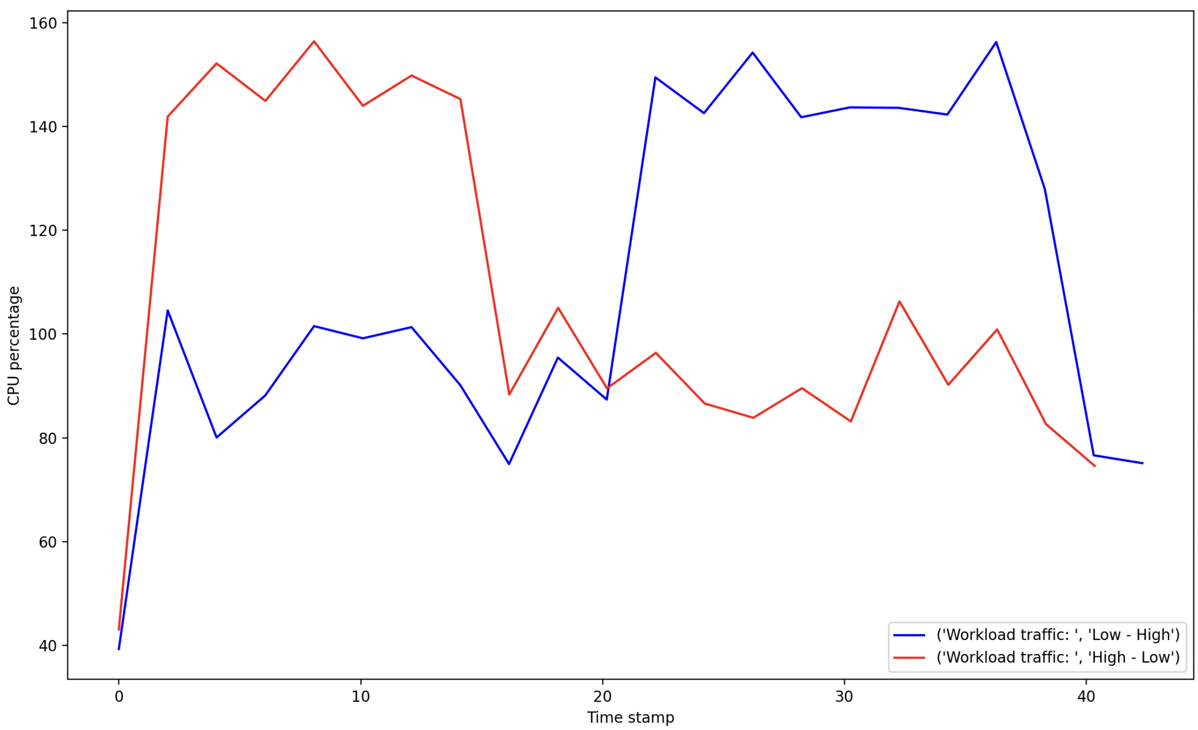

Figure 4.2: CPU consumption for the sudden increase and sudden decrease in traffic of workloads

Comparison of Low Traffic and Oscillating Traffic Workload Here, I am showing the CPU usage for oscillating and low transaction arrival rates. As expected, when the transactions submitted per second is high, the CPU usage is high and during the low traffic workload, the CPU usage is lower.

In Table 4.2, I am summarizing the sanity check comparisons.

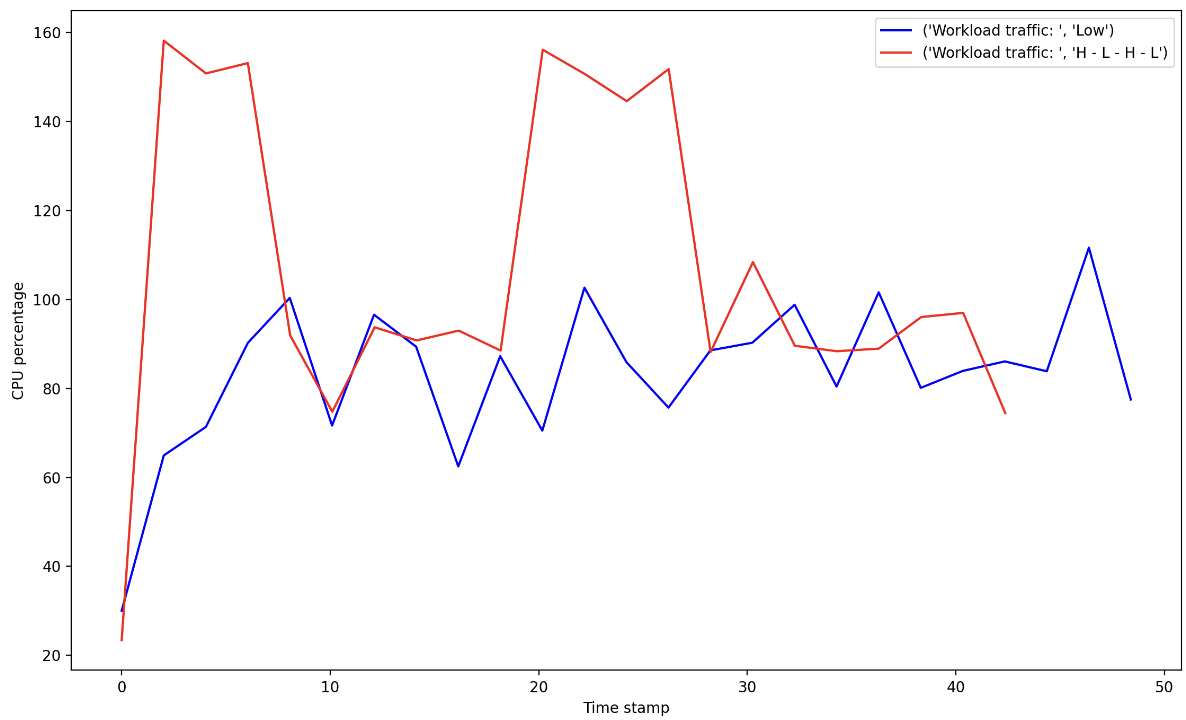

Figure 4.3: CPU consumption for oscillating and low traffic of workloads

4.3 Transaction Arrival Rate

I conducted these tests on a private Algorand network consisting of 5 nodes.

Comparison	High and Low	Sudden Increase and Sudden Decrease	Oscillating and low
Description of the sanity check	Testing CPU usage and response time	Testing CPU usage	Testing CPU usage
Behavior expected	Higher CPU usage when traffic is high. and shorter response time when traffic is high	Increase in CPU usage when traffic increases	Increase and decrease in CPU usage when traffic goes high and low
Behavior observed	As expected	As expected	As expected

Table 4.2: Summarizing sanity check comparisons

Research Questions The experiments conducted in this section aim to address the following questions:

- How does modifying the transaction arrival rate affect CPU usage?
- How does modifying the transaction arrival rate affect the number of rounds required to submit all transactions?
- How does modifying the transaction arrival rate affect the response time?

In the Algorand blockchain, a "round" is a fixed time interval where a new block is proposed and validated. Each round has a set duration and a randomly chosen block proposer that proposes a block containing transactions. Other network participants, called verifiers, independently validate the proposed block and vote on its acceptance. If the proposed block receives enough votes, it is considered accepted, and the round is successful. Accepted blocks are added to the blockchain, ensuring efficient and secure consensus in the Algorand network.

Insights these experiments offer By answering these questions, we can gain insights into the effects of modifying the transaction arrival rate on the Algorand blockchain.

Expected Behavior It is anticipated that these parameters will demonstrate a linear relationship once a specific duration in the experiment has elapsed.

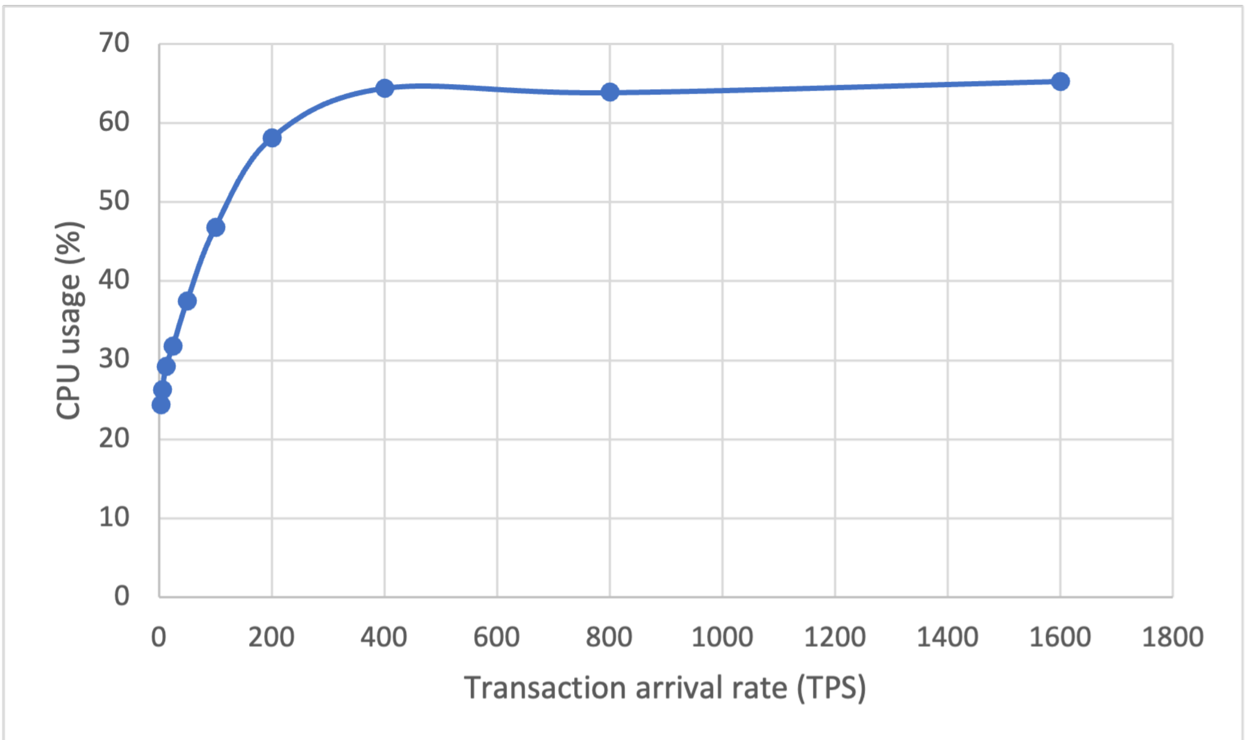

Figure 4.4: CPU usage and transaction arrival rate

Observed Behavior In the next paragraphs, I discuss the observed behavior of the relationship between CPU usage, number of rounds, average response time, and transaction arrival rate.

CPU Usage and transaction arrival rate As depicted in Figure 4.4, there is a clear correlation between the transaction arrival rate and CPU usage. As the arrival rate increases, CPU usage also shows a corresponding increase. However, it is worth noting that once the workload speed surpasses the 400 transactions per second (TPS) threshold, the CPU usage no longer exhibits the same level of growth. Instead, it stabilizes around 64% utilization.

In the Algorand blockchain, after writing and deploying a smart contract, we need to create an application and by calling that application with specific parameters, we can submit our transactions to the network. Calling the application takes a short amount of time. But when the workload is working very fast, the process of calling the application will be the bottleneck of submitting the transactions. Normally, the bottleneck is the time interval that we add to the workload generator to adjust the transaction arrival rate. But when the data is arriving faster than the limit of the network, then it does not matter how much we increase the rate of generating the data. The network will receive the same rate. As I checked the details of calling the application from the experiments, I noticed that the case in which I submit the transactions with a 0.000625 second time interval (which corresponds to 1600 TPS) and the case with a 0.00125 second time interval (which corresponds to 800 TPS), both cases the network is receiving the transactions with the same rate which as mentioned shows the delay that calling the application to submit the transaction is causing.

Average response time and transaction arrival rate The average response time demonstrates an inverse relationship with the transaction arrival rate. This correlation arises from the fact that each experiment involves the submission of the same number of transactions

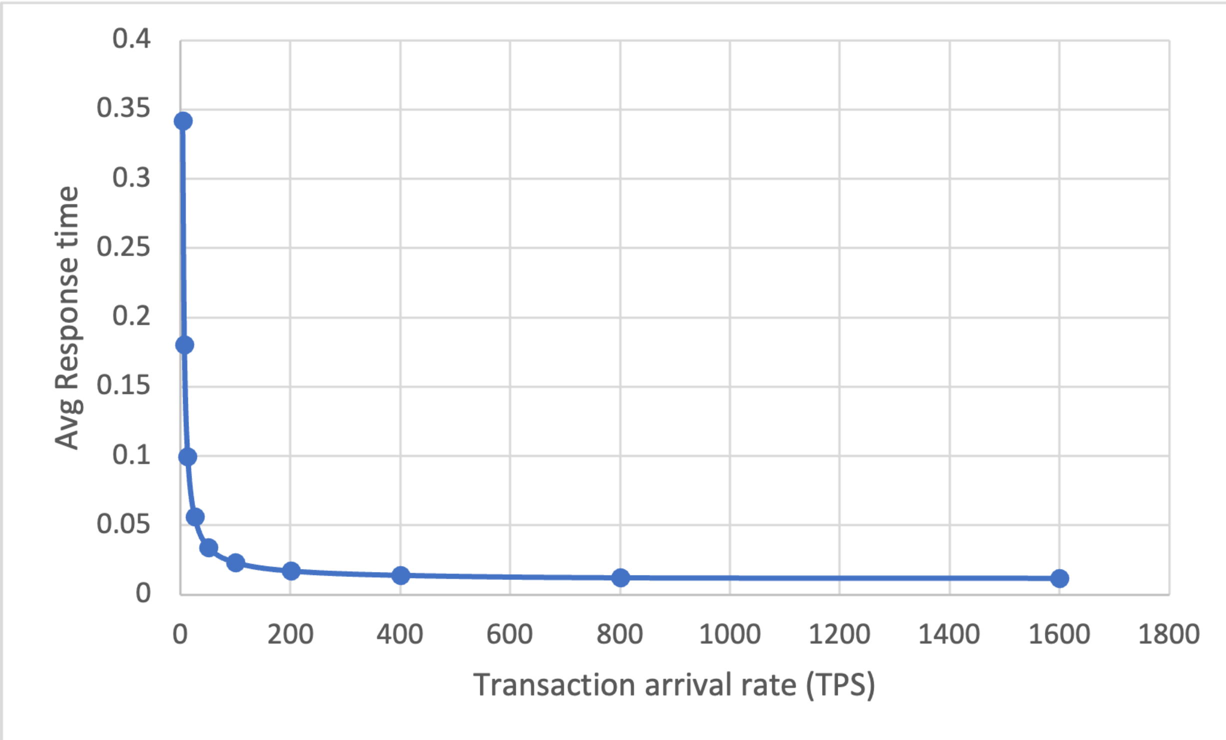

Figure 4.5: Average response time and transaction arrival rate

at varying speeds. For calculating the average response time in my experiment, I measure the time it takes from the submission of the first transaction until the time that I receive the confirmations for all of the submitted transactions. And I divide this number by the total number of transactions that were submitted. Based on this definition, average response time includes the time interval between submitting the transactions to the network as well. This will directly decrease the response time in the case that the transaction arrival rate is higher. In other words, the time interval between sending transactions is lower. Consequently, a higher speed of transaction submission results in a shorter overall response time. It should be mentioned that all of the submitted transactions were successfully received and written on the blockchain network. Otherwise based on the definition of response time, I would not get the second timestamp which is generated after getting the confirmations for all of the transactions.

In Figure 4.5, it is evident that beyond a transaction arrival rate of 400 transactions per second (TPS), the rate of change in the response time decreases, eventually leading to a linear relationship. When the transaction arrival rate exceeds 400 TPS, the response time stabilizes or shows minimal changes. As mentioned in the previous experiment, there is a limit to the rate of the transaction that can be sent to the blockchain network. So, when we continue to increase the rate of generating the data, the application which is responsible to submit the transaction will be called with the constant rate. Because after the application is called, there are some commands that need to be executed which will take some time. This time will be the bottleneck for the system only when the data is generated at a very high speed which is more than 400 TPS.

Number of Rounds and transaction arrival rate The relationship between the number of rounds and transaction arrival rate exhibits a striking similarity to the response time and transaction arrival rate. By submitting the same number of transactions but at a higher rate, the network receives the transactions earlier. Consequently, during the generation of each new block in every round, the network has a larger pool of transactions to include. This results in a reduced need for additional blocks to accommodate all the received transactions. Figure 4.6 clearly demonstrates that beyond a transaction arrival rate of 400 transactions per second (TPS), the rate of decrease in the number of rounds becomes more gradual. This is

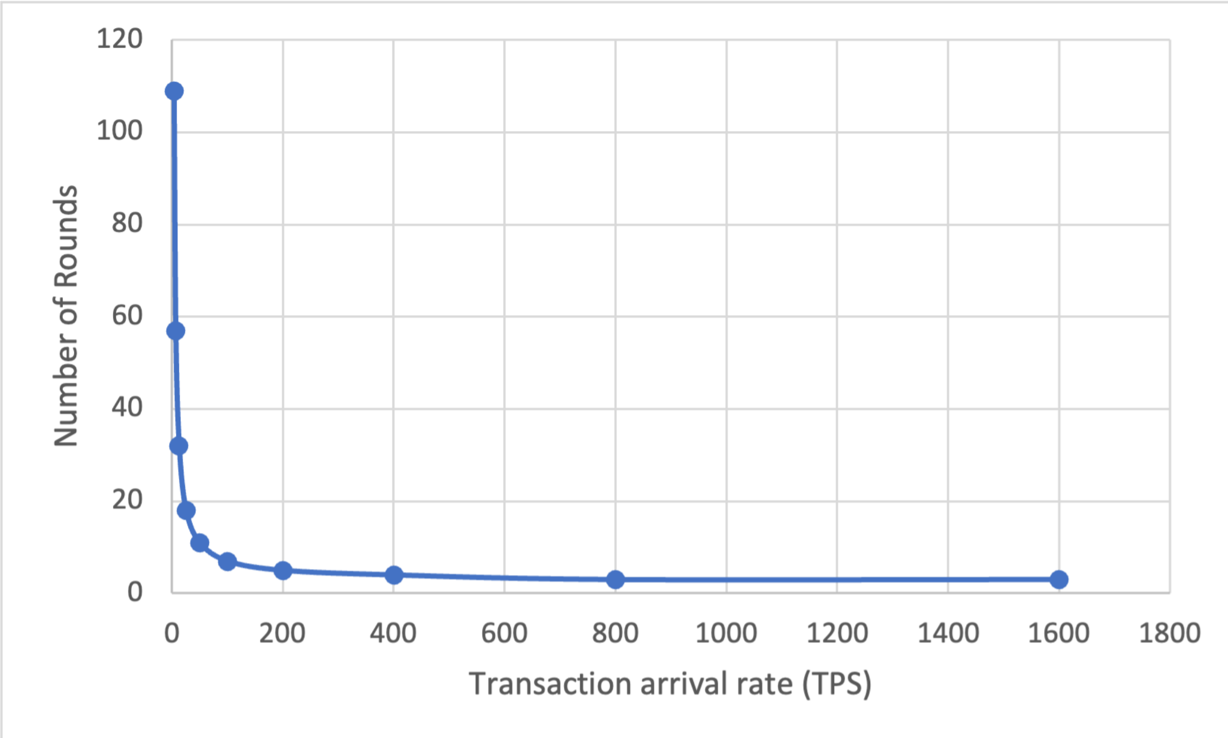

Figure 4.6: Number of rounds and transaction arrival rate

happening because even if we can send transactions quickly enough to fill up the capacity of a block, there is still a constraint in the form of the time it takes to reach a consensus and create a new block. Once the transaction arrival rate surpasses a specific threshold (in this case, 400 TPS), the network faces limitations in accommodating all the incoming transactions within the current block due to its limited capacity. Consequently, even if the transaction arrival rate continues to increase beyond 400 TPS, the number of rounds remains relatively constant because you have to wait to reach a consensus and create a new block.

4.4 Transaction Size

I conducted these tests on a private Algorand network consisting of 5 nodes.

Research Questions The experiments conducted in this section aim to address the following questions:

- How does modifying the size of transactions sent to the network affect CPU and memory usage?
- How does modifying the size of transactions sent to the network affect response time and number of rounds?

Insights these experiments offer By answering these questions, we can gain insights into the effects of modifying the size of transactions on resource utilization and performance metrics of Algorand network.

Expected Behavior I anticipated that altering the transaction size would result in a significant impact on memory usage or the number of rounds.

Observed Behavior After conducting a series of experiments involving varying sizes of input, it became evident that modifying the size of the transaction has minimal impact on both CPU and memory usage. Furthermore, the response time and the number of rounds needed to submit all transactions remained relatively unchanged across different transaction sizes. Consequently, altering the size of the workload did not result in significant variations in resource utilization or performance metrics. I am presenting the data illustrating the impact of altering the transaction size on various parameters in Table 4.3. It should be noted that the number of rounds in each experiment is always an integer. The presence of a decimal point is due to the fact that I conducted multiple iterations of each experiment with identical parameters to obtain a more consistent and reliable output.

Transaction size	**CPU %**	**Memory %**	**Response time (s)**	**Number of rounds**
2	48.374	4.998	0.023058	7
4	43.094	5.1	0.021794	6.6
8	46.77	5.198	0.023012	7
16	45.584	5.2	0.022432	6.8
32	44.45	5.376	0.022412	6.8
64	45.136	5.16	0.02242	6.8
128	46.98	5.24	0.02295	7

Table 4.3: Impact of transaction size on resource utilization and performance metrics

4.5 Number of Nodes

I conducted these tests on private Algorand networks consisting of different numbers of nodes.

Research Questions The experiments conducted in this section aim to address the following questions:

- How does modifying the number of nodes affect response time?
- How does modifying the number of nodes affect CPU and memory usage?

Insights these experiments offer By answering these questions, we can gain insights into the effects of modifying the number of nodes on the resource utilization and response time of the Algorand network.

Expected Behavior I expect to observe an increase in CPU and memory usage by increasing the number of participating nodes in the consensus protocol of Algorand. Also, I expect to see an increase in the response time of the network since more nodes are participating in the consensus protocol.

Observed Behavior Modifying the number of nodes participating in the consensus protocol of the blockchain had a notable effect on memory consumption. Increasing the number of nodes in the network resulted in higher memory utilization throughout the experiment. As the number of nodes participating in the consensus protocol increases, the number of messages exchanged between nodes also increases. Each node needs to store and process these messages, leading to higher memory utilization. The memory requirement grows as more messages are received and stored during the consensus process.

According to the data presented in Table 4.4, it is evident that increasing the number of nodes primarily results in an increase in memory usage. On the other hand, CPU usage and response time exhibit fluctuating patterns, with their values remaining relatively consistent.

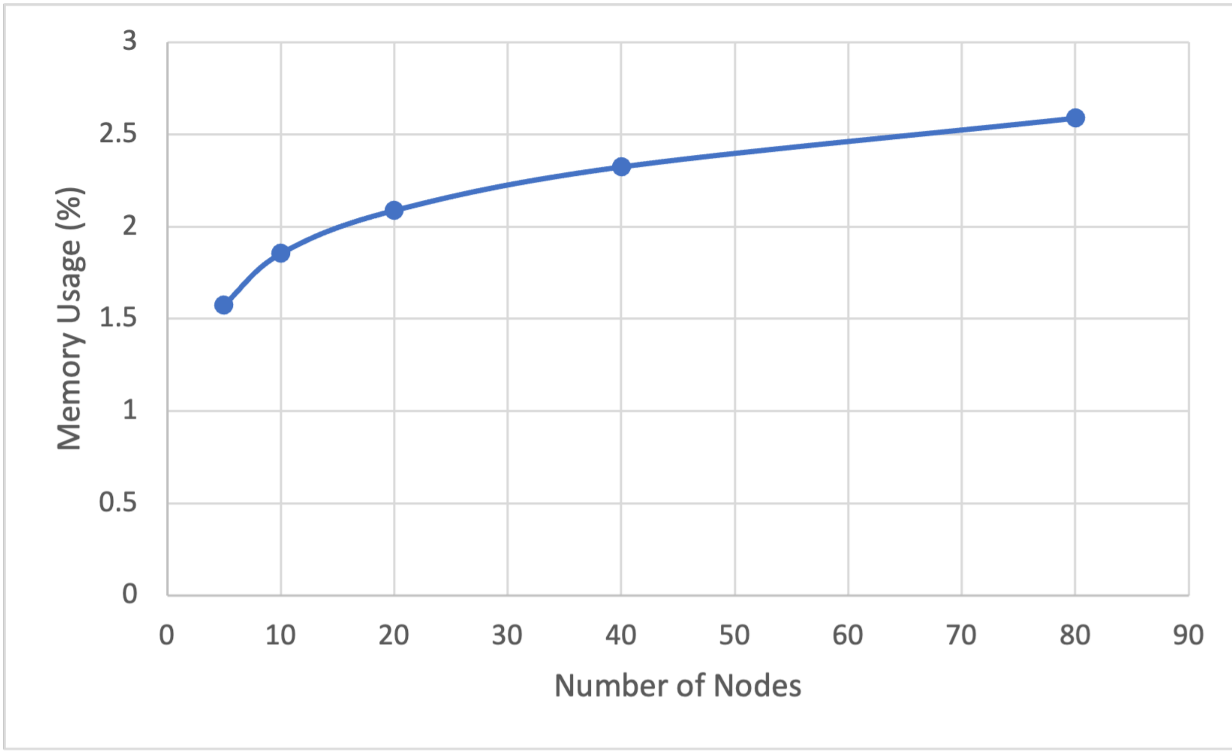

Figure 4.7: Memory Usage and Number of Nodes

This suggests that the impact on CPU usage and response time is minimal. The term "response time" in this context denotes the average duration for a transaction to be submitted on the blockchain. A lower response time indicates a faster processing of transactions within the network. According to the data presented in Table 4.4, the network with 10 nodes exhibits the fastest performance, while the one with 80 nodes shows the slowest. However, the experimental results do not consistently demonstrate a linear decrease in network speed with the addition of more nodes.

Number of Nodes	CPU %	Memory %	Response time (s)
5	26.969	1.572	0.0722
10	25.802	1.852	0.07185
20	26.854	2.087	0.07322
40	24.936	2.324	0.07281
80	29.074	2.589	0.07417

Table 4.4: Impact of number of nodes on resource utilization and performance metrics

4.6 Block Generation

I conducted these tests on two different private Algorand networks consisting of 5 nodes.

Research Questions For this section, I created two different private Algorand networks. One of them created a new block for the batch of the transactions that it received and the other network created a new block for each transaction that it receives. The experiments conducted in this section aim to address the following questions:

- How does changing the block generation policy affect CPU and memory usage?
- How does changing the block generation policy affect response time?

Insights these experiments offer By answering these questions, we can gain insights into the effects of changing the block generation policy on the resource utilization and performance metrics of the Algorand network.

Expected Behavior Due to the number of transactions that I sent to the network in this experiment, I expect the network that creates a block for each transaction to be faster and more efficient.

Observed Behavior In this experiment, a different approach was implemented, where the network generated a new block for each incoming transaction, rather than waiting for a pool of transactions to accumulate and adding them to a new block at the start of a new round. The comparison of these results is presented in Table 4.5. This modification offers a couple of advantages.

First, it enables faster transaction confirmation receiving since there is no need to wait for a specific time interval before creating a block. As soon as a transaction is received, a new block can be created for it, reducing the overall transaction confirmation time. Second, this approach may utilize fewer CPU and memory resources compared to creating a block for a batch of transactions. By generating blocks for a batch of transactions, transactions are accumulated within each round before a block is proposed. This means that the system needs to process and store a larger pool of transactions before creating a block. The accumulation of transactions requires additional CPU and memory resources to handle the larger dataset. Also, the process of creating a block for each round involves selecting a set of transactions to include, forming the block, and validating its contents. This process can be computationally intensive, requiring more CPU resources. Additionally, validating a larger block with multiple transactions may require more memory to store the necessary data for verification. So, by generating blocks for individual transactions, the system can potentially optimize resource usage, as it doesn't need to process and store a large pool of transactions before creating a block. This could lead to improved efficiency and reduced computational requirements.

It's worth noting that while this modification may offer benefits such as faster transaction processing and resource optimization, there may also be trade-offs to consider. For example, generating a block for each transaction could result in increased data flow and overhead, as well as potential challenges in achieving consensus efficiently. Creating a block for every transaction means that the number of blocks generated will increase significantly. This can result in increased network traffic as each block needs to be propagated across the network to reach all participating nodes.

In an Algorand blockchain, the validation of each block by participating nodes is an essential step before it can be accepted and added to the blockchain. As the frequency of block creation increases, the validation process can become more resource-intensive. Each individual block needs to be processed and verified by the nodes, which requires computational resources.

In practice, various blockchain platforms have distinct capacities for processing transactions per second (TPS) based on their applications. For instance, bitcoin TPS capacity is around 7-10 TPS [48], Ethereum is around 15-30 TPS [49], Ripple is up to 1500 TPS [49].

In this experiment, the rate that the transactions were submitted to both of the networks were 100 TPS but I had a limitation on the number of transactions I could send to my private Algorand blockchain. Due to this limitation, I was not able to test the network with a high number of transactions. However, in the scenario where I created a block for each transaction instead of each round, I observed better resource utilization and performance metrics. It's important to note that the observed better resource utilization and performance metrics in my experiment are specific to the limited number of transactions tested and may not necessarily scale linearly with a higher transaction load. The performance and resource utilization characteristics of the network can vary based on factors such as transaction size, network conditions, and the capabilities of the participating nodes.

Block Generation	Response time (s)	Transaction submission (s)	Receiving the confirmation (s)	CPU %	Memory %
Per Round	0.0722	0.01726	0.05495	26.969	1.572
Per Transaction	0.02317	0.0184	0.00475	20.471	1.126

Table 4.5: Specifications for the experiments with different block generation policy

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In conclusion, the experiments conducted in this thesis aimed to simulate the Algorand blockchain platform and analyze its performance metrics and resource utilization under various parameters. Here are the parameters of the experiment:

- Transaction arrival rate
- Transaction size
- Number of nodes
- Block generation policy

Here are the resource utilization and performance metrics that were measured:

- CPU usage

- Memory usage
- Average response time
- Number of rounds

Several key observations were made regarding the relationship between different parameters and metrics.

One significant finding was the correlation between CPU usage and transaction arrival rate. As the arrival rate increased, CPU usage also showed a corresponding increase. However, once the workload speed surpassed the limitation for the process of calling the application that is responsible for submitting the transactions, CPU usage stabilized at around 64% utilization. In other words, although the data is being generated at a higher speed, submitting the transactions to the network has a limit which would be the bottleneck after reaching the limit of 400 TPS.

Regarding average response time, it exhibited an inverse relationship with the transaction arrival rate. Higher speeds of transaction submission resulted in shorter overall response times. This is happening because the response time includes the time intervals between generating the transactions. However, beyond a transaction arrival rate of 400 TPS, the rate of change in response time decreased, eventually leading to a linear relationship. This is due to the rate of transaction that the network can receive which I mentioned will be the bottleneck when the data is being generated at a rate higher than 400 TPS.

The relationship between the number of rounds and transaction arrival rate also demonstrated an interesting behavior. Beyond a transaction arrival rate of 400 TPS, the rate of decrease in the number of rounds became more gradual. This was due to the system's limitations in generating a new block. Even if the transaction arrival rate continued to increase, the response time remained relatively constant because of the need to wait for consensus and block creation.

The experiments also explored the impact of transaction size and the number of nodes on resource utilization. It was found that modifying the size of the transaction had minimal impact on CPU and memory usage, as well as response time and the number of rounds needed to submit all transactions. Increasing the number of nodes participating in the consensus protocol led to higher memory utilization due to the increased number of messages exchanged and stored by each node during the consensus process.

Furthermore, a different approach was implemented by generating a new block for each incoming transaction instead of waiting for a pool of transactions to accumulate. This modification offered advantages such as faster transaction confirmation and optimized resource usage. However, trade-offs such as increased network traffic and challenges in achieving consensus efficiency were also noted.

The experiments conducted in this thesis provide valuable insights into the behavior of the Algorand blockchain platform under different parameters. The findings contribute to a better understanding of the platform's performance, and resource utilization under different conditions.

5.2 Future Works

The future steps for the current research are listed as follows:

- **YorkBlockSim Expansion**: YorkBlockSim [46] is the simulator used in this project to simulate Algorand private network. Adding the Ethereum blockchain platform to the YorkBlockSim will allow us to compare the resource monitoring and performance metrics of these two blockchain platforms under the same workload generator. Although there are other simulators that are able to simulate Ethereum, having these two platforms in the same simulator would give us more precise data since the workload and resource monitoring modules are behaving exactly the same.

- **Scalability Analysis**: Explore the scalability of the Algorand blockchain platform by conducting experiments with a higher number of transactions and larger network sizes. Investigate the impact of scaling on performance metrics, resource utilization, and the system's ability to handle increased transaction loads.

- **Optimization Strategies**: Develop and evaluate optimization strategies to improve resource utilization and performance metrics. This could involve exploring different block generation policies, transaction prioritization mechanisms, or consensus algorithms. Investigate the trade-offs between resource utilization, and transaction confirmation time to find the most efficient configurations.

- **Dynamic Block Size**: Investigate the impact of dynamically adjusting the block size based on network conditions and transaction arrival rates. Explore adaptive block size algorithms that can efficiently accommodate varying transaction workloads while maintaining optimal resource utilization and response times.

- **Network Optimization**: Analyze network topology and communication protocols to identify potential bottlenecks and optimize the efficiency of message propagation. Investigate the impact of different network configurations on overall system performance and resource utilization.

- **Security Analysis**: Conduct a comprehensive security analysis of the Algorand blockchain platform, including potential vulnerabilities and attack vectors. Evaluate the effectiveness of existing security mechanisms and propose enhancements to mitigate security risks.

Bibliography

[1] M. Murray, "Tutorial: A descriptive introduction to the blockchain," *Communications of the Association for Information Systems*, vol. 45, no. 1, p. 25, 2019.

[2] H. F. Atlam, A. Alenezi, M. O. Alassafi, and G. Wills, "Blockchain with internet of things: Benefits, challenges, and future directions," *International Journal of Intelligent Systems and Applications*, vol. 10, no. 6, pp. 40–48, 2018.

[3] S. Tanwar, N. Gupta, C. Iwendi, K. Kumar, and M. Alenezi, "Next generation iot and blockchain integration," *Journal of Sensors*, vol. 2022, 2022.

[4] M. Conti, A. Gangwal, and M. Todero, "Blockchain trilemma solver algorand has dilemma over undecidable messages," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1–8.

[5] A. Kaushik, A. Choudhary, C. Ektare, D. Thomas, and S. Akram, "Blockchain—literature survey," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, IEEE, 2017, pp. 2145–2148.

[6] D. Valdeolmillos, Y. Mezquita, A. González-Briones, J. Prieto, and J. M. Corchado, "Blockchain technology: A review of the current challenges of cryptocurrency," in *International Congress on Blockchain and Applications*, Springer, 2019, pp. 153–160.

[7] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[8] G. A. F. Rebello, G. F. Camilo, L. C. Guimarães, L. A. C. de Souza, and O. C. M. Duarte, "On the security and performance of proof-based consensus protocols," in *2020 4Th conference on cloud and internet of things (CIot)*, IEEE, 2020, pp. 67–74.

[9] A. Corso, "Performance analysis of proof-of-elapsed-time (poet) consensus in the sawtooth blockchain framework," Ph.D. dissertation, University of Oregon, 2019.

[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, p. 21 260, 2008.

[11] P. Szalachowski, D. Reijsbergen, I. Homoliak, and S. Sun, "{Strongchain}: Transparent and collaborative {proof-of-work} consensus," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 819–836.

[12] A. Back *et al.*, "Hashcash-a denial of service counter-measure," 2002.

[13] R. Bansal. "Proof of elapsed time or poet consensus in blockchain (part 22)." (2023), [Online]. Available: `https://www.linkedin.com/pulse/proof-elapsed-time-poet-consensus-blockchain-part-22-roohi-bansal`.

[14] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, “On security analysis of proof-of-elapsed-time (poet),” in *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, Springer, 2017, pp. 282–297.

[15] S. Kaur, S. Chaturvedi, A. Sharma, and J. Kar, “A research survey on applications of consensus protocols in blockchain,” *Security and Communication Networks*, vol. 2021, 2021.

[16] S. Zhang and J.-H. Lee, “Analysis of the main consensus protocols of blockchain,” *ICT express*, vol. 6, no. 2, pp. 93–97, 2020.

[17] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, “Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain,” 2018.

[18] L. Zhang and Q. Li, “Research on consensus efficiency based on practical byzantine fault tolerance,” in *2018 10th International Conference on Modelling, Identification and Control (ICMIC)*, IEEE, 2018, pp. 1–6.

[19] D. Ongaro and J. Ousterhout, “The raft consensus algorithm,” *Lecture Notes CS*, vol. 190, 2015.

[20] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling byzantine agreements for cryptocurrencies,” in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[21] M. Abbasi, M. H. Manshaei, M. A. Rahman, K. Akkaya, and M. Jadliwala, "On algorand transaction fees: Challenges and mechanism design," in *ICC 2022-IEEE International Conference on Communications*, IEEE, 2022, pp. 5403–5408.

[22] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theoretical Computer Science*, vol. 777, pp. 155–183, 2019.

[23] B. Haney and A. Chaudhury, "Algorand autonomous," *Brian Haney and Archie Chaudhury, Algorand Autonomous*, 2021.

[24] H. Arslanian, "The emergence of new blockchains and crypto-assets," in *The Book of Crypto*, Springer, 2022, pp. 99–119.

[25] M. F. Esgin, V. Kuchta, A. Sakzad, *et al.*, "Practical post-quantum few-time verifiable random function with applications to algorand," in *International Conference on Financial Cryptography and Data Security*, Springer, 2021, pp. 560–578.

[26] E. Anceaume, A. Pozzo, T. Rieutord, and S. Tucci-Piergiovanni, "On finality in blockchains," *arXiv preprint arXiv:2012.10172*, 2020.

[27] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[28] R. Paulavičius, S. Grigaitis, and E. Filatovas, "A systematic review and empirical analysis of blockchain simulators," *IEEE access*, vol. 9, pp. 38 010–38 028, 2021.

[29] R. Paulavičius, S. Grigaitis, and E. Filatovas, "An overview and current status of blockchain simulators," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2021, pp. 1–3.

[30] E. Babulak and M. Wang, "Discrete event simulation," *Aitor Goti (Hg.): Discrete Event Simulations. Rijeka, Kroatien: Sciyo*, p. 1, 2010.

[31] J. Bélanger, P. Venne, and J.-N. Paquin, "The what, where and why of real-time simulation," *Planet Rt*, vol. 1, no. 1, pp. 25–29, 2010.

[32] J. A. Weekley, B. Hawkes, N. Guenole, and R. E. Ployhart, "Low-fidelity simulations," *Annu. Rev. Organ. Psychol. Organ. Behav.*, vol. 2, no. 1, pp. 295–322, 2015.

[33] J. M. Carey and K. Rossler, "The how when why of high fidelity simulation," 2020.

[34] M. Alharby and A. Van Moorsel, "Blocksim: A simulation framework for blockchain systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 135–138, 2019.

[35] L. Stoykov, K. Zhang, and H.-A. Jacobsen, "Vibes: Fast blockchain simulations for large-scale peer-to-peer networks," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*, 2017, pp. 19–20.

[36] Y. Aoki, K. Otsuki, T. Kaneko, R. Banno, and K. Shudo, "Simblock: A blockchain network simulator," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2019, pp. 325–329.

[37] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM international conference on management of data*, 2017, pp. 1085–1100.

[38] L. Alsahan, N. Lasla, and M. Abdallah, "Local bitcoin network simulator for performance evaluation using lightweight virtualization," in *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, IEEE, 2020, pp. 355–360.

[39] E. Androulaki, A. Barger, V. Bortnikov, *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[40] M. Rasolroveicy, W. Haouari, and M. Fokaefs, "Public or private? a techno-economic analysis of blockchain," in *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*, 2021, pp. 83–92.

[41] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.

[42] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.

[43] S. Kwon and J.-H. Lee, "Divds: Docker image vulnerability diagnostic system," *IEEE Access*, vol. 8, pp. 42 666–42 673, 2020.

[44] K. Jangla, "Docker compose," in *Accelerating Development Velocity Using Docker*, Springer, 2018, pp. 77–98.

[45] *What is a rest api?* `https://www.redhat.com/en/topics/api/what-is-a-rest-api`, Published: 2020-05-08.

[46] S. S. Monfared, *Blockchain-simulator*, `https://github.com/sahand-monfared/blockchain-simulator`, 2023.

[47] *What is a smart contract?* `https://www.investopedia.com/terms/s/smart-contracts.asp`, Published: 2023-05-31.

[48] R. Bala and R. Manoharan, "Security enhancement in bitcoin protocol," in *2018 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, IEEE, 2018, pp. 1–4.

[49] S. V. Akram, P. K. Malik, R. Singh, G. Anita, and S. Tanwar, "Adoption of blockchain technology in various realms: Opportunities and challenges," *Security and Privacy*, vol. 3, no. 5, e109, 2020.

Chapter 6

Appendix

6.1 RESTful API

```
import saveKeyValue
from flask import Flask, jsonify, request

app = Flask(__name__)
@app.route('/invoke', methods=['POST'])
def create_store():
    request_data = request.get_json()
    new_store = {
        'key': request_data['key'],
        'value': request_data['value']
    }
    if saveKeyValue.main(request_data['key'], request_data['value']) == "
```

```
    successful":
            print("Data stored on the blockchain successfully!")
            return jsonify(new_store)
        else:
            return jsonify({"error": "Bad Request"}), 400

app.run(port=3000)
```

Listing 6.1: RESTful API

6.2 PyTeal Smart Contract

```
from pyteal import *

def approval_program():
    on_creation = Seq([
        App.globalPut(Bytes("Count"), Int(0)),
        App.globalPut(Bytes("Key"), Txn.sender()),
        App.globalPut(Bytes("Value"), Txn.sender()),
        Return(Int(1))
    ])

    scratchCount = ScratchVar(TealType.uint64)
    add = Seq([
        scratchCount.store(App.globalGet(Bytes("Count"))),
        App.globalPut(Bytes("Count"), scratchCount.load() + Int(1)),
```

```
        Return(Int(1))
    ])

    deduct = Seq([
        scratchCount.store(App.globalGet(Bytes("Count"))),
        If(scratchCount.load() > Int(0),
            App.globalPut(Bytes("Count"), scratchCount.load() - Int(1)),
        ),
        Return(Int(1))
    ])

    keyValue = Seq([
        App.globalPut(Bytes("Key"), Txn.application_args[1]),
        App.globalPut(Bytes("Value"), Txn.application_args[2]),
        Return(Int(1))
    ])

    handle_noop = Cond(
        [And(
            Global.group_size() == Int(1),
            Txn.application_args[0] == Bytes("Add")
        ), add],
        [And(
            Global.group_size() == Int(1),
            Txn.application_args[0] == Bytes("Deduct")
```

```
        ), deduct],
        [And(
            Global.group_size() == Int(1),
            Txn.application_args[0] == Bytes("KeyValue")
        ), keyValue],
    )

    program = Cond(
        [Txn.application_id() == Int(0), on_creation],
        [Txn.on_completion() == OnComplete.OptIn, Return(Int(0))],
        [Txn.on_completion() == OnComplete.CloseOut, Return(Int(0))],
        [Txn.on_completion() == OnComplete.UpdateApplication, Return(Int(0))],
        [Txn.on_completion() == OnComplete.DeleteApplication, Return(Int(0))],
        [Txn.on_completion() == OnComplete.NoOp, handle_noop]
    )

    return compileTeal(program, Mode.Application, version=5)

def clear_state_program():
    program = Return(Int(1))
    return compileTeal(program, Mode.Application, version=5)
```

Listing 6.2: PyTeal Smart Contract

6.3 Client Adapter

```
import random
import time
from datetime import datetime
import base64
from algosdk.future import transaction
from algosdk import account, mnemonic
from algosdk.v2client import algod
from pyteal import *
import os

iterationCallApp = 100
firstQuarterSleepTime = 0.01
secondQuarterSleepTime = 0.01
thirdQuarterSleepTime = 0.01
forthQuarterSleepTime = 0.01
workloadLabel = 'Fast'
sleepTimeCheckConfirmation = 0.1

creator_mnemoic = "violin piece assault sense fatal loud allow alcohol curve neutral crack ivory another safe image road exist another apology knock wire acid buddy ability index"
algod_address = "http://localhost:4001"
algod_token = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
```

```python
transactionsList = []
appId = 1

def wait_for_confirmation(client, transaction_id, timeout):
    start_round = client.status()["last-round"] + 1
    current_round = start_round

    while current_round < start_round + timeout:
        try:
            pending_txn = client.pending_transaction_info(transaction_id)
        except Exception:
            return
        if pending_txn.get("confirmed-round", 0) > 0:
            return pending_txn
        elif pending_txn["pool-error"]:
            raise Exception('pool error: {}'.format(pending_txn["pool-error"]))
        client.status_after_block(current_round)
        current_round += 1
    raise Exception("pending tx not found in timeout rounds, timeout value = : {}".format(timeout))

def check_for_confirmation(client):
    for item in transactionsList[:]:
        try:
```

```
            pending_txn = client.pending_transaction_info(item)
        except Exception:
            return
        if pending_txn.get("confirmed-round", 0) > 0:
            print("--------------------------------------------")
            print("Transaction confirmed:", item)
            print("confirmed-round:", pending_txn['confirmed-round'])
            print("--------------------------------------------")
            transactionsList.remove(item)

def format_state(state):
    formatted = {}
    for item in state:
        key = item['key']
        value = item['value']
        formatted_key = base64.b64decode(key).decode('utf-8')
        if value['type'] == 1:
            if formatted_key == 'voted':
                formatted_value = base64.b64decode(value['bytes']).decode(
    'utf-8')
            else:
                formatted_value = value['bytes']
            formatted[formatted_key] = formatted_value
        else:
            formatted[formatted_key] = value['uint']
```

```python
    return formatted

def read_global_state(client, addr, app_id):
    results = client.account_info(addr)
    apps_created = results['created-apps']
    for app in apps_created:
        if app['id'] == app_id:
            return format_state(app['params']['global-state'])
    return {}

def create_app(client, private_key, approval_program, clear_program,
   global_schema, local_schema):
    sender = account.address_from_private_key(private_key)
    on_complete = transaction.OnComplete.NoOpOC.real
    params = client.suggested_params()
    txn = transaction.ApplicationCreateTxn(sender, params, on_complete,
   approval_program, clear_program, global_schema, local_schema)
    signed_txn = txn.sign(private_key)
    tx_id = signed_txn.transaction.get_txid()
    client.send_transactions([signed_txn])
    wait_for_confirmation(client, tx_id, 5)
    transaction_response = client.pending_transaction_info(tx_id)
    app_id = transaction_response['application-index']
    print("Created new app_id:", app_id)
    return app_id
```

```python

def call_app(algod_client, private_key, index, app_args):

    sender = account.address_from_private_key(private_key)
    params = algod_client.suggested_params()
    txn = transaction.ApplicationNoOpTxn(sender, params, index, app_args)
    signed_txn = txn.sign(private_key)
    tx_id = signed_txn.transaction.get_txid()
    transactionsList.append(tx_id)
    algod_client.send_transactions([signed_txn])
    # wait_for_confirmation(client, tx_id, 5)
    check_for_confirmation(algod_client)
    print("Application called")
    print("--------------------------------------------")
```

Listing 6.3: Client Adapter