A Performance Evaluation Framework for Public Blockchain based Systems

by

Fakid Arman
23141035
Mahdi Hossain
20301194
A.S.M Amir Abdullah
20101219
Md Danial Islam
23241067
Maruf Bin Murtuza
23141042

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
October 2024

Declaration

It is hereby declared that

1. The thesis report submitted is our own original work while completing the degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis Report does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Fakid Arman
20101219

Mahdi Hossain
20301194

A.S.M. Amir Abdullah
23141042

MD. Danial Islam
23241067

Maruf Bin Murtuza
23141035

Approval

The thesis titled "A Performance Evaluation Framework for Public Blockchain based Systems" submitted by:

1. Fakid Arman (23141035)
2. Mahdi Hossain (20301194)
3. A.S.M Amir Abdullah (20101219)
4. Md. Danial Islam (23241067)
5. Maruf Bin Murtuza (23141042)

of Summer, 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering in October 22nd, 2024.

Examining Committee:

Supervisor:
(Member)

Md. Sadek Ferdous, Ph.D.

Professor
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam, Ph.D.

Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, Ph.D.

Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

With the revolution of blockchain technology, smart contracts open a door for programs to be executed inside the blockchain. This allows programmers to build applications and deploy them in different blockchain systems. Over the years these smart contracts have been redefined offering transparency, immutability, and distributed consensus. However, if we were to evaluate how each of these applications performs and which parameters are used to identify the performance, it would be unclear given the scarcity of a proper blockchain evaluation system in today's time. Our research aims to establish a comprehensive performance evaluation protocol specifically for public blockchains, referencing insights from seminal works. Our goal is to leverage important metrics that will be necessary to provide a standardized framework for evaluating and comparing the performance of blockchain based applications. We aim to help developers by providing a framework that lets them measure and compare their smart contracts on different blockchain systems. This solves the problems they face when deploying smart contracts and allows us to practically test important theories about different blockchain systems.

Keywords: Blockchain; Performance; Evaluation; Ethereum; Tron.

Acknowledgement

Firstly, all praise to the Great Allah, for whom our thesis has been completed without any major interruption.

Secondly, to our supervisor, Dr. Md. Sadek Ferdous, for his kind support and advice in our work. He helped us with our shortcomings and whenever we needed help.

Thirdly, to Research Assistant Md. Yeasin Ali, who helped us in our thesis planning and work and also guided us as a mentor.

And finally, to our parents, without their constant support, it may not be possible.

With their kind support and prayer, we are now on the verge of our graduation.

Contents

Declaration i

Approval ii

Abstract iii

Acknowledgement iv

Table of Contents v

List of Figures viii

List of Tables ix

1 Introduction 1
- 1.1 Problem Statement . . . 1
- 1.2 Research Objectives . . . 2
- 1.3 Report Structure . . . 3

2 Background 5
- 2.1 Blockchain . . . 5
- 2.2 The Evolution of Blockchain Technology . . . 5
  - 2.2.1 Origin of Blockchain Technology . . . 5
  - 2.2.2 Use of Blockchain Beyond Cryptocurrencies . . . 5
- 2.3 Key Components of Blockchain Performance . . . 6
  - 2.3.1 Consensus Mechanisms . . . 6
  - 2.3.2 Transaction Throughput and Latency . . . 7
  - 2.3.3 Security and Immutability . . . 7
- 2.4 Types of Blockchain . . . 7
  - 2.4.1 Public . . . 7
  - 2.4.2 Private . . . 8
- 2.5 Test Networks . . . 9
  - 2.5.1 Sepolia . . . 9
  - 2.5.2 Holesky . . . 9
  - 2.5.3 Shasta . . . 10
  - 2.5.4 Nile . . . 10
- 2.6 Blockchain Applications . . . 10
  - 2.6.1 Decentralized Applications . . . 11
  - 2.6.2 Smart Contracts . . . 11
  - 2.6.3 Non-Fungible Tokens (NFTs): . . . 11

2.7 Methods for Evaluating Blockchain Performance . . . 11
2.7.1 Simulation and Modeling . . . 11
2.7.2 Real-world Testing . . . 11
2.7.3 Benchmarking . . . 12
2.8 Challenges of Public Blockchains . . . 12
2.8.1 Scalability vs. Decentralization . . . 12
2.8.2 Energy Consumption . . . 13
2.8.3 Difference in Consensus Mechanisms . . . 13

3 Literature Review **15**

4 Research Proposal **21**
4.1 Proposal . . . 21
4.2 Research Methodology . . . 21
4.2.1 Problem Identification . . . 23
4.2.2 Research Objective . . . 23
4.2.3 Literature Review . . . 23
4.2.4 Metric Analysis . . . 23
4.2.5 Metric Identification . . . 23
4.2.6 Platform Analysis . . . 23
4.2.7 Sample Framework Development . . . 23
4.2.8 Testing Framework . . . 24
4.2.9 Result Analysis . . . 24
4.3 Metric Wise Findings . . . 24
4.3.1 Common Findings . . . 24
4.3.2 Contradictory View . . . 24
4.4 Hardware and Setup Findings . . . 24
4.4.1 Common Findings . . . 25
4.4.2 Contradictory View . . . 25
4.5 Implementation wise findings . . . 25
4.5.1 Common Findings . . . 25
4.5.2 Contradictory View . . . 25
4.6 Experimental Design . . . 25
4.6.1 Common Findings . . . 26
4.6.2 Contradictory View . . . 26
4.7 Evaluation Metrics . . . 26
4.7.1 Primary Evaluation Metrics . . . 26
4.7.2 Secondary Evaluation Metrics . . . 26

5 System Model and Architecture **28**
5.1 Evaluation Metrics . . . 28
5.2 Software Architecture . . . 29
5.3 Protocol Flow . . . 31
5.4 Implementation . . . 32

6 Experiments **34**
6.1 Experimental Setup . . . 34
6.2 Result Analysis from Data and Graphs . . . 35
6.2.1 Testnet Result Analysis . . . 35

6.2.2 Overall Response Time vs Load . . . 38
6.2.3 TPS vs Load . . . 40
6.2.4 Cost & Resource Usage . . . 41
6.2.5 Local Network Result Analysis (Ethereum) . . . 48
6.2.6 Overall Response Time vs Load . . . 49
6.2.7 Overall Response Time vs Load . . . 50

7 Discussion **51**
7.1 Research Objective Analysis . . . 51
7.2 Advantages . . . 52
7.3 Challenges & Limitations . . . 53
7.4 Future Work . . . 54

8 Conclusion **55**

Bibliography **58**

List of Figures

2.1 Work Flow of Proof of Stake Consensus Mechanism [28] 6
2.2 Approaches to DLT Performance Evaluation [15] 12

3.1 Various Metrics Corresponding to a Blockchain System [9] 16
3.2 Categories of distributed ledger technologies [15] 17
3.3 DAGBENCH engine [10] 18

4.1 Research Methodology Flowchart 22

5.1 Software Architecture 29
5.2 Protocol Flow 31
5.3 Framework User Interface 32
5.4 Example Smart Contract 33

6.1 Overall Response Time vs Ethereum Load 39
6.2 Overall Response Time vs Tron Load 39
6.3 Transaction Per Second vs Ethereum Load 40
6.4 Transaction Per Second vs Tron Load 40
6.5 Ether Used vs Increasing bits 42
6.6 TRX Used vs Increasing bits 42
6.7 Ether Used vs nth Fibonacci 44
6.8 TRX Used vs nth Fibonacci 44
6.9 Ether Used vs Array size 47
6.10 TRX Used vs Array size 47
6.11 Overall Response Time vs Ethereum Load 50
6.12 Transaction Per Second vs Ethereum Load 50

List of Tables

6.1 Ethereum Testnet Benchmark for Simple Function Smart Contracts . 35
6.2 Tron Testnet benchmark for Simple Function Smart Contracts 36
6.3 Ethereum Testnet benchmark for CPU Heavy Smart Contracts 36
6.4 Tron Testnet benchmark for CPU Heavy Smart Contracts 37
6.5 Ethereum Testnet benchmark for Data Heavy Smart Contracts 37
6.6 Tron Testnet benchmark for Data Heavy Smart Contracts 38
6.7 Ethereum bit increase input data benchmark for simple smart contract function 41
6.8 Tron bit increase input data benchmark for simple smart contract function 41
6.9 Ethereum nth Fibonacci benchmark for cpu heavy smart contract function 43
6.10 Tron nth Fibonacci benchmark for cpu heavy smart contract function 43
6.11 Ethereum Array size increase benchmark for data-heavy smart contract function 46
6.12 Tron Array size increase benchmark for data-heavy smart contract function 46
6.13 Ethereum Local benchmark for Simple Smart Contracts 48
6.14 Ethereum Local benchmark for CPU Heavy Smart Contracts 48
6.15 Ethereum Local benchmark for Data Heavy Smart Contracts 49

Chapter 1

Introduction

The invention of blockchain technology has significantly altered the digital landscape, introducing a new paradigm in the form of public blockchain applications. These applications, renowned for their transparency, immutability, and distributed consensus, have revolutionized various sectors. However, this evolution brings forth a critical challenge: the lack of a systematic and reliable protocol for evaluating the performance of public blockchain applications. This thesis research tries to overcome this challenge by benchmarking important metrics of a blockchain application and how public blockchains perform under different loads of data.

The current state of blockchain technology is marked by a notable absence of uniformity in performance evaluation methodologies. This lack of consistency becomes particularly evident when assessing platforms such as Ethereum [2], Solana [8], Tron [34], etc. where diverse approaches and metrics have led to a fragmented understanding of the efficacy of blockchain systems. This prevailing inconsistency underscores the pressing need for a unified benchmark against which performance can be measured. Our research aims to unify different evaluation methods into a clear framework, specifically for public blockchain systems. We want to help developers build blockchain solutions that are efficient and sustainable.

The primary objective of this thesis research is to develop a framework that provides user evaluations as to how a deployed application in the public blockchain system performs. We execute the framework using different testnets of blockchain systems to demonstrate its effectiveness in delivering meaningful performance insights.

Challenges like the lack of standardized evaluation metrics, the rapidly evolving nature of blockchain technology, and the diversity in blockchain architectures also need addressing.

1.1 Problem Statement

Currently, assessing the performance of applications is a crucial requirement in public blockchain systems. However, the existing situation reveals a discrepancy in the methodologies used to evaluate blockchain platforms. The use of diverse parameters and inconsistent experimental setups across different research efforts has resulted in

a puzzled state, making it difficult to establish a unified approach.

Consider, for instance, the evaluation of Ethereum, where some studies opt for private network configurations, while others leverage tools like Ganache [31], Hardhat, or Truffle. Recently, the usage of Ganache and Truffle suite is being sunsetted by their developers which makes it harder to test Ethereum smart contracts before deploying them in the mainnet.

A similar divergence unfolds across various other blockchain platforms, such as Solana and Tron regarding their metrics, implementations, and methodology criteria. These experimental setups are done in various ways which adds a sort of complexity to the performance measurement paradigm.

- First problem is that the existing methods evaluate blockchain performance, but there are no methods to assess the performance of blockchain applications.

Current evaluation methods focus on the overall performance of blockchain platforms. However, they do not provide specific tools or protocols to measure how individual blockchain applications perform when deployed. This gap makes it difficult to understand the effectiveness of decentralized applications (dApps) and smart contracts within different blockchain platforms.

- Second problem is about the current methods that lack user-friendly interfaces or easy ways for users to test their smart contracts.

Most existing performance evaluation tools are technical and require a deep understanding of blockchain systems, making them inaccessible to everyday users and developers. Without user interfaces or easier methods to upload and test smart contracts, users face significant challenges in deploying and optimizing their applications, hindering adoption and innovation in the blockchain field.

- Third problem is that there are limited or no options for benchmarking data and resource-heavy smart contracts.

Evaluating smart contracts that handle large amounts of data or require substantial computational resources is challenging due to the lack of specialized benchmarking tools. Existing frameworks do not adequately support the testing of resource-intensive applications, preventing developers from accurately measuring performance, identifying bottlenecks, and optimizing their smart contracts for better performance.

1.2 Research Objectives

Our research aims to create a standardized and adaptable framework that serves as a foundation for evaluating the performance of applications when deployed in various public blockchains. This framework will incorporate essential components and guidelines to ensure consistency and comparability across evaluations. To reach such a conclusion, we have outlined the specific research objectives focusing on the

development of such a comprehensive evaluation system:

i. **RO1: Metrics Prioritization and Integration:** To categorize and prioritize key performance metrics significant to public blockchains. This objective involves a thorough analysis of existing literature to determine the significant metrics that largely reflect the effectiveness and efficiency of public blockchains.

ii. **RO2: Construct Best Practices for Experimental Setup:** By stating and rationalizing top practices for doing experiments and implementations regarding public blockchains in research scenarios. Through the establishment of a standardized approach, aims to reduce experimental inconsistency and boost the reliability of performance evaluations.

iii. **RO3: Guidelines for Resource Efficiency:** To highlight the implication of resource efficiency by focusing on metrics related to throughput utilization, Overall response time, wallet consumption, and network usage.

iv. **RO4: User-Focused Performance Emphasis:** To prioritize parameters that directly influence the user experience on public blockchains, such as transaction throughput and overall response time. This will highlight the importance of lining up the evaluations with real-world usability for end-users.

v. **RO5: Practical proof of Consensus Mechanisms:** Our research aims to practically prove the theories of consensus mechanisms such as Proof of Stake (POS) and delegated Proof of Stake (dPOS). To analyze how each blockchain having different consensus performs against each other when the same application is deployed in both systems.

Through these research objectives, we hope to contribute a practical and applicable framework, raising a clear approach to evaluating the performance of applications deployed in public blockchains.

1.3 Report Structure

In this report, we briefly discuss the evaluation of blockchain performance and our research objectives in chapter 1.

In chapter 2, the background of blockchain and information needed for evaluating blockchain performance are discussed.

In chapter 3, we have presented our literature review. A literature review consists of seminal works that gave us deep insights as to how we can properly develop a public blockchain performance framework.

In chapter 4, we provide our analysis of the evaluation criteria from all the past research work we have reviewed. Additionally, we propose our system and research methodology by addressing the metrics we have selected.

In chapter 5, we present our framework's software architecture, protocol flow, and implementation.

In chapter 6, we analyze the results we found after measuring and benchmarking our system with different data loads.

In chapter 7, we discuss our findings and result analysis of our research objectives. We also address the advantages, and challenges we faced while doing the research limitations, and future work of our thesis.

In chapter 8, we focused on the significance of our report and how it makes an impact on future research and real-world use.

Chapter 2

Background

2.1 Blockchain

Blockchain technology has gained a lot of popularity in recent years for its potential to change various industries by providing secure and decentralized solutions for managing data and transactions. As the top organizations worldwide look into using blockchain in their systems, it's important to thoroughly assess how well this technology performs. This thesis aims to explore the details of blockchain performance evaluation, including the key factors, methods, and challenges involved in assessing the efficiency and effectiveness of blockchain systems.

2.2 The Evolution of Blockchain Technology

Early research began in the 1980s but the research didn't take off until 2009 with the revolution of Bitcoin [1]. The following section will discuss the origin of blockchain, how it was initially used by cryptocurrencies, and how it is now used for other purposes.

2.2.1 Origin of Blockchain Technology

The idea of a decentralized distributed ledger system was introduced during 1980s by the researchers like David Chaum, Stuart Haber, and others. However, the idea of blockchain mostly became popular with the introduction of Bitcoin in 2009 by someone named Satoshi Nakamoto [1]. Blockchain was the technology behind Bitcoin, providing a decentralized and transparent ledger for recording transactions. The decentralized nature of blockchain, along with its use of special codes, ensures the safety and security of the data stored within the system.

2.2.2 Use of Blockchain Beyond Cryptocurrencies

While blockchain first became popular with cryptocurrencies, its uses quickly expanded to various industries such as finance, supply chain, healthcare, and more. Smart contracts, which are self-executing contracts with the terms written into code, further expanded the use of blockchain technology. At present there are various uses of blockchain technologies As the number of uses grew, it became very important to assess how well blockchain systems perform.

2.3 Key Components of Blockchain Performance

Key factors such as consensus mechanisms, scalability, speed of transactions, and security are crucial for evaluating a blockchain system's performance. Understanding how these elements work in a given blockchain network, and how they perform under different conditions is important for our evaluation.

2.3.1 Consensus Mechanisms

One of the most important things that affect blockchain performance is the way that agreement is reached on the network. Traditional proof-of-work (PoW) and newer alternatives like proof-of-stake (PoS), delegated proof-of-stake (DPoS), and practical Byzantine fault tolerance (PBFT) have a big impact on the speed, scalability, and energy efficiency of blockchain networks [26]. It's important to evaluate the good and bad points of different ways of reaching an agreement to understand how they affect performance.

Proof of Work

The original consensus mechanism for Bitcoin is PoW [1]. In this consensus mechanism, the computational resource is used to help miners to solve complex mathematical puzzles. When the puzzle is solved the person first who solves it has the right to add a new block to the blockchain and receive a cryptocurrency reward. PoW is secure because it is computational as opposed to cryptographic, and however secure it is, we have to mine with a lot of power, which leads to scalability issues due to this consumption of power.

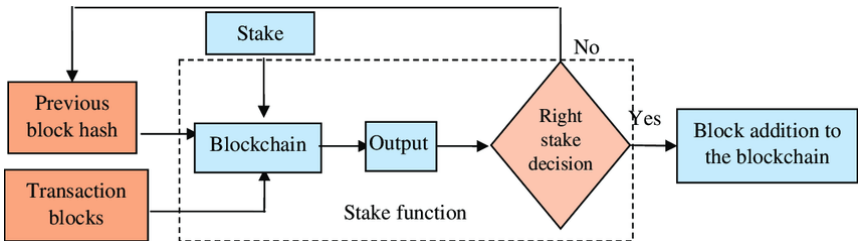


Figure 2.1: Work Flow of Proof of Stake Consensus Mechanism [28]

Proof of Stake

Proof of Stake (PoS) is a consensus mechanism used in blockchain networks to validate transactions and create new blocks. In this consensus, validators are chosen to validate transactions based on the number of coins they hold and are willing to stake as collateral [26]. The more coins a validator stakes, the higher the chance of being selected to add a new block to the blockchain and earn rewards. In Figure 2.1, we see the workflow of how the proof of stake consensus mechanism works. A block is added to the blockchain network using a staking process. It starts with validating the transaction blocks and the previous block's hash, followed by evaluating the stake function. If the stake decision is correct, the block is added to the blockchain, otherwise the block is rejected. If we compare it with proof of work, proof of stake

does not require any extensive computational power to solve complex mathematical analysis, which is why it is faster and more efficient than PoW.

Delegated Proof of Stake

Delegated Proof of Stake (DPoS) is an evolved version of PoS. In this consensus, stakeholders vote for a small group of trusted delegates or witnesses who are responsible for validating transactions and producing blocks [26]. The voting power is proportional to the amount of stake held, and stakeholders can change their votes to elect different delegates if current delegates perform poorly. However, the consensus is not decentralized as it is designed to be more democratic. A small number of trusted validators maintains the network, which leads to faster transaction time.

2.3.2 Transaction Throughput and Latency

The efficiency of a blockchain system is often measured by how many transactions it can handle in a certain amount of time. At the same time, transaction latency, which is how long it takes for a transaction to be confirmed, is very important for user experience. Balancing high throughput with low latency is essential for good blockchain performance.

2.3.3 Security and Immutability

The main value of blockchain comes from its security and immutability. It's very important for building trust in blockchain networks that the data is safe and can't be tampered with. But it's also important to check how cryptographic techniques and the trade-offs between security and speed affect performance.

2.4 Types of Blockchain

There are different types of blockchain systems such as public, private, consortium, and hybrid. In our research, we will be focusing on public blockchain systems but we will also be discussing private blockchain systems.

2.4.1 Public

Public blockchains are open-source and transparent, allowing anyone to view and participate in the network. They are typically used for cryptocurrencies like Bitcoin and Ethereum.

Ethereum

Ethereum is one of the earliest public blockchains that support the deployment of smart contracts and dApps in their network. At the start, this blockchain utilized a Proof of Work (PoW) consensus mechanism. However, it has always faced problems with scalability and high fees due to its early PoW algorithm, which allowed up to 15 TPS throughput [2]. At present, Ethereum 2.0 has migrated from PoW to PoS and integrates sharding with Layer 2 scaling. Now it achieves better transaction throughput and reduced costs, making it more suitable for a wide range of

decentralized applications. Despite its improved scalability and decentralization, Ethereum's performance under high network congestion is an area requiring continuous research. While the PoS upgrade has brought considerable enhancements, real-world performance benchmarks under varying loads would further solidify its claims of scalability. Ethereum remains a critical case study for public blockchain performance evaluation, particularly in assessing trade-offs between decentralization, energy efficiency, and transaction throughput.

Tron

Tron supports high scalability for performance and a DPoS consensus mechanism able to reach a theoretical upper limit of up to 2,000 transactions per second. The DPoS model ensures much quicker transaction finality and much lower costs, hence being ideal for applications entailing frequent microtransactions. With low fees and high throughput, Tron is suitable for both decentralized gaming and content platforms. However, on the other side, there are a couple of concerns regarding decentralization, considering that its validation power is highly centralized among super representatives. Yet, with Tron, deterministic finality ensures that confirmed transactions are secure and reliable.

Inside the Tron Blockchain Documentation, we found an excellent exposition of Tron's public blockchain platform, which discusses its scalability, transaction efficiency, and low cost [34]. Since Tron uses a Delegated Proof of Stake (DPoS) consensus mechanism, throughput, and latency are increased and far greater than Ethereum's Proof of Work (PoW) and Proof of Stake (PoS) consensus mechanisms. The charter covers the architecture of the Tron Virtual Machine compatibility with Ethereum's Solidity and makes Tron one of the most convincing options for the migration of decentralized applications (dApps) to the Tron network.

However, Tron, with its high throughput and a developer-friendly ecosystem can be looked at as a relevant study case while considering public blockchain performance evaluation thesis, but the question about centralization of Tron's DPoS system with the limited number of super representatives has to be further investigated. In addition, real-world performance benchmarks under heavy network loads are lacking from the documentation, which may be crucial for development.

2.4.2 Private

Private blockchains such as Hypreledger Fabric and Corda are controlled by a central authority and have limited access.

Hyperledger Fabric

Hyperledger Fabric works as a very modular, and easily extensible private blockchain platform aimed at enterprises. The architecture is a plug-and-play one where organizations can deploy customizable consensus mechanisms and smart contracts that are known as chaincode [6]. Permissioned networks are Hyperledger Fabric supported, allowing only allowed users to access and improve privacy and scalability.

In industry sectors, for example, supply chain, finance, and healthcare, where secure, effective transaction processing is required, it is widely used.

Corda

Corda is a distributed ledger platform developed by R3, created originally for financial industry use, which is being used today in many other industries [3]. Corda is a privacy and efficiency-enhancing blockchain as opposed to the traditional blockchains which batch transactions into blocks. Data is kept on a need-to-know basis and only shared with relevant parties. Corda can handle complex workflows, and smart contracts and comply with the requirements of regulatory bodies, making it well-suited to those applications that demand strict privacy and security provisions.

2.5 Test Networks

Test networks, or testnets, are blockchain environments that mimic mainnet conditions but allow developers to experiment without the risks and costs associated with actual mainnet transactions. For each blockchain system, there can be several testnets for different development usage. The importance of testnets is crucial for testing smart contracts, decentralized applications, and network performance under simulated real-world scenarios.

2.5.1 Sepolia

Sepolia is an Ethereum testnet that allows dApp developers and smart contracts to deploy and test on the Ethereum mainnet [33]. It functions as a Proof of Stake (PoS) network similar to Ethereum's post-merge consensus and provides a nearly identical environment for testing. The Sepolia is a testnet for Ethereum 2.0 functionalities like staking & transaction validation without the high transaction fees on the mainnet. Sepolia, with its ultra-low traffic and very low-cost environment, is a perfect space for people to test DeFi and NFTs that require an expansion of Ethereum infrastructure. That's because it has limits in terms of simulating the real-world aspects of Ethereum's network congestion and scalability, whose traffic and transaction volumes are lower than the mainnet's. There are several points at which Sepolia becomes relevant to our thesis, being capable of simulating a given performance of Ethereum 2.0 under varying transaction loads, ultimately giving us insight into how the public blockchain system, in general, handles the performance and transaction efficiency of transaction loads.

2.5.2 Holesky

Ethereum's latest testnet, announced in September 2023, named Holesky, is set to replace Goerli. Specifically, it supports a high throughput environment to perform large-scale testing of all Ethereum 2.0 functions such as staking, transaction processing, and all related validator activities [32]. With 1.4 billion Holesky ETH tokens for testing, Holesky has more Validators and can support a greater Transaction volume than previous Ethereum testnets, making it perfect for testing decentralized

applications (dApps) just prior to deployment on the mainnet. What makes Holesky great is it runs large-scale network conditions that hint at what Ethereum on the ground looks like, allowing devs to get a sense of what throughput means for transactions, what its scalability limits are, and what validator performance looks like. But Holesky just got launched and it is a testnet, so it still needs to grow; community support, tool compatibility, and even possible issues in development could still negatively impact Holesky's performance. What's more relevant to our research is Holesky's ability to simulate large-scale conditions and data on Ethereum's scalability and feasibility with performance in a high-volume network.

2.5.3 Shasta

TRON's testnet shasta [36] is meant to be the place where the dApps, smart contracts, and rector can be tested on a testnet that mirrors TRON's mainnet. TRON's Delegated Proof of Stake (DPoS) consensus mechanism provides high throughput, low latency, and scalability to Tron's blockchain. Developers can play around with the code with no financial risk and with free TRX tokens on offer to test. One of the strengths of Shasta is that it can mimic the mainnet and offer development in low traffic and at an affordable cost. It is the most friendly platform supporting smart contracts and decentralized applications with high transaction efficiency and thus is the best choice for developers developing projects on TRON. Yet, its omission of the mainnet's real-world TRON traffic is only one aspect, and possibly an insidious one: TRON's mainnet is a DPoS system, which brings along with it the risk of centralization that, if adopted for the mainnet, could affect TRON's security.

2.5.4 Nile

Another is Nile, an official testnet on the TRON's network, where we can test decentralized applications (dApps), smart contracts, and so on in a long-term field [35]. Nile is just like Shasta, reflecting TRON's mainnet environment, as well as supporting its Delegated Proof of Stake (DPoS) consensus mechanism. Nile is, however, dedicated to testing complex and big apps under the hood for a longer period of time. Nile's main strength is its stability and scalability, fit for developers interested in testing high-performance applications that need continuous validation as well as low transaction costs. Despite being closely aligned to the TRON mainnet, the Nile's DPoS system raises red flags around centralization, and actual world performance benchmarking may be hampered by relatively small volumes of transacting. Nile is an incredible supplementary dataset reflecting the performance of large-scale applications on public blockchains, such as scalability, transaction efficiency, and longevity of decentralized applications,

2.6 Blockchain Applications

Nowadays different applications of blockchain systems are found. Each type of application works in different cases but all of them have leveraged the decentralization of the blockchain systems to enhance transparency, and security in fields like finance, gaming, and digital asset ownership.

2.6.1 Decentralized Applications

Decentralized applications or dApps for short are software applications with a front-end that run on a blockchain network. The difference between centralized and decentralized applications is that Apps are those that work on a blockchain network but not on a centralized server. Based on the fact that blockchain is a decentralized network, they use it to ensure transparency, security, and user control. Smart contracts are the function execution vehicle of DApps, covering finance, gaming, and even social media.

2.6.2 Smart Contracts

A smart contract is a code-based deal, drawn up by several parties, enforcing these terms to facilitate a reliable and transparent means of exchange. They automatically impose and do actions based upon certain conditions that are pre-determined without intermediating [20]. Through increasing trust and efficiency, smart contracts achieve transparency, immutability, and verifiability of transactions.

2.6.3 Non-Fungible Tokens (NFTs):

NFTs are unique digital tokens representing the ownership of a specific item, for instance, or even an asset, on a blockchain. NFTs are nonfungible and indivisible as opposed to cryptocurrencies which are fungible and interchangeable [27]. The authentication and trade of digital assets, including art, collectibles, and virtual real estate, can be done by using them to prove authenticity and ownership.

2.7 Methods for Evaluating Blockchain Performance

Till now various seminal works have discussed and evaluated the performance of distributed ledger technologies like blockchain. As seen in Figure 2.2, the evaluation methods can be divided into empirical evaluation and analytical modeling. While empirical evaluation methods involve practical testing and observation, analytical modeling uses mathematical and theoretical techniques to evaluate performance, etc.

2.7.1 Simulation and Modeling

Simulating blockchain networks allows researchers and practitioners to study performance under different conditions without actually building anything. Modeling tools and simulation frameworks, like SimBlock and Blockchain Simulator, give insights into scalability, throughput, and latency under different situations [5]. These simulations are very important for checking if proposed blockchain designs will work well and for finding potential problems.

2.7.2 Real-world Testing

Real-world testing involves putting blockchain systems into use in real situations to see how well they perform. This gives valuable insights into the practical challenges

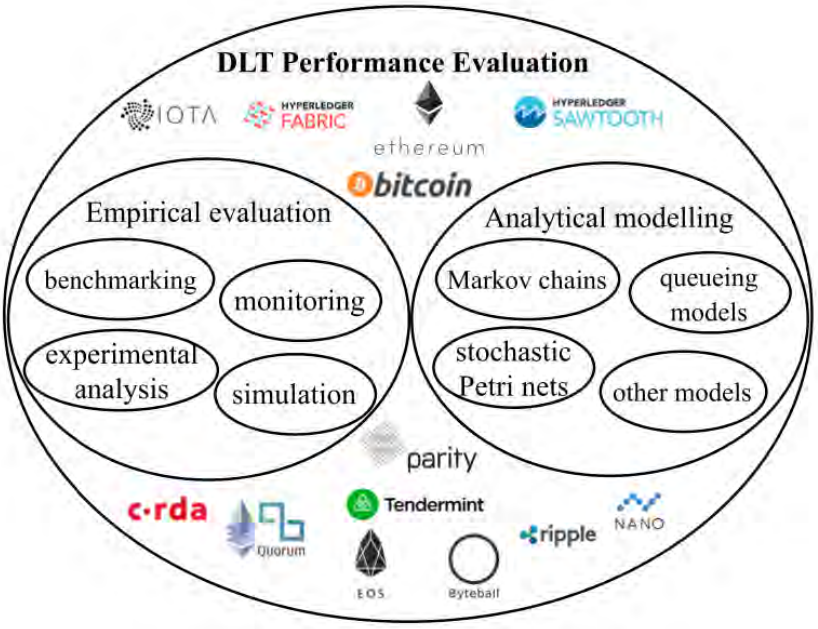


Figure 2.2: Approaches to DLT Performance Evaluation [15]

and benefits of using a blockchain system. Case studies of blockchain applications in different industries provide a lot of data for performance evaluation, allowing researchers to draw conclusions based on real experiences.

2.7.3 Benchmarking

Benchmarking involves comparing how well different blockchain systems perform using standardized measures and test situations. Well-known benchmarks, like the Hyperledger Caliper project, help to systematically evaluate key performance indicators across different blockchain platforms. Benchmarking is very important for making good decisions when choosing or designing a blockchain solution for specific uses.

2.8 Challenges of Public Blockchains

Before trying to evaluate public blockchain performance. We have to discuss the challenges we came upon that make it difficult to come up with proper evaluation criteria for measuring performance. It is necessary to tackle these challenges to develop a standardized protocol for evaluating blockchain performance.

2.8.1 Scalability vs. Decentralization

Vitalik Buterin co-founder of Ethereum, has given a clear insight into the scalability trilemma, defining scalability, security, and decentralization in public blockchains as difficult to balance [2]. Scalability allows high transaction throughput, decentralization stops single-entity control, and security determines data integrity. Most blockchains use decentralization and security first: Bitcoin, and Ethereum. This

all sets limited scalability. Bitcoin manages to process approximately 7 TPS, while Ethereum processes 15 TPS. Solutions such as sharding and second-layer protocols try to enable scalability with no compromise for decentralization, though much has to be done for the complete security and decentralization of the system.

2.8.2 Energy Consumption

Most public blockchains using the Proof of Work consensus algorithm are extremely energy-intensive owing to their computation-intense mining processes. Annual energy consumption by Bitcoin is estimated to be rivaling that of whole countries annually. Thus, other consensus mechanisms have been developed, including Proof of Stake. PoS, as used by Ethereum 2.0, is less energy-intensive since the validators are chosen by means of the cryptocurrency they have staked and not via mining, which is very energy-intensive [18]. While much more energy-efficient, PoS introduces a potential risk for centralization in general, where participants with greater wealth can have inordinate control over the means of validation.

2.8.3 Difference in Consensus Mechanisms

Consensus algorithms are used to determine how transactions on the blockchain are verified. Proof of Work, as used on Bitcoin and Ethereum before its upgrade, is secure but slow and energy-intensive. Proof of Stake, on which Ethereum 2.0 is based, improves the scalabilities and energy efficiency by choosing validators about the coins staked, although it risks centralization. Tron makes use of Delegated Proof of Stake, whereby high scalability is achieved at the sacrifice of some decentralization by the concentration of the validation power in an elite group of a few elected representatives. Each of them represents different consensus mechanisms that make different trade-offs between scalability, security, and decentralization.

PoS vs DPoS

If we compare the theories of PoS vs DPoS, we observe some differences in their performance and trade-offs. Generally, DPoS performs better than PoS in terms of transaction speed and overall response time because it uses a smaller, elected group of validators to produce blocks, which makes the process faster. On the other hand, PoS, while more efficient than Proof of Work (PoW), may not reach the same level of speed, especially in larger networks where more validators participate in the block validation process [25].

Decentralization: PoS tends to be more decentralized because it allows a larger number of validators to participate in staking and validation. DPoS, however, sacrifices some decentralization for efficiency, as it relies on a smaller group of elected delegates to maintain the network.

Security: PoS is generally considered more secure due to its broader validator set, making it harder for an attacker to gain control. In contrast, DPoS could be more vulnerable if the small number of elected delegates collude, although the option to re-elect delegates is intended to preserve trust.

Energy Consumption: Both PoS and DPoS are more energy-efficient than the proof of work mechanism because they don't require solving complex computational puzzles. DPoS may have a slight advantage in energy efficiency since it uses fewer active validators.

Trade-offs: The trade-off between these two consensus mechanisms depends on choosing between better decentralization with PoS and higher performance with DPoS. PoS may offer a higher level of security due to its larger validator base, while DPoS provides faster transactions but at the cost of some decentralization and potential risks if the elected delegates act improperly. Ultimately, the choice depends on what the user prioritizes, whether it is speed, security, or overall response flexibility.

Chapter 3

Literature Review

The performance evaluation of blockchain application systems is a major concern for blockchain developers. Since there are a lot of metrics and tools used in past performance evaluation protocols, it is tough to comprehend how a proper blockchain performance evaluation protocol will look and can easily be tested by the developers. Researchers and developers are trying to figure out how this problem can be solved. However, since there are varieties of use cases of blockchain application systems in various fields such as healthcare, finance, supply chain management, education, etc. it is difficult to establish a proper protocol for evaluating all the blockchain applications and testing their performance.

In [24], the authors analyze different blockchain consensus protocols such as Proof of Work (PoW), Proof of Stake (PoS), and Practical Byzantine Fault Tolerance (PBFT) in great detail. The research explores how mechanisms behind these achieve scale, security, and overall performance of large-scale blockchain systems. The authors discuss the tradeoffs between performance and decentralization: on the one hand, PoW offers strong security, but poor throughput and high latency; on the other hand, PoS is better scalable but more centralized. The main review we found in this research was the detailed comparison and analysis of the influence of consensus protocols on the performance of blockchain under different conditions. This sheds light upon factors such as network size and transaction volume, and how they impact these protocols, especially important for understanding scalability problems within public blockchain systems like Bitcoin and Ethereum. The research however omits other important elements that drive performance, including network architecture and resource management.

The authors in [11] highlight the rise of Bitcoin blockchain technology and the next generation of blockchain systems. How with the use of new technologies and network parameters blockchain systems became more efficient is discussed in the research. While a lot of consensus mechanisms are discussed, most blockchains still use the computationally intensive Proof of Work (PoW) mechanism. It also discusses the difference between the parameters and workloads between and public and private blockchain. The research also notes the lack of detailed studies on the relationship between performance and security in PoW blockchains. Within the study comprehensive evaluation of Ethereum, Parity, and Hyperledger blockchain systems are conducted with the help of the BlockBench evaluation framework which is normally

used for conducting experiments on private blockchain systems.

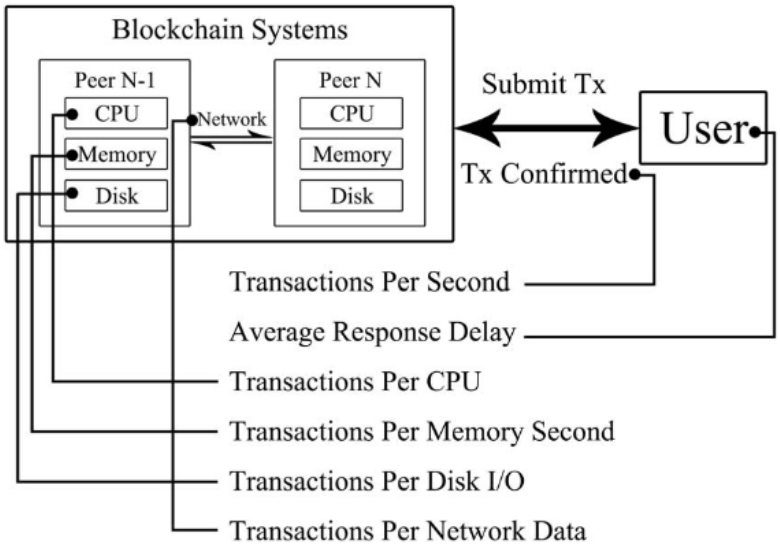


Figure 3.1: Various Metrics Corresponding to a Blockchain System [9]

In this research [9], the authors acknowledge the current performance evaluation systems of blockchain applications as poor and a major constraint. It also identifies the lack of a standard performance evaluation that can adapt to different types of blockchain systems. As a solution, the authors propose a performance monitoring framework that employs a log-based method. In Figure 3.1, the metrics used by the authors are shown. They used metrics such as transactions, average response delay, etc. to measure peers inside a blockchain. The results demonstrated the feasibility of the proposed framework and its ability to provide detailed and real-time performance monitoring. For future research, they suggested extending this work in several ways, including monitoring more blockchain systems and tracing blockchain systems using the log-based method.

Authors in [23] discuss the Hyperledger Fabric blockchain and how the applications based on this blockchain system work. In the experiments done in the thesis research, the authors used Hyperledger Fabric as the primary technology where the configurations are managed through Hyperledger Caliper. The experiments varied two main parameters, the input data rate and input standard deviation which is important to analyze for understanding the impact on write transaction latency.

Omar et al. deployed a smart contract in the Ethereum blockchain platform and tested the key functionalities that are required in a clinical trial system [17]. The study involved the development of getter functions for information retrieval, setter functions for writing data, and modifiers to restrict network member interactions. They also said that this solution to the clinical trial system can also be worked in other blockchain systems like Hyperledger or Quorum.

In [15], the authors categorize performance metrics of blockchain systems into macro and micro metrics. As shown in Figure 3.2, macro metrics provide an overall view of

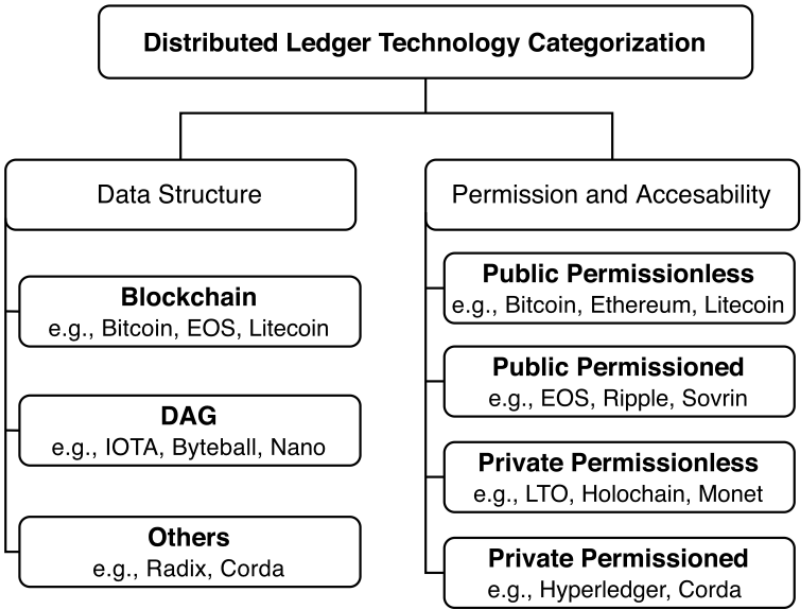


Figure 3.2: Categories of distributed ledger technologies [15]

the system's performance at the application level, focusing on transaction throughput, latency, scalability, fault tolerance, and transactions per unit of computing resources. Micro metrics identify specific processes of blockchain technology, such as peer discovery rate, RPC response rate, and encryption efficiency. The research emphasizes empirical analysis and analytical modeling for a better understanding of how a blockchain system and its applications work. The research suggests future research areas for improving performance evaluation, including developing standard interfaces for workload uploading in blockchain platforms.

Authors in [21] review state-of-the-art theoretical models for blockchain systems. They have covered various insights into various blockchain protocols, and consensus mechanisms. They also emphasized the importance of performance measurements for blockchain applications. In doing so, the authors identify a gap in the existing literature, noting the lack of focused surveys on performance measurements, datasets, and experimental tools for blockchain technology. They identified challenges in blockchain-based applications regarding centralized security mechanisms, including the risk of single-point failure. They also termed scalability and interoperability as significant challenges to overcome and urged further research regarding evaluating these metrics. The survey aims to fill this gap and provide guidance for future blockchain research and development.

Dong et al. have standardized a performance evaluation framework for various DAG implementations [10]. As shown in Figure 3.3, we see the protocol flow of the Dagbench engine. The flow starts by loading network and workload configuration files. The DAGBENCH engine then initializes the DAG network, followed by initializing the workload. It runs the workload, generates a report, and finalizes the DAG network. The framework uses several performance metrics for evaluation such as

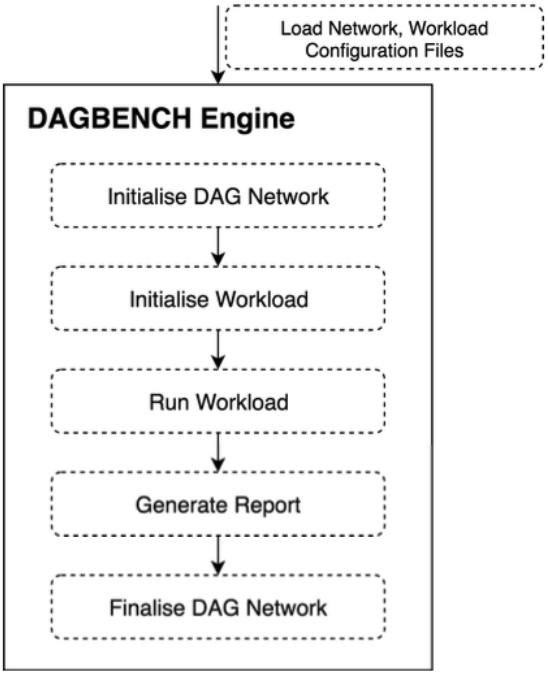


Figure 3.3: DAGBENCH engine [10]

throughput, latency, scalability, success indicator, resource consumption, transaction data size, and transaction fee. It compares popular DAG implementations like IOTA, Nano, and Byteball. It also discusses a need for a tool that will make it easier to compare and analyze different DAG implementations under the performed metrics. The framework was evaluated in real-time by Amazon EC2 for the mentioned DAG implementations. The results helped identify the advantages and disadvantages of various DAG implementations.

In this research [22], the modern usages of blockchain systems and applications are discussed. With the evolution of blockchain systems, many more important parameters and workloads are required to properly analyze the blockchain systems. The research highlights the rise of modern blockchains, which are environmentally sustainable and capable of advanced functionality, such as smart contracts that are based on the Ethereum blockchain system. The statistically significant shift from Proof of Work (PoW) protocol to environment and resource-friendly Proof of Stake (PoS) in the updated blockchains. The researchers took into account the parameters of consensus protocol efficiency, transaction throughput, etc for properly identifying the benefits of using a Proof of Stake (PoS) system.

In [14], the authors compare the Hyperledger Fabric and Ethereum using different metrics. According to the author, Hyperledger Fabric outperformed Ethereum across various performance metrics. A significant difference in memory consumption between Ethereum and Hyperledger Fabric was observed, with Ethereum consuming considerably more memory. The authors note the scarcity of research studies that thoroughly evaluate and compare the performance of different blockchain platforms. Metrics such as success rate, latency, resource consumption, and throughput are observed during the experiments.

Alom et al. fills the gap for standardized performance-assessing tools specifically for private blockchains which are widely used in the finance and supply chain management industry [30]. The key strength of this framework is its application-agnostic nature, and therefore it can be deployed across different blockchain platforms without any application-specific modification. It evaluates critical performance metrics (such as transaction latency, throughput, and resource consumption) to provide qualitative insight into which configuration of the underlying blockchain optimizes performance. The authors validate the practicality of BlockMeter on several private blockchain platforms, showing how it can make useful comparisons between different systems.

The research [16] shows a detailed comparison between performance metrics, such as throughput, latency, and scalability between public and private blockchain platforms. The importance of understanding these dynamics is highlighted: especially in the fields of finance and supply chain management. The author argues that there are trade-offs between decentralization and performance, while public blockchains like Bitcoin and Ethereum prioritize this decentralization at the expense of speed and energy, for example with the consensus mechanism proof of work (PoW). For instance, Private blockchains such as Hyperledger Fabric are aimed at performance by using more refined consensus protocols like Practical Byzantine Fault Tolerance (PBFT). Key contributions of the research include a detailed comparison of public and private blockchains and exploring solutions including sharding and sidechains as means to scale public systems. Nevertheless, the theoretical analysis is not backed with empirical data, and the research is not deep into the hybrid blockchain model. However, these limitations give you valuable insights into the performance of public blockchains and offer potential solutions to your work in the development of a performance evaluation framework of public blockchain systems.

In this research [29], a comprehensive evaluation of the tools and frameworks for dApp development on the Ethereum platform is presented. The thesis compares prominent development environments (Hardhat, Truffle, Remix) and analyzes their usability, development support, and performance. Based on factors of setup complexity, testing features, EVM compatibility, and debugging capabilities, the authors explore these environments. There is also a focus on how these tools help developers maximize smart contract performance with gas fees and deployment on the Ethereum mainnet. One important contribution of their research is the detailed analysis of what these environments can do in different circumstances such as deploying smart contracts and interacting with the decentralized finance (DeFi) protocols. Hardhat's plugin ecosystem famed for its amazing flexibility in testing and development takes the position as a developer's tool of choice when one needs customizability. From a simple standpoint, Hardhat has these, however, Truffle's structured framework is recognized. In terms of performance testing with Ethereum and dApp development, the value this thesis provides is especially valuable for those using Hardhat as it helps developers understand on what its strengths lie and how they can improve their own ability to do the same.

In [31], authors explore how Ganache, the development tool of choice for local

Ethereum blockchains, can be expanded. Ganache is intended to help developers test decentralized applications (dApps), smart contracts and transaction flows in a controlled environment quickly. Ganache's importance for private Ethereum blockchains is the main analysis of the research, with a special focus on how it allows ease of data storage, efficiency, and security, along with fast development and testing. For developers and researchers working with private Ethereum blockchains, this research is relevant in particular because it touches on ways to improve smart contract testing and performance evaluation with Ganache. As a product, it is highly suitable for dApp and smart contract developers who need to ensure security and efficiency.

Our research is trying to solve this problem by including the most major and crucial parameters, metrics, and workloads that are needed to scrutinize the performance of a blockchain application system properly. Of course, this research is only possible with past research works, surveys, and performance evaluations done by experienced researchers in the blockchain field. The protocol we are trying to standardize will be possible only by previous research on performance evaluation systems.

Chapter 4

Research Proposal

4.1 Proposal

This chapter outlines the methods we used to address the challenges identified in our research problem. It also reviews common findings and contradictory views of previous works. Our goal is to develop an effective framework that can accurately measure and compare the performance of public blockchains.

We aim to create a system that allows developers to easily assess and compare their blockchain applications on various platforms like Ethereum and Tron without facing the current complexities. For getting our benchmarks, we have used testnets of the blockchain systems. Since the testnets replicate the behavior of the mainnet we can get the best results without any risk.

By using this framework, developers can measure important performance metrics such as transaction per second, wallet consumption, and overall response time consistently across different blockchain systems. This helps them choose the best public blockchain platform for their needs and optimize their applications for better efficiency and cost-effectiveness.

To implement the proposed framework, we perform an analysis of metric-wise findings, hardware and setup findings, and implementation-wise findings of our literature review. This will help us build a framework that addresses the current limitations in public blockchain performance evaluation.

4.2 Research Methodology

We have sequentially proceeded with our research methodology. From the start of our methodology, we followed a structured approach to benchmark the performance of public blockchain-based systems like Ethereum and Tron. In Figure 4.1, a flow diagram of our methodology is given. This illustrates our whole research from problem identification to result analysis.

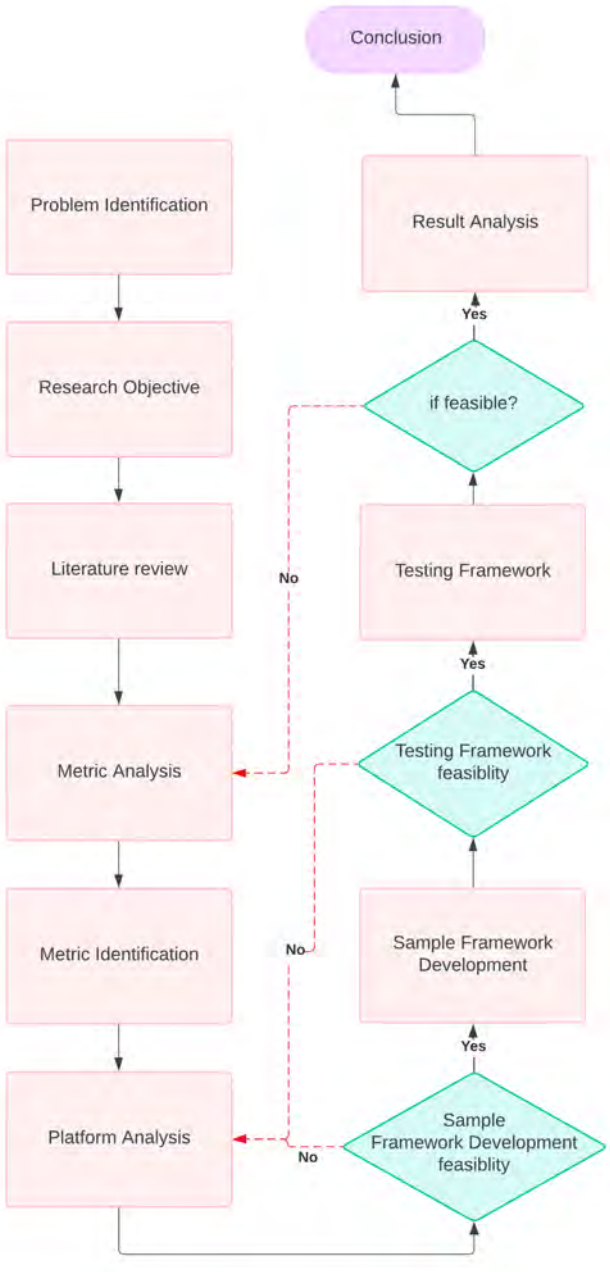


Figure 4.1: Research Methodology Flowchart

4.2.1 Problem Identification

The main problem we identified in our thesis research is that the current blockchain performance evaluation lacks a unified and standardized approach. In this stage, we scrutinized past research works to highlight how we will implement, develop, and deploy our framework. This step ensured we were tackling a critical gap in blockchain performance evaluation.

4.2.2 Research Objective

Our main research objective is to create a performance framework using key evaluation metrics for benchmarking blockchain applications in blockchain systems. A framework that is also user-friendly. Then we ensure that applications deployed on public blockchains can be compared effectively against each other.

4.2.3 Literature Review

We analyzed past research works to identify what evaluation metrics, experimental setups, and deployment processes have been used. This review informed our approach to metric selection and testing, ensuring that we built on proven methods while addressing identified shortcomings.

4.2.4 Metric Analysis

Metric analysis is crucial for making our evaluation impactful. We analyze metrics across different research works and divide the primary and secondary metrics. These metrics are key to evaluating the performance of blockchain platforms in a way that is both meaningful and comparable.

4.2.5 Metric Identification

By properly identifying our primary metrics, we were able to determine which metrics we require for our evaluation framework. This step focused on narrowing down the most relevant metrics for blockchain performance, ensuring accuracy and efficiency in the evaluation.

4.2.6 Platform Analysis

We had to select which blockchain platforms we were going to use for evaluating our framework. Public blockchains incorporating different consensus mechanisms were analyzed in this phase based on their ability to support smart contracts and their robust testnet environments, which allowed for thorough performance testing.

4.2.7 Sample Framework Development

We used different experimental setups in both local and online environments. When we encountered issues in setting up initial development, we returned to platform analysis to adjust the framework. This iterative process helped refine our framework until it was ready for comprehensive testing.

4.2.8 Testing Framework

After creating a successful framework using the selected primary metrics, we tested it by deploying smart contracts on the testnet. Before that, we had to understand if the framework was feasible with the selected blockchain platforms, and if not, we returned to platform analysis for adjustments.

4.2.9 Result Analysis

If the testing was feasible, we used our faucets and testnet wallet addresses to generate data for result analysis. This data was essential for assessing the framework's effectiveness and drawing conclusions about the performance of the public blockchain platforms under test.

4.3 Metric Wise Findings

In the past works for developing standard evaluation metrics, the researchers have used many different metrics for analyzing performance. Some of these metrics are important while some does not make sense. We try to find common findings of the evaluation metrics that are observed across many different research. We also give our contradictory view on the evaluation metrics that we do not find useful for building our framework.

4.3.1 Common Findings

Across multiple papers, there is a common ground on the significance of certain transaction metrics, such as Transactions Per Second (TPS), Transactions Per CPU (TPC), and Transactions Per Memory Second (TPMS), indicating a shared emphasis on scalability, CPU efficiency, and memory management [4][7][9][11][12][13][19]. Importance has also been given to the Security Metrics (e.g., 51% Attack Resistance) as they are vital for maintaining the integrity of the network and ensuring robustness against potential attacks.

4.3.2 Contradictory View

While Alsahan et al. (2020) and Yang et al. (2019) prioritize TPS due to its fundamental representation of transaction processing capacity, Wang and Wang (2019) emphasize TPC for efficient CPU usage, showcasing varied perspectives on the primary transaction metric [11][12][13].

4.4 Hardware and Setup Findings

For building our framework, we have researched the hardware and experimental setup of previous works. While most of the findings are common we also found some contradictions.

4.4.1 Common Findings

The majority of past research done has consensus with the specifics such as CPU, memory, virtual machines, Docker containers, cryptography libraries, and simulation frameworks.

4.4.2 Contradictory View

However, there are differences in the scale and specifications of the hardware used across papers. For instance, varying numbers of virtual machines, and different CPU configurations. Moreover, differences in simulation frameworks OMNeT++ in [19], and Docker containers in [13]. Also in [11] use of 1,200 virtual machines with 8 cores and 32GB memory, while in [12] experiments on Amazon EC2's c5d.4xlarge instances with 16 cores and 16GB RAM. In [19] employs OMNeT++ for simulation, whereas [13] uses Docker containers for simulating a local blockchain network.

The diversity in hardware specifications and simulation tools reflects various approaches to experimentation, most likely impacting the validity of results.

4.5 Implementation wise findings

As frameworks can be developed in various ways, we analyzed past research to find which implementations have the best use cases and share our contradictory view on the ones that have limitations.

4.5.1 Common Findings

The implementation of consensus mechanisms and smart contract execution is consistently identified as crucial for blockchain performance across the examined papers [11][12][13]. Besides, Wang has emphasized cryptographic strength as it underpins blockchain security. The strength and appropriateness of cryptographic algorithms directly impact the system's resistance to attacks and safeguarding of user assets.

4.5.2 Contradictory View

Wang et al. (2019) highlight the importance of Peer Discovery Implementation, arguing that the specifics of how peer discovery is implemented may impact public blockchains differently. However, Yang et al. (2019) suggest that Peer Discovery Implementation details may be less crucial in the broader context of decentralized networks, revealing varying perspectives on the relevance of specific implementation aspects [11][12].

4.6 Experimental Design

How the experiments are done and what results are analyzed in the past works are important for building our framework design.

4.6.1 Common Findings

Overall throughput experiments, covering various transaction metrics, are commonly employed to provide a holistic view of blockchain performance [4][9][11][12]. Mention of network parameters, bandwidth restrictions, latency measurements, and the overall configuration.

4.6.2 Contradictory View

While Alsahan et al. (2020), Yang et al. (2019), and Woo et al. (2020) conduct experiments focusing on peer discovery rate, Wang et al. (2019) suggest that such experiments might be less critical due to the dynamic nature of peer connections in public blockchains, introducing diverging perspectives on the necessity of specific experimental focuses [11][12][13][19].

4.7 Evaluation Metrics

After a rigorous analysis of all the research works in our literature review we have decided to divide all the mentioned metrics into Primary and Secondary evaluation metrics.

4.7.1 Primary Evaluation Metrics

Primary metrics are the ones that we found to be impactful for a general evaluation and benchmark.

Overall Response Time

Efficient CPU usage is vital for responsiveness in public blockchains; high CPU usage can lead to delays in transaction processing.

Transactions Per Second (TPS)

TPS is fundamental, representing the blockchain's transaction processing capacity, crucial for scalability and overall performance.

Wallet Consumption

Wallet consumption quantifies the expenditure of users in spending to execute transactions on the blockchain, being the ultimate point of reference to evaluate the cost efficiency and economic utility of the network.

4.7.2 Secondary Evaluation Metrics

Secondary evaluation metrics are the ones that seemed to have little to no impact on our evaluation.

Individual Transaction Confirmation Time

While individual transaction confirmation time is relevant, the overall throughput and scalability of the blockchain are crucial for assessing its performance on a larger scale.

CPU Utilization

While individual transaction confirmation time is relevant, the overall throughput and scalability of the blockchain are crucial for assessing its performance on a larger scale.

RPC Response Rate (RRR)

In the context of public blockchains, where consensus and transaction processing are paramount, the rate of responses to external RPC calls may be secondary.

Transaction Propagating Rate (TPR)

While important, the specific rate of transaction propagation may be less critical compared to overall throughput and consensus efficiency.

Single Address Hotspots

While identifying single-address hotspots can be insightful for optimization, it might not be a crucial metric for overall network performance. Hotspots can often be addressed through specific optimizations without significantly impacting the entire network.

In our research, we propose the development of a standardized performance evaluation framework for public blockchain systems. For getting our benchmarks, we have used testnets of the blockchain systems. Since the testnets replicate the mainnet functionalities, we will deploy various smart contracts that represent different types of workloads to measure key performance metrics such as Transactions Per Second (TPS), transaction latency, wallet consumption, and resource utilization. The aim is to identify and compare the strengths and weaknesses of each blockchain platform, providing developers with consistent and reliable methods to assess and optimize their decentralized applications (dApps). By addressing the current lack of uniform evaluation protocols, this framework seeks to enhance the efficiency and effectiveness of blockchain application deployment, ultimately contributing to the advancement and sustainability of public blockchain technologies.

Chapter 5

System Model and Architecture

5.1 Evaluation Metrics

After rigorous research, the parameters that we are considering to be the top prioritized parameters for evaluating public blockchain application performance are as follows:

1. **Wallet Consumption:** Wallet consumption quantifies the expenditure of users in spending to execute transactions on the blockchain, being the ultimate point of reference to evaluate the cost efficiency and economic utility of the network.

2. **Overall Response Time:** Latency of local processing network, network delay, pending pool delay which is the total addition of time from transaction pool to miner, and after the data is mined the time related to getting the data registered in the latest block.

3. **Transaction Per Second** TPS is fundamental, representing the blockchain's transaction processing capacity, crucial for scalability and overall performance. Below is the formula of how Transaction per second is calculated.

$$\text{TPS} = \frac{(\text{Total number of Transactions})}{\text{Time taken}}$$

4. **Resource Utilization:** Resource utilization measures how efficiently the blockchain network uses its computing resources, such as CPU, memory, and storage. It is important to understand the system's performance under different workloads and ensure optimal resource allocation.

According to our research, these parameters are crucial for evaluating public blockchain application performance because they cover key aspects that determine the network's effectiveness and reliability. Throughput latency and transaction latency provide insights into the network's speed and responsiveness, which are essential for processing a large number of transactions efficiently. Resource utilization is important for understanding how well the network uses its computing resources, ensuring optimal performance under varying workloads.

5.2 Software Architecture

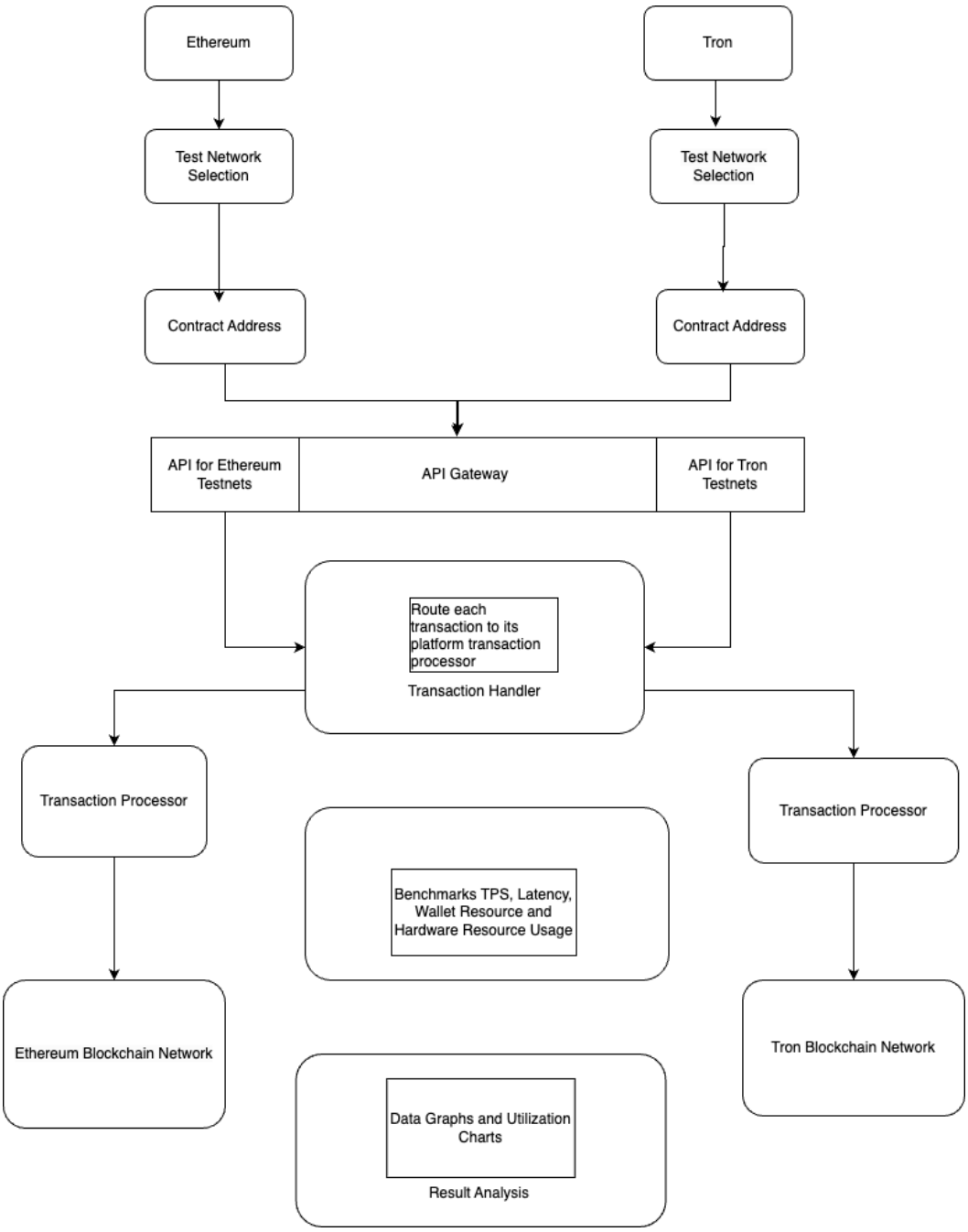


Figure 5.1: Software Architecture

Here we are discussing our software architecture and the explanation of each part of the software architecture is as follows:

- **Blockchain Application:** This system involves two blockchain platforms: Ethereum and Tron. Each of these DApp runs on its corresponding platform. Depending on the requirements the client application selects Ethereum or Tron. The most important part of the test network selection stage is choosing the test network from an extensive list of the test networks (e.g., Holesky for

Ethereum and Shasta for Tron) before interacting with the blockchain. Once this has been done, the user selects whether their contract address should be generated or entered, and it is then generated or inputted into the user.

- **API Gateway:** Between the external client application and the Ethereum and Tron internal processes, the API Gateway works as a Middleware. It includes two distinct APIs for Ethereum and Tron. Each of these APIs represents the routing of incoming transactions with the API Gateway to its respective Transaction Processor. This is where the API gateway comes in: It acts as the front end that receives the request and forwards the Ethereum-specific and Tron-specific requests to the right chain platform.

- **Transaction Handler:** Each transaction is routed from the Transaction Handler to one of the Platform-specific Transaction Processors. The Transaction Handler handles client requests (smart contract execution, token transfer, etc.) using the same objects on the blockchain as client requests translate to (such as transactions). This guarantees that every request works on the relevant blockchain's hardware and setting.

- **Transaction Processor:** We have one service per blockchain (Ethereum + Tron) for transaction processors. These processors take the form of middleware components that interpret the client's requests and work with the Ethereum Blockchain Network or the Tron Blockchain Network. The data is prepared in a way to be processed by the underlying blockchain architecture.

- **Performance Monitor:** Once the transaction is processed, the Performance Monitor is used extensively in measuring the critical performance metrics. This includes:

  1. Transaction Throughput (TPS): Number of transactions per second.
  2. Overall Response Time: The time taken to confirm a transaction.
  3. Wallet Resource Usage: Quantity of resources consumed by the user's wallet – e.g. gas fees on Ethereum and Trx and energy consumption in Tron.
  4. Hardware Resource Usage: It tracks how much CPU usage, memory, and network bandwidth the system resources use during transaction processing, shoes.

- **Result Analysis:** Result Analysis visualizes the data of gathered performance metrics as Data Graphs and Utilization Charts. They make the performance of your transactions easily understandable as users can observe this interpreted visually between Ethereum and Tron on various metrics. Like BlockMeter, this module helps us understand how different blockchains behave under different workloads, measuring how much data can be loaded and stored for the platform in the given period and providing extra analysis based on the criteria chosen by the end user.

- **Blockchain Network:** At the backend of the system lies the two blockchain networks:

– Ethereum Blockchain Network

– Tron Blockchain Network

Smart contracts represent their business logic through these networks, called decentralized applications. The contracts are executed, transactions are validated and the consensus is maintained by the networks.

5.3 Protocol Flow

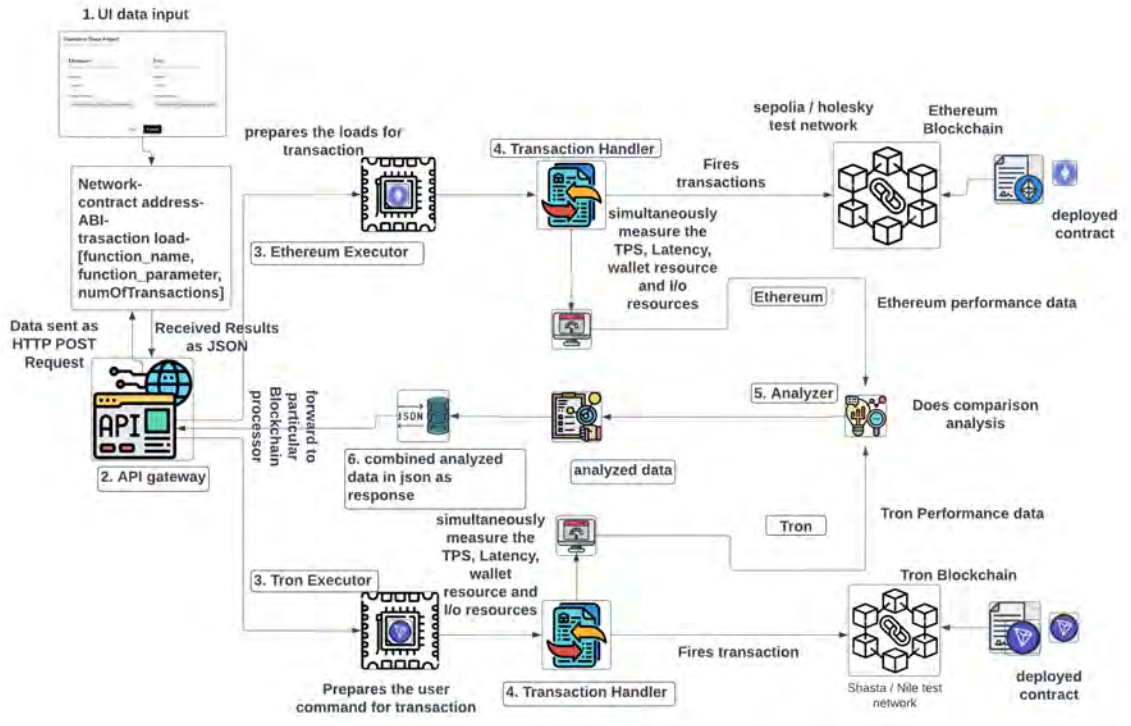


Figure 5.2: Protocol Flow

Our framework contains a protocol flow that shows a system to interact with both Ethereum and Tron platforms and measures and analyzes their ability in transaction processing. Here's a breakdown of the key components of the protocol flow:

1. **UI Data Input:** Users will see the UI of our frontend where they can define various parameters, such as network selection, smart contracts address, function name, parameters, and the number of transactions to execute.

2. **API Gateway:** The framework has API for establishing a connection to the backend and passing the input data to the different blockchain executers.

3. **Executor:** Then the system will forward the prepared transactions to a specific executor that will send the transactions to their specific networks like Ethereum or Tron. The Sepolia/Holesky networks are being used by Ethereum and the Shasta/Nile networks by Tron.

4. **Transaction Handler:** The system will prepare the transaction loads based on the provided parameters from the executor.

5. **Analyzer:** During the transaction execution, the system measures key performance metrics simultaneously.

   (a) TPS (Transactions per Second)

   (b) Overall Response Time

   (c) Wallet resource consumption

   The framework ends by evaluating the benchmarked performance between Ethereum and Tron and compares analyzed data between the two.

6. **API Response:** After analyzing data the metrics are stored in JSON format and returned as API response.

5.4 Implementation

In our research work, we used nodejs express server for implementing our backend. For connecting with different testnet, ethers, and tronweb npm package is used. Testnets are provided by Alchemy provider for Ethereum and Tron we are using Trongrid provider. Our frontend is written in React Js library and we used the Axios library to connect our frontend with the backend. In our backend, we have connected APIs of different test networks. As shown in Figure 5.3, our frontend takes inputs of wallet addresses, ABIs, and the option for choosing testnets for Ethereum and Tron. After proceeding with the given information from the user, the information is passed to our backend server. The server works as a middleware for connecting with the test networks.

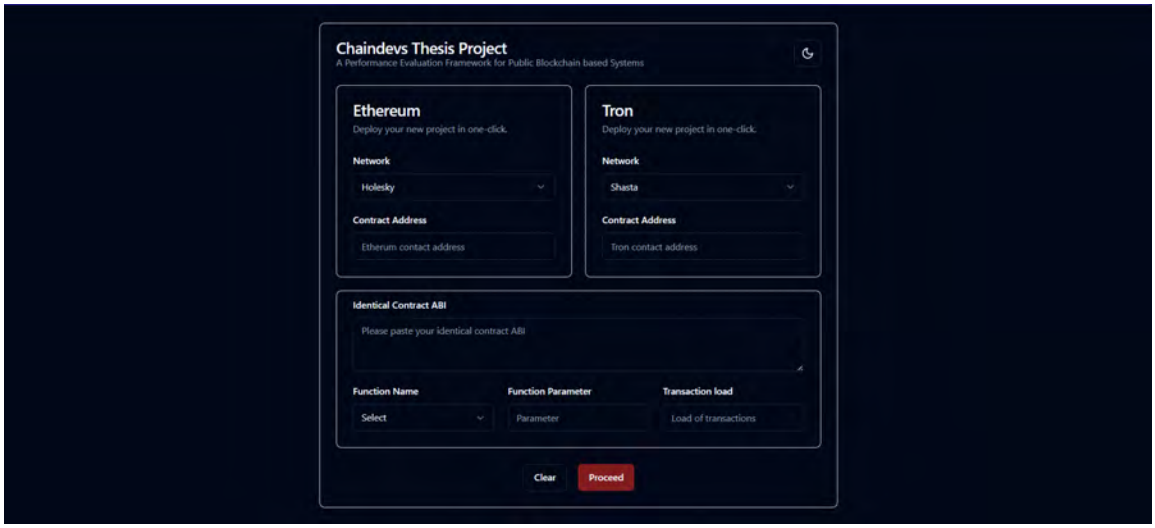
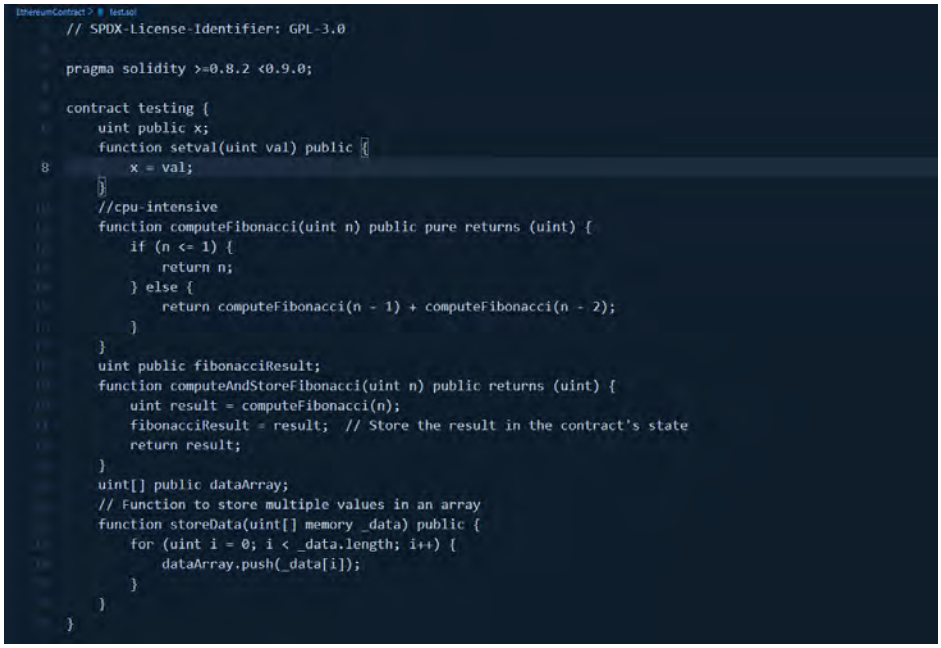


Figure 5.3: Framework User Interface

For Ethereum, we have tested both locally and in test networks. The reason for benchmarking Ethereum both locally and in testnets is that, in testnets, the data

of the metrics did not give a proper conclusion that could be reached. The local tests are not connected to our system. However, implementing a local environment for Ethereum was important in understanding the discrepancies of test networks that are discussed further in the result analysis. The local implementation was done using Hardhat. Even though hardhat itself is a framework for measuring performance, we used it for collecting data which would give us insights that the testnet is performing poorly.



```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract testing {
    uint public x;
    function setval(uint val) public {
        x = val;
    }
    //cpu-intensive
    function computeFibonacci(uint n) public pure returns (uint) {
        if (n <= 1) {
            return n;
        } else {
            return computeFibonacci(n - 1) + computeFibonacci(n - 2);
        }
    }
    uint public fibonacciResult;
    function computeAndStoreFibonacci(uint n) public returns (uint) {
        uint result = computeFibonacci(n);
        fibonacciResult = result;  // Store the result in the contract's state
        return result;
    }
    uint[] public dataArray;
    // Function to store multiple values in an array
    function storeData(uint[] memory _data) public {
        for (uint i = 0; i < _data.length; i++) {
            dataArray.push(_data[i]);
        }
    }
}
```

Figure 5.4: Example Smart Contract

In the case of Tron, we found the expected results after analyzing the benchmarked data derived from the test networks. For that reason, we still did not perform any local analysis for Tron. We tested our experiment based on the smart contract shown in Figure 5.4. Here we implemented three types of functions and collected our data based on this smart contract.

Chapter 6

Experiments

6.1 Experimental Setup

For our experimental setup, we tested on test networks such as Sepolia and Holesky for Ethereum, and for Tron, we used the Shasta and Nile test networks. Our main goal was to analyze performance in terms of throughput, overall response time, wallet consumption, and resource utilization.

For the experiments, we ran our scripts using the APIs of the test networks. The generated logs provided information on the metrics we used. In each procedure, we increased loads to generate data. However, to show a comparison between the two blockchain platforms, we conducted versatile tests by deploying simple, CPU-heavy, and data-heavy functions.

Simple Function: This is a straightforward function that allows users to set the value of a public variable. It represents typical read/write operations, where a simple value is stored or updated on the blockchain. The time complexity of this simple function is O(1).

CPU Heavy Function: This function performs CPU-intensive calculations by computing the Fibonacci sequence, which involves a recursive operation with increasing complexity as the input grows. It simulates high computational workloads within blockchain transactions, making it ideal for testing CPU performance in a smart contract. The time complexity of this function is $O(2^n)$.

Data Heavy Function: This function is designed to handle large amounts of data, storing multiple values in an array on the blockchain. It represents a data-heavy operation, commonly seen in decentralized storage applications, where multiple records or large data sets are written to the blockchain. The time complexity of this function is O(n).

The CPU heavy function has exponential time complexity due to recursion, while the data-heavy function scales linearly with the size of the input data. The simple function is a constant-time operation. Each procedure involved running each load on the test networks.

For the whole framework, javascripts nodejs runtime is used and for interacting with the Ethereum blockchain, the ethers.js library was used. This library facilitates the creation of wallets, sending of transactions, and interaction with smart contracts. On the Tron blockchain, the TronWeb library was utilized to interact with the Tron network, send transactions, and call contract functions.

Resource utilization (CPU and memory) was monitored using the pidusage library, which provides detailed statistics on process resource consumption before and after transaction execution.

The built-in fs module was used to log key performance metrics such as overall response time, transaction gas/energy usage, and system resource usage to local text files for post-experiment analysis. The API keys, wallet addresses, etc., were kept in a dotenv file. For different scenarios and different simultaneous cycles, the rate of incoming traffic or the load varied steadily.

For all the loads, we started with 5 transactions and incremented by 5 in each subsequent procedure. At the end of the experiments, the data recorded by the performance monitor was represented using graphs to facilitate comprehensive performance analysis. We tested locally to generate the evaluation metrics using both single and parallel processing.For the simple function smart contracts, a single state variable updates with every call.

6.2 Result Analysis from Data and Graphs

After generating data for around 50 loads for all the functions, we managed to get proper data for our result analysis, we created line graphs for Overall Response Time vs Load and Transaction per second vs Load. For our currency vs usage graphs, we also tried to generate similar loads for these data, but they had some discrepancies that are analyzed in the respective sections.

6.2.1 Testnet Result Analysis

For Simple Function

Load	Overall Response Time (s)	TPS	Gas Price (wei)	Gas Required	Ether Used
5	19.2349	0.2274	6,894,075,239	26,640	0.00016435
10	29.2168	0.2850	7,599,142,858	23,928	0.00018116
15	19.2072	0.6410	6,707,188,732	26,652	0.00015998
20	34.4267	0.4598	5,980,143,094	23,940	0.00014264
25	19.0941	0.8602	6,583,817,864	26,640	0.00015696
30	35.9344	0.7969	5,530,678,117	26,640	0.00013198
35	27.3125	0.9089	1,777,731,438	26,652	0.00004736
40	47.3843	0.8453	6,165,049,123	23,928	0.00018923
45	52.9146	0.9381	7,132,041,374	29,238	0.00021137
50	41.3823	1.0264	7,549,023,172	26,640	0.00023498

Table 6.1: Ethereum Testnet Benchmark for Simple Function Smart Contracts

Load	Overall Response Time	TPS	Trx Used	Energy Need	Bandwidth
5	1.6666	2.644103649	2.61796	5488	313
10	1.6022	5.685043823	2.61796	5488	313
15	1.7021	7.56429652	2.61796	5488	313
20	3.012	4.389815628	2.61796	5488	313
25	1.78712	9.255831174	2.61796	5488	313
30	1.1912	16.12036539	2.61796	5488	313
35	0.7915142857	18.786930284	2.61796	5488	313
40	1.229475	17.37619461	2.61796	5488	313
45	0.7901555556	23.35236118	2.61796	5488	313
50	0.82284	31.78639542	2.61796	5488	313

Table 6.2: Tron Testnet benchmark for Simple Function Smart Contracts

Simple function smart contracts on Ethereum and Tron testnets have distinct performance metrics differences in scalability and efficiency. Response times on the Ethereum Testnet shown in Table 6.1 sit within a 19–53 second range for an increase in load from 5 to 50 TPS and increase modestly from 0.2274 TPS to 1.0264 TPS. This shows that gas prices are wildly variable with values between 5.53 billion to 7.6 billion wei, and Ether usage jumps from 0.00004736 to 0.00023498 Ether per transaction, indicating variable transaction costs and resource needs with more people running big clusters. Such response time and lack of TPS growth indicate that Ethereum will have to contend with scaling problems as transaction volumes increase.

On the contrary, the Tron Testnet shown in Table 6.2 performs much better under the same load. TPS scales robustly at high load from 5 to 50 with a low increase (94.4x) in response times from 0.79 to 3.012 seconds. No matter the load, the Tron has no resource consumption (Trx usage fixed at 2.61796 Trx, energy needs at 5488 units, bandwidth usage at 313 units). With high and scalable TPS, this predictability of resource usage combined with Tron's ability to handle high transaction volumes at high efficiency makes it superior. As a result, Tron provides a more economical and viable alternative for transaction processing for applications requiring quick and congruent transactions, over Ethereum.

For CPU Heavy Function

Load	Overall Response Time	TPS	Gas Price	Gas Required	Ether Used
5	27.3323075	0.1082671267	6076592388	7652788	0.04650287675
10	146.7955172	0.03305937924	5333969099	7652788	0.040806105977
15	520.2639111	0.00999475082	5126885107	7652788	0.03923496482
20	587.4347528	0.01987432387	5974055778	7652788	0.0457223914
25	632.943211	0.014529467	4827465312	7652788	0.03823458123
30	751.432928	0.018762394	5731486132	7652788	0.04410987612
35	842.681234	0.024381748	6023145821	7652788	0.04823109127
40	925.473182	0.029302743	6523418973	7652788	0.05243172871
45	1014.12391	0.032785432	7156230451	7652788	0.05687458932
50	1105.29481	0.038123123	7698124317	7652788	0.06123471235

Table 6.3: Ethereum Testnet benchmark for CPU Heavy Smart Contracts

Scalability and efficiency are quite different between the performance metrics of CPU-heavy smart contracts on Ethereum and Tron testnets. As the load becomes

Load	Overall Response Time	TPS	Trx Used	Energy Need	Bandwidth
5	0.693	2.912055911	150.31264	357142	313
10	1.9361	3.365870077	150.31264	357142	313
15	1.8524	5.613772455	150.31264	357142	313
20	1.4251	7.648183556	150.31264	357142	313
25	1.15008	8.164598302	150.31264	357142	313
30	1.6530	18.14882033	150.31264	357142	313
35	1.7320	20.20785219	150.31264	357142	313
40	1.7740	22.49718785	150.31264	357142	313
45	1.8860	23.00054584	150.31264	357142	313
50	1.9235	23.40057097	150.31264	357142	313

Table 6.4: Tron Testnet benchmark for CPU Heavy Smart Contracts

higher shown in Table 6.3, TPS (Transactions Per Second) decreases, and Overall Response Time (in seconds) changes dramatically, moving from 27.33 seconds at load 5 to 1,105.29 seconds at load 50, while from this time point, Response Time increases rapidly as well, as more nodes will block their actions, resulting a sharp increase in Overall Response Time. The Gas Price fluctuates between 5,130,000,000 to 7,700,000,000 wei, yet Ethereum Used is increasing by only 0.0157 in this decrease in Gas Required even to 7,652,788. This shows that the scalability of Ethereum under the CPU intensive task will take longer to process and lower transaction throughput.

However, the Tron Testnet as shown in Table 6.4 exhibits brilliant performance under similar CPU-heavy loads. Overall Response Time measures between 0.693 and 1.9235 seconds, while TPS increases steadily from 2.91 to 23.40 as the 5—50 load from 5 to 50 grew. Constant across all load levels are resource metrics, including Trx Used (150.31 Trx), Energy Need (357,142 units), and Bandwidth (313 units). Thus, Tron's constant consumption behaviors around resources and minimal variation of response time account for its outstanding ability to handle CPU-heavy smart contracts. Compared to Ethereum, Tron is a more reliable and scalable platform for the deployment of CPU-intensive smart contracts.

For Data Heavy Function

Load	Overall Response Time	TPS	Gas Price	Gas Required	Ether Used
5	16.80925496	0.2000874118	6339062329	273126	0.001622964772
10	104.9961814	0.05480833189	434708140	256026	0.001112110222
15	201.4128391	0.032813471	5104285732	240325	0.001054872312
20	312.3156782	0.045712389	5673428174	273126	0.001312342157
25	404.5128371	0.059123456	6287454124	289734	0.001523872813
30	502.7314123	0.073421283	6541827312	302178	0.001731412392
35	623.8231912	0.091723812	7102831942	273126	0.001964823712
40	731.4132813	0.112312381	7658273412	289734	0.002193741237
45	845.6782931	0.123872391	8142738123	302178	0.002321234189
50	923.1837193	0.134123821	8712481732	273126	0.002531789134

Table 6.5: Ethereum Testnet benchmark for Data Heavy Smart Contracts

The operational efficiencies of data-heavy smart contracts on Ethereum and Tron testnets are seen in the performance metrics. However, in regards to the Overall Response Time, the overall response time rises from 16.81 seconds at load 5 to 923.18

Load	Overall Response Time	TPS	Trx Used	Energy Need	Bandwidth
5	2.9422	0.9368559116	108.754	257350	667
10	1.89	2.490660025	108.754	257350	667
15	1.926333333	4.215851602	108.754	257350	667
20	1.21	7.942811755	108.754	257350	667
25	1.44952	6.084205403	108.754	257350	667
30	0.87634	15.26717557	108.754	257350	667
35	1.6543	21.10977081	108.754	257350	667
40	0.783	22.39641657	108.754	257350	667
45	1.7653	22.68602541	108.754	257350	667
50	2.1912	22.81854691	108.754	257350	667

Table 6.6: Tron Testnet benchmark for Data Heavy Smart Contracts

seconds at load 50, signaling a major delay in Ethereum Testnet shown in Table 6.5 as transaction load increases. At increasing load, the reduced transaction throughput is measured to be 0.2001 at load 5 and 0.1341 at load 50, a decrease in TPS from 0.2001 to 0.1341. Gas Price sees a considerable range, starting at 6.34 billion wei up to 8.71 billion wei, and Gas Required goes from 273,126 up to 302,178 units. Ether used shows a small increase from .00162 Ether to .00253 Ether as the load increases indicating more expensive costs of processing greater volumes of transactions.

Instead, features where data is used heavily (i.e., great LP rate, high utilization, and congestion) show more stable performance for the Tron Testnet shown in Table 6.6. Overall Response Time is consistently low across all loads, from 0.783 s to 2.9422 s. TPS is increased from 0.9369 at load 5 to 22.8185 at load 50 which indicates enhanced capability of transaction processing with increasing load. The transaction load has no effect on resource consumption metrics like Trx Used (108.754 Trx), Energy Need (257,350) or Bandwidth (667 units). Tron's metrics can sustain stability, meanwhile, implying that it is capable of handling data-intensive processes within the predictable resource usage that Ethereum also boasts, but as the data increases.

6.2.2 Overall Response Time vs Load

The performance metrics show the operational efficiencies of data-heavy smart contracts on the Ethereum and Tron testnets. The Overall Response Time is 16.81 seconds at a load of 5 but ballooned to 923.18 seconds at a load of 50 for the Ethereum testnet, which reflects a significant delay in transaction loads shown in Table 6.1. Ethereum is struggling to handle higher transaction volumes efficiently as this delay shows. This also results in a reduction in transaction throughput (TPS), from 210.0 at load 5 to 134.1 at load 50, indicating that the ability of the system to process transactions decreases with increasing load. Like Gas Required, Gas Price shows wide variation, from 6.34 billion wei to 8.71 billion wei. With higher processing costs for bigger transaction volumes, Ether rises slightly from 0.00162 to 0.00253.

On the other hand, the Tron testnet executes data-heavy operations at a more stable performance than the Tron mainnet. As shown in Table 6.2, across loads it is found that there is a low Overall Response Time (from 0.783 s to 2.9422 s for load 5 to load 50), where the performance in dealing with the loads has been proven better. At load 5 the TPS improves to 22.8185 and since the load only increases from 5

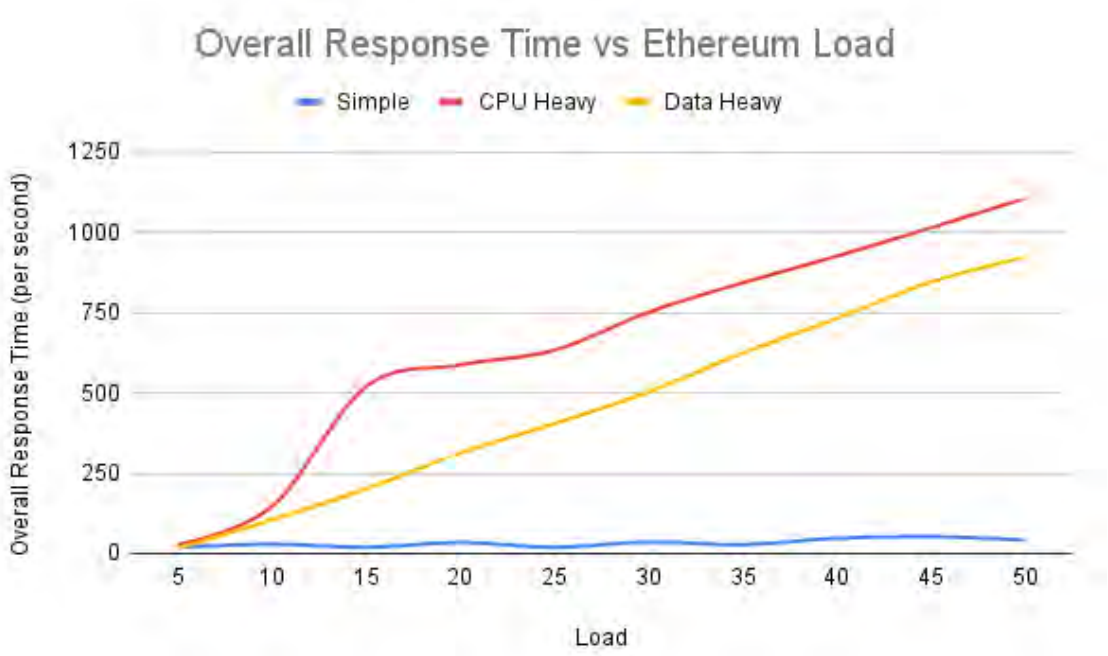


Figure 6.1: Overall Response Time vs Ethereum Load

Figure 6.2: Overall Response Time vs Tron Load

to 50, this means Tron has the capability to process more transactions. Other metrics of resource use, for instance, Trx Used (108.754) Energy Need (257,350), and Bandwidth (667 units), do not surprisingly vary with the transaction load, demonstrating Tron's ability to stay predictable and stable resource usage. Tron's ability to handle such high-load scenarios indicates that it is more prepared to handle data-intensive processes which also remain stable, and efficient.

6.2.3 TPS vs Load

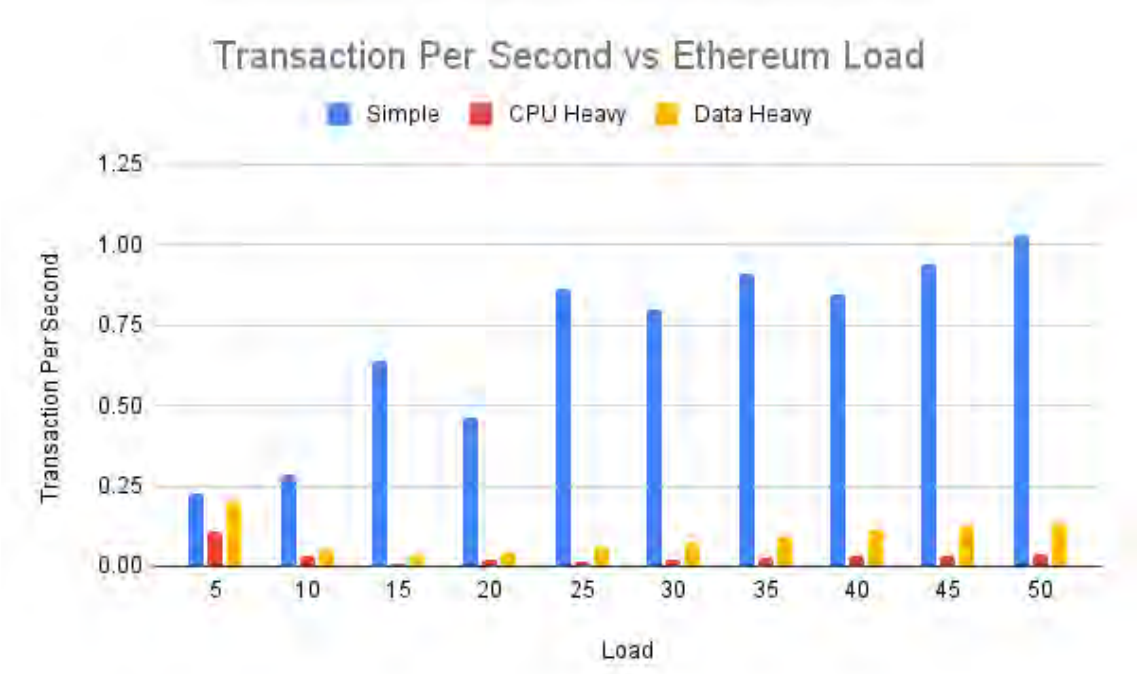


Figure 6.3: Transaction Per Second vs Ethereum Load

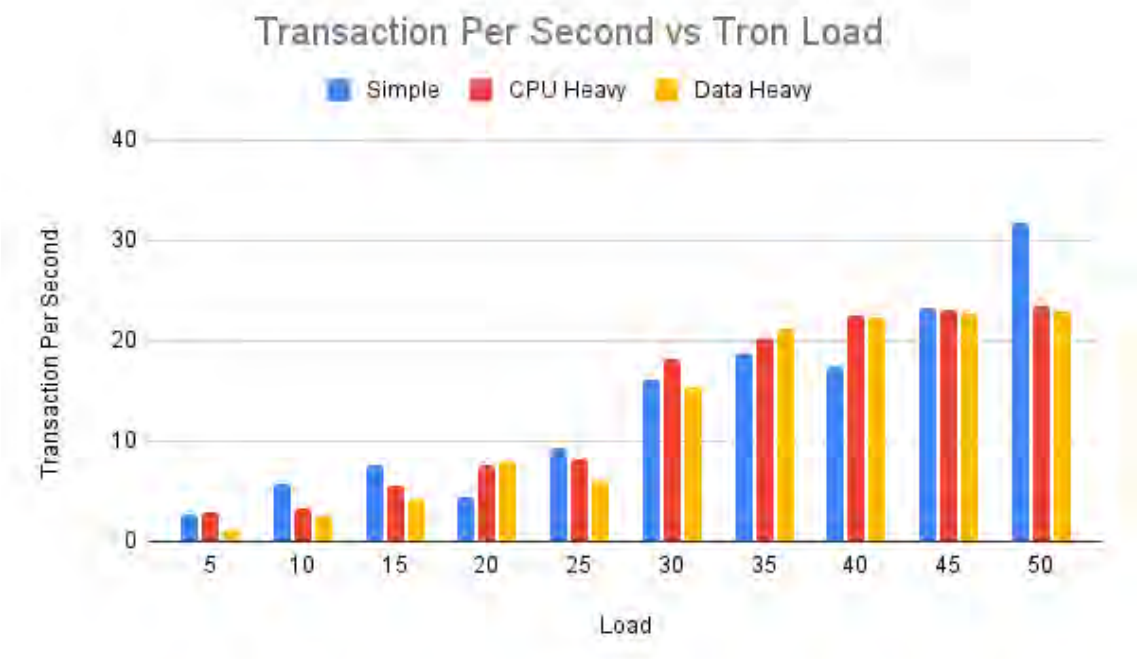


Figure 6.4: Transaction Per Second vs Tron Load

Both 6.3 and 6.4 Graph findings show that there are large population differences between Ethereum and Tron for a given load. TPS for Ethereum in Graph 6.3 findings are low in all function types and remain so over a range of load. While simple functions exhibit marginal improvements in TPS, CPU-heavy, and data-heavy functions

do not experience substantial gains and demonstrate Ethereum's inherent shortages in solving transaction throughput effectively at inflated volumes. This trend suggests that although the load increases heavily, Ethereum's capacity for processing a larger number of transactions per second does not increase substantially, meaning the system's poor efficiency at handling a higher number of transactions.

Conversely the Tron Graph 6.4 findings shows a very adaptive behavior with TPS growing substantially with load for all function types. For simple functions, the TPS improves from 0.94 at 5 to over 22 at 50, while CPU-heavy and data-heavy functions increase steadily up to 25-28 at maximum. This upward trend of TPS shows that Tron is much better position to handle a growing transactional load than before, and demonstrates the superior scalability and higher throughput the platform can maintain as the load increases.

6.2.4 Cost & Resource Usage

We also conducted our experiment by trying different input loads in each function to get an insight into how Ethereum and Tron handle different input loads within the same function. This allowed us to precisely analyze how cost-efficient Ethereum and Tron are in terms of handling simple, computationally heavy, and data-heavy tasks.

Simple function: (state variable update)

Value (increasing bits)	Response Time	Gas Price (Gwei)	Gas Required	Ether Used
4	23.17273196	5184000577	26640	0.0001381017754
12	141.4369161	5361516934	26640	0.0001428308111
227	13.0385139	6951775620	26640	0.0001851953025
3456	450.0084763	842805088	26652	0.0001557224412
23456	37.0999743	6079414856	26652	0.0001620285647
456839	21.2410686	7603572837	26664	0.0002027416661
3482912	13.1161415	6409665956	26664	0.0001709073331
4294967295	27.312549	8313640314	26676	0.00022178

Table 6.7: Ethereum bit increase input data benchmark for simple smart contract function

Value (increasing bits)	Overall Response Time	Trx Used	Energy Need	Bandwidth
4	1.542	2.61796	5488	313
12	1.556	2.61796	5488	313
227	1.702	2.61796	5488	313
3456	1.631	2.61796	5488	313
23456	1.725	2.61796	5488	313
456839	1.467	2.61796	5488	313
3482912	1.951	2.61796	5488	313
4294967295	1.788	2.61796	5488	313

Table 6.8: Tron bit increase input data benchmark for simple smart contract function

Table 6.7 and 6.8 shows overall response time, gas price, gas required, ether used, trx used, energy need and bandwidth data generated from the simple function where

each data input was experimented by increasing the bits for example, 12 being a 4-bit number, 227 being an 8-bit number upto 4294967295 being a 32-bit number.

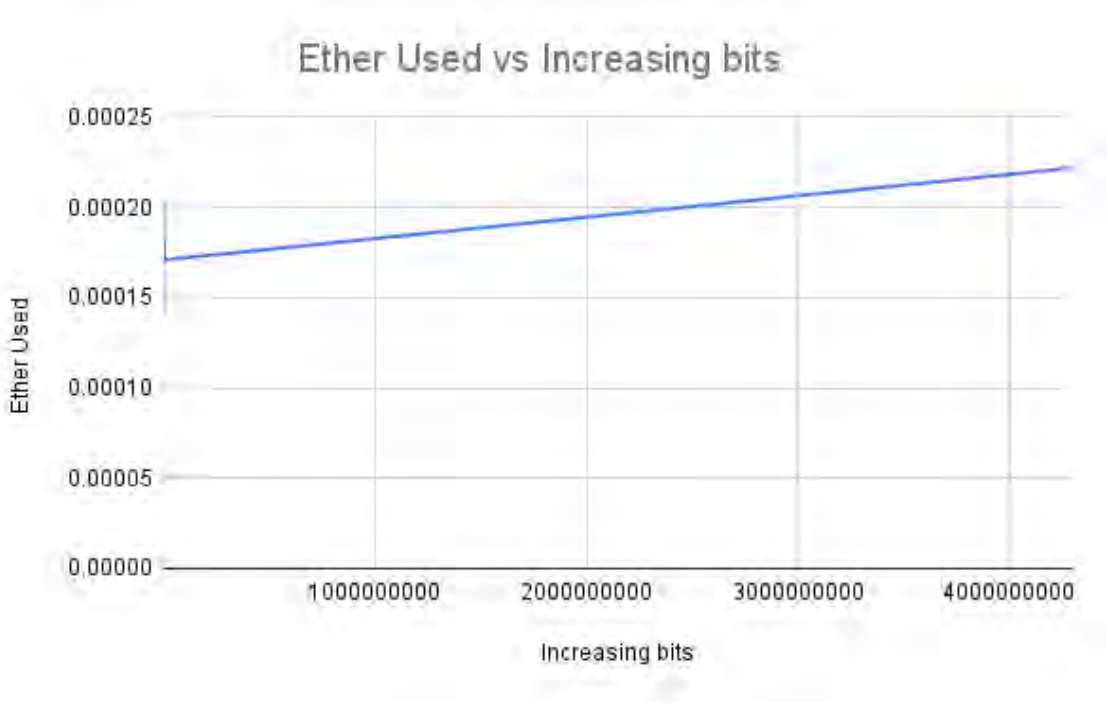


Figure 6.5: Ether Used vs Increasing bits

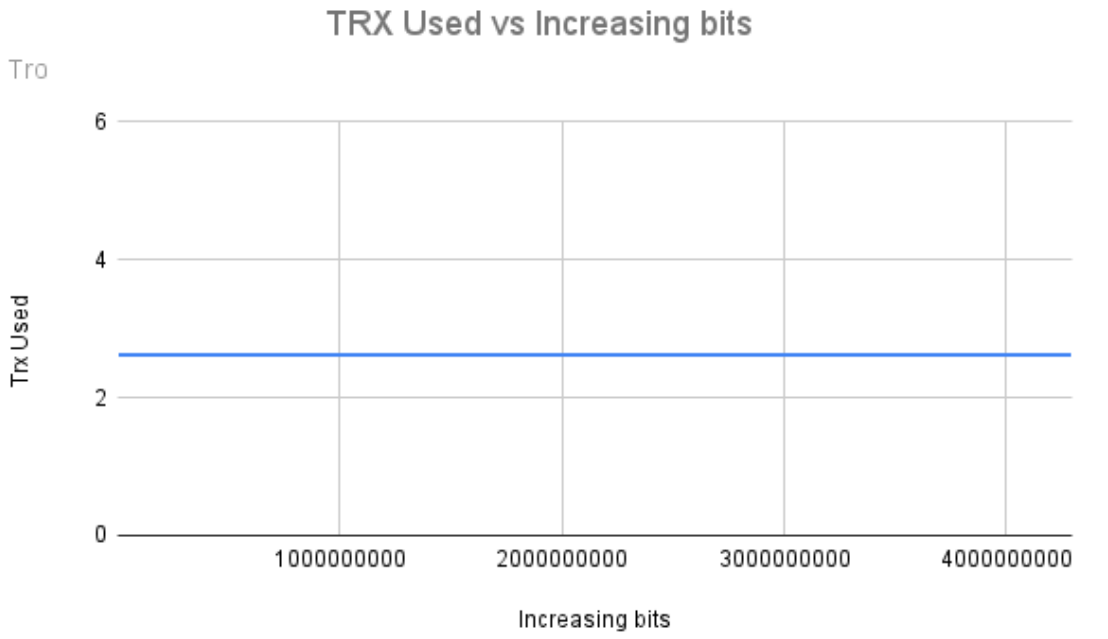


Figure 6.6: TRX Used vs Increasing bits

For the simple function smart contracts, a single state variable updates with every call. We gradually increased the numerical input value with higher bits and observed that the gas required or gas estimation started to increase with some of the higher bit jumps. For example from 8 bit to 12 bit, from 15 bit to 19 bit, from 19 bit to 22 bit, and 22-32 bit. The jump remained at a constant rate of 12 units of gas as

shown in Table 6.7. If we compare with the current market price of Ether which is 1 ETH = $2,477.57 according to 2024-10-15, the cost of Ether was low consistently in comparison to Tron even though the change in gas required was noticed.

In Tron, the cost was constant being 2.61796 TRX for each of the transactions shown in 6.8, and the same for the Energy and Bandwidth which is $ 0.42 per transaction (1 TRX = 0.16$) as of 2024-10-15. Ethereum's cost increased from 0.34 to 0.46, where almost half of the transactions were cheaper compared to Tron with some being a bit more expensive than Tron.

This reveals that for simpler and low-complexity functions, Ethereum showcases better cost-efficient characteristics than Tron overall. However, the notable response time variation still brings a drawback to the equation.

CPU Heavy Function:

Table 6.9 and 6.10 shows overall response time, gas price, gas required, ether used, trx used, energy needed, and bandwidth data generated from the CPU heavy function based on nth Fibonacci calculation for both Tron and Ethereum.

nth Fibonacci	Response Time	Gas Price (Gwei)	Gas Required	Ether Used
10	17.7365517	5376960523	88259	0.0004745651588
15	44.4888236	5227004541	714165	0.003732943698
20	18.2809708	4485086155	7655888	0.03433597175
25	block limit exceed	-	-	-
30	block limit exceed	-	-	-
40	-	-	-	-

Table 6.9: Ethereum nth Fibonacci benchmark for cpu heavy smart contract function

nth Fibonacci	Response Time	Trx Used	Energy Need	Bandwidth
10	1.628	35.42542	83601	313
15	1.682	150.31264	357142	313
20	1.880	150.31264	357142	313
25	1.640	150.31264	357142	313
30	1.753	150.31264	357142	313
35	1.593	150.31264	357142	313
40	1.793	150.31264	357142	313

Table 6.10: Tron nth Fibonacci benchmark for cpu heavy smart contract function

For Ethereum, the Ether cost rose quickly as the computational complexity increased. The steep increase from Fibonacci of 15 reflects the increasing computational complexity, where each step lines up exponentially to the calculations and the gas fees as shown in the Graph 6.7 findings. Furthermore, our experiment revealed that EVM cannot process the transaction when the Fibonacci value is incremented to 25 because of Ethereum's maximum gas limit of 30 million gas units per block mechanism shown in Table 6.9. The computational cost of Fibonacci of 25 is so much

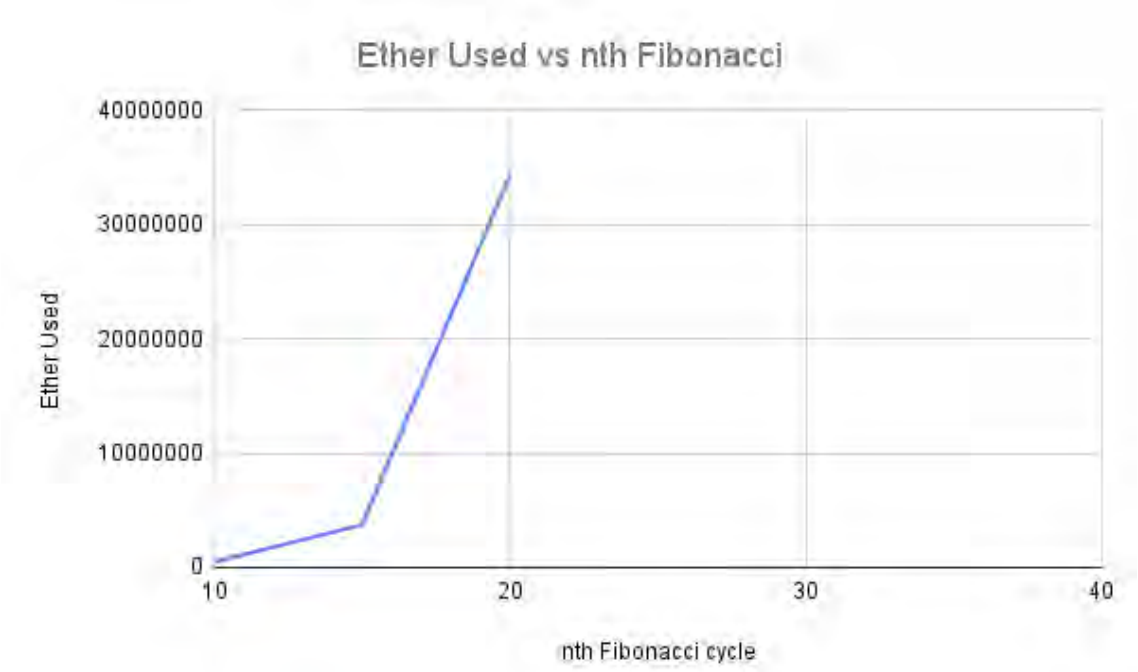


Figure 6.7: Ether Used vs nth Fibonacci

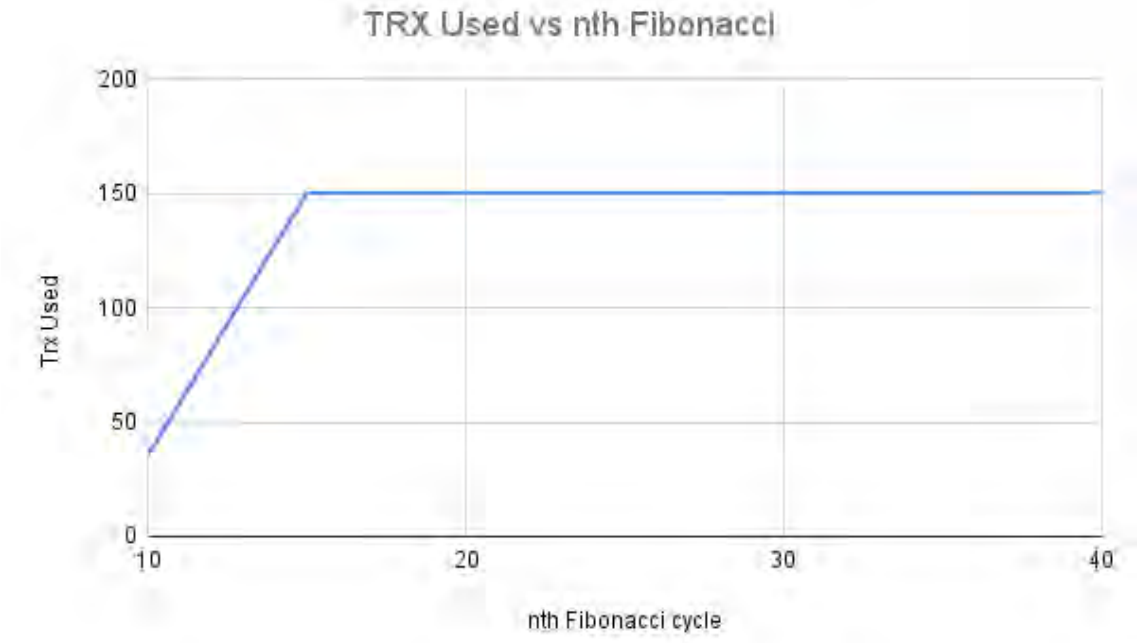


Figure 6.8: TRX Used vs nth Fibonacci

that such a huge number of recursive calls demands extremely high gas consumption which exceeds the 30 million gas per block cap and causes a failure of transaction. So from such observation, we can say that Ethereum faces exponentially higher costs as the complexity of computation increases.

In Tron, the TRX usage was much more stable and consistent from Fibonacci of 15. Where 35.42 trx was used for Fibonacci of 10 and From Fibonacci of 15 a sig-

nificant increase to 150.31 Trx can be seen as shown in Table 6.10. Surprisingly the cost remained constant even till Fibonacci of 40 showing no further increase in cost as the Fibonacci number increased. This Explains that Tron's resource usage model handles high computational tasks efficiently compared to Ethereum. First of all, it didn't fail when the value was pushed to 25 which happened in the case of Ethereum, on top of that continued to process even higher values with constant Trx usage. Such characteristics explain the Virtual Machine of Tron consists of optimizations that once a threshold is crossed, the TVM assigns a higher, but fixed, Energy requirement for a range of complex operations which we can see through our experiment. As a result, making it more scalable and predictable for CPU-heavy operations.

If we were to analyze the current market prices of both expenses, where 1 ETH = $2,477.57 and 1TRX = $0.16 according to 2024-10-15, for Ethereum Fibonacci of 10 costs $1.18, Fibonacci of 15 around $9.25 and, for 20 the cost increased to $85.08 which portrays exponential rise in cost as the computation bar was raised. Here $ is the unit currency of United States Dollar (USD). For Tron on the other hand, the TRX usage was much more stable and consistent from Fibonacci of 15. Where 35.42 trx equivalents to $5.67, the large increase to 150.31 Trx can be seen but way less cost-wise which is $24.05 in compared to Ethereum.

So for Computational computational-focused tasks Tron wins in terms of scalability and cost-effectiveness. The obvious cost gap between Ethereum and Tron as computational tasks increased shows that Tron is way more economical for managing CPU-heavy tasks.

Data Heavy (Increasing Array size)

Table 6.11 and 6.12 shows overall response time, gas price, gas required, ether used, trx used, energy needed, and bandwidth data generated from the data-heavy function based on different array sizes on Ethereum and Tron testnet. The value of each of the Array sizes was filled with increasing numbers for example for array size 5, [1,2,3,4,5] and increasing according to n size array to the last element being n of each array.

For Ethereum, with an increase in array size, the gas required increased gradually by about 114470–114471 units, with only a slight variation to 114472 at one point as shown in Table 6.11. A steady rise in Gas Prices can also be noticed. However, we used the default set gas price of the network where the network selects a relatively optimized gas price based on network congestion. The reason why even though a linear increase in gas required can be noticed the fluctuating nature of the gas Price caused the graph to be steeper in some of the points since Ether usage is the combination of gas price and gas used as shown in Graph 6.9 findings.
For, Tron Trx Use rises largely from size 5 until size 15 being, 54.7584 Trx to 108.754 Trx and finally to noticeable jump to 150.82664 Trx as we can see in Table 6.12 as we can see the increase in the Graph 6.10 findings. After that, the increase rate reduces a lot to only steadily increasing 0.16 Trx till when the array size input rises to 60. And even if the array size is pushed further the rate of Trx usage increase remains constant at 0.16. So, we stopped the experiment till the array size was 60.

Value (array size)	Response Time	Gas Price (Gwei)	Gas Required	Ether Used
5	14.1312886	664032811	141556	0.00009399782859
10	73.1304387	637594208	256026	0.0001632406947
15	29.0873101	472766152	370496	0.0001751579683
20	13.0625352	467621620	484967	0.0002267810542
25	16.4315898	722018802	599437	0.0004328047846
30	76.5838303	839587462	713908	0.0005993882058
35	25.0773725	993486345	828379	0.000822983225
40	261.6598356	843245580	942849	0.000950532519
45	21.1548135	1181105815	1057320	0.0012488068
50	17.620757	1187453394	1171791	0.0013914472
55	25.620757	1189014893	1286263	0.00152939
60	221.6310457	2301363354	1400734	0.003223597896

Table 6.11: Ethereum Array size increase benchmark for data-heavy smart contract function

Value (array size)	Response Time	Trx Used	Energy Need	Bandwidth
5	1.962	54.7584	129170	507
10	1.494	108.754	257350	667
15	1.515	150.82664	357142	827
20	1.940	150.98664	357142	987
25	1.863	151.14664	357142	1147
30	1.908	151.30664	357142	1307
35	1.632	151.46664	357142	1467
40	1.944	151.62664	357142	1627
45	1.609	151.78664	357142	1787
50	1.514	151.94664	357142	1947
55	1.468	152.10664	357142	2107
60	1.735	152.26664	357142	2267

Table 6.12: Tron Array size increase benchmark for data-heavy smart contract function

As Energy and Bandwidth are the resource representation of transactions in Tron, So Energy usage and Bandwidth usage is a direct representation of how much Trx was burned for the expense. In this experiment, Energy usage can be seen as constant after an array size of 15. But Bandwidth increases by exactly 160 bandwidth from the start. This explains the minor TRX difference we can see where though the Energy Usage remained the same from size 15, the extra increase of 160 bandwidth resulted in a 0.16 difference in Trx used.

So comparing the cost with the current market price, for Ethereum the cost revealed to be an array size of 5 equals $0.23, array size of 10 around $0.40, array size of 15 $0.43, array size of 25 $1.07 till array size of 60 which increased to around $7.98. A relatively low cost compared to what we saw in the case of the computational heavy experiment. For Tron however, the cost was high starting with $8.76 to more than twice in the next input which is $17.40. Even though the increasing rate was reduced a lot after that, the cost was revealed to be way too much than what Ethereum charged.

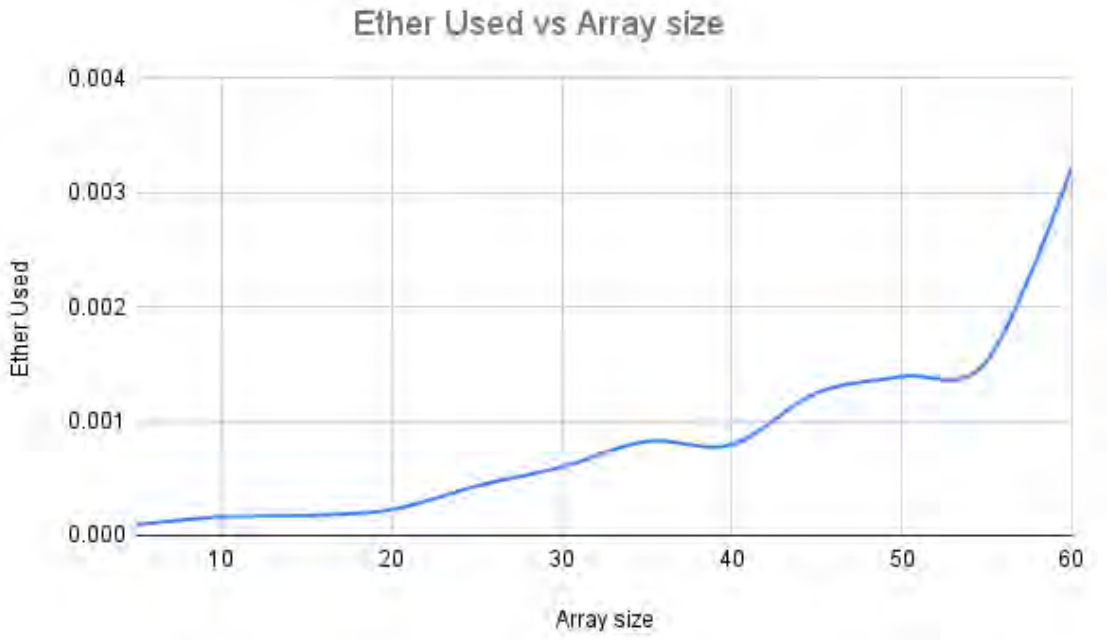


Figure 6.9: Ether Used vs Array size

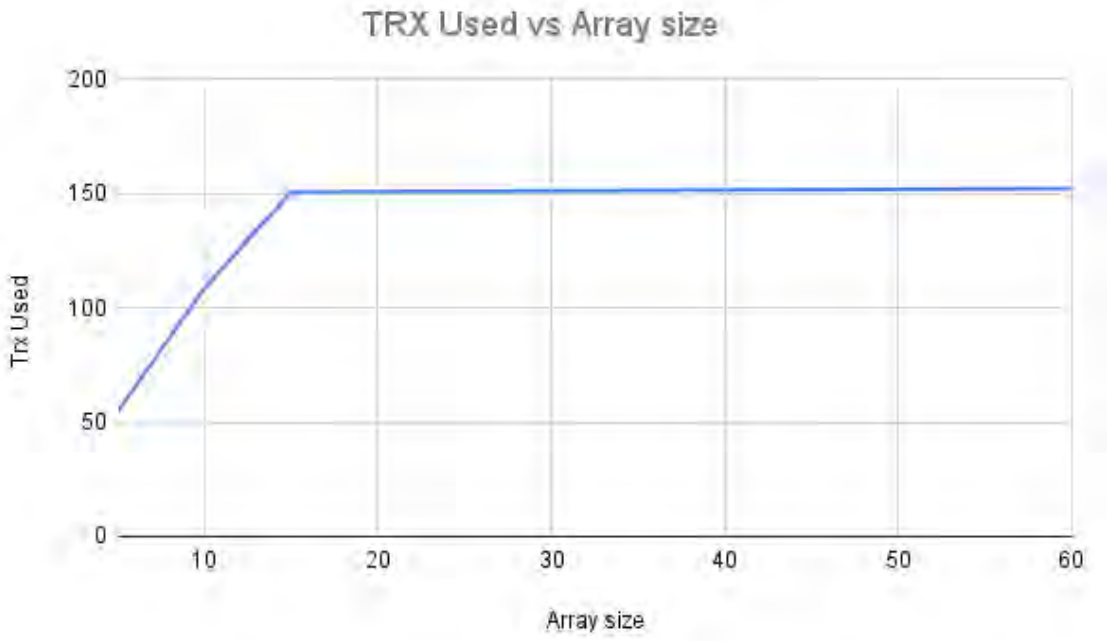


Figure 6.10: TRX Used vs Array size

From this, we can get insight that Tron handles Energy consumption well but costs higher in comparison to Ethereum. This is due to the fixed cost structure of Tron which focuses more on performance at the cost of relatively higher baseline cost. Ethereum despite the scalability challenges that it faces, shows more affordability for handling data-intensive tasks as the array size increases making Ethereum more cost-efficient for operations that involve data-heavy characteristics compared

to Tron.

So from the insight of the overall resource usage characteristic of both blockchain platforms, Tron Wins where Predictability and scalability are observed to be Tron's primary strengths. For computational-intensive operations, where the computational complexity increases considerably as we saw in the Fibonacci calculations, Tron stays cost-effective and predictable after reaching a certain threshold. Users who require to run operations that involve high computation and cannot afford the risk of gas limit failures on Ethereum will find Tron's pricing model superior. Ethereum shines on Simple tasks like we saw for the regular state variable update function as well as for reasonably complex data-handling operations in terms of cost-effectiveness. Data-heavy operations, Ethereum gives a clear cost advantage over Tron, especially for larger array sizes. The overall response time may be a negative side of Ethereum, but if we were to focus solely on the cost and resource usage characteristics Ethereum takes the lead.

6.2.5 Local Network Result Analysis (Ethereum)

Load	Overall Response Time	TPS	Gas Price	Gas Required	Ether Used
5	0.07841648	58.84360547	1674879209	43746	0.00007326926588
10	0.08416278	103.2850854	1346610484	24032	0.0000321112736
15	0.07756344	160.3000818	1091392051	24032	0.00002602533485
20	0.07390456	228.2185375	1000982962	24032	0.00002386943971
25	0.07392842	225.5153477	1000860288	24032	0.00002386651443
30	0.07410254667	310.1852426	1000000008	24032	0.00002384600019
35	0.0795265	336.5423447	1000000008	24032	0.00002384600019
40	0.07939052	369.2874691	1000000008	24032	0.00002384600019
45	0.08394902444	416.2592722	1000000008	24068	0.00002388200019
50	0.081300636	455.5792052	1000000008	24068	0.00002388200019

Table 6.13: Ethereum Local benchmark for Simple Smart Contracts

Load	Overall Response Time	TPS	Gas Price	Gas Required	Ether Used
5	0.21802242	18.20856822	1000018514	728137	0.0007281504807
10	0.28968522	25.33017247	1000001674	728137	0.0007281382189
15	0.23524482	40.28234162	1000000254	728137	0.0007281371849
20	0.242947025	46.19920311	1000000198	728137	0.000728137016
25	0.206525228	77.14737401	1000000008	728137	0.0007281370058
30	0.24866855	67.13654791	1000000008	728137	0.0007281370058
35	0.2086621229	89.9245687	1000000008	728137	0.0007281370058
40	0.191672625	122.9007779	1000000008	728137	0.0007281370058
45	0.2163456044	108.7558094	1000000008	728137	0.0007281370058
50	0.209398962	134.7742316	1000000008	728137	0.0007281370058

Table 6.14: Ethereum Local benchmark for CPU Heavy Smart Contracts

Load	Overall Response Time	TPS	Gas Price	Gas Required	Ether Used
5	0.07297624	60.71070378	1000000008	257456	0.0002574560021
10	0.06847398	110.4191511	1000000008	257456	0.0002574560021
15	0.07610920667	162.0766558	1000000008	257456	0.0002574560021
20	0.071279925	198.8846549	1000000008	257456	0.0002574560021
25	0.080815868	259.0201165	1000000008	257456	0.0002574560021
30	0.07920648	318.1329414	1000000008	257456	0.0002574560021
35	0.07730942571	335.4652952	1000000008	257456	0.0002574560021
40	0.07850613	377.8825352	1000000008	257456	0.0002574560021
45	0.07999850889	446.583192	1000000008	257456	0.0002574560021
50	0.07943323	476.5017907	1000000008	257456	0.0002574560021

Table 6.15: Ethereum Local benchmark for Data Heavy Smart Contracts

Table 6.13: The results from the Ethereum Simple Function indicate that as the Load increases in the range of 5 to 50, the Overall Response Time is exceptionally low and moves between the values of 0.073 and 0.083 seconds. At load 5, we have the Transactions Per Second (TPS) of 58.84, and at load 50 of 455.58, which shows exceptional performance of processing simple transactions with higher loads. For 5 and above, load, the gas price begins at 1,674,879,209 wei and stabilizes at 1,000,000,008 wei. Gas Required decreased from 43,746 units at load 5 to 24,068 units at load 50, and Ether Used decreased from 0.00007327 Ether to 0.00002388 Ether as more transactions pass through the network.

Table 6.14: Ethereum CPU Heavy Function also tells that the overall response time of CPU-intensive tasks remains almost the same when the load changes from 5 to 50; from 0.19 to 0.29 seconds. The TPS goes from 18.21 at load 5 to 134.77 at load 50, all the time being able to handle heavy transactions. Gas Price is at a constant rate of 1,000,000,008 wei, Gas Required is 728,137 units and Ether Used is 0.0007281370058 Ether through all loads, all of which show resource usage with low CPU heavy operations.

Table 6.15: Overall Response Time is consistently low, running from 0.07 to 0.08 seconds, and Ether Data Heavy Function shows that that the function stays consistently low, varying by about 0.01 seconds between 0.07 and 0.08 seconds from 5 to 50. This increases from 60.71 at load 5 to 476.50 at load 50 and presents efficient data-intensive transaction processing. For data-heavy functions, they consistently set Resource metrics such as Gas Price to 1,100,000,007 wei, Gas Required to 257,456 units, and the use of Ether to 0.0002574560021 Ether, which insinuates constant and predictable resource use.

6.2.6 Overall Response Time vs Load

In Graph 6.11, the visualization section includes Overall Response Time vs. Ethereum Load, which views the relationship between Overall Response Time and Load for simple, CPU-heavy, and data-heavy functions. As shown in the graph, response times are consistently low across all types of functions, and simple and heavy data functions stay below 0.1 seconds while CPU-heavy functions stay under 0.3 seconds. This shows how Ethereum can maintain high frequency and low latency in transaction types across a local network.

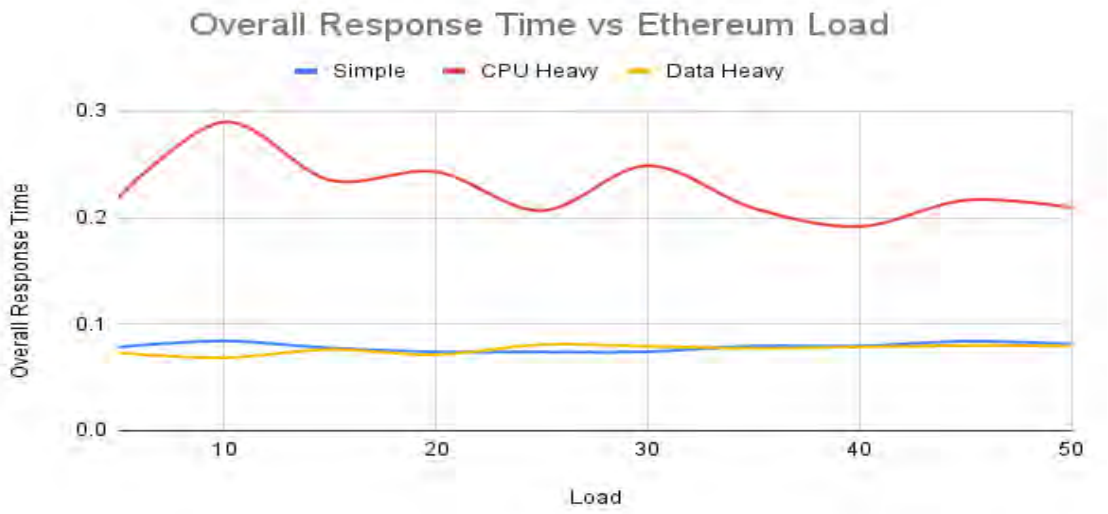


Figure 6.11: Overall Response Time vs Ethereum Load

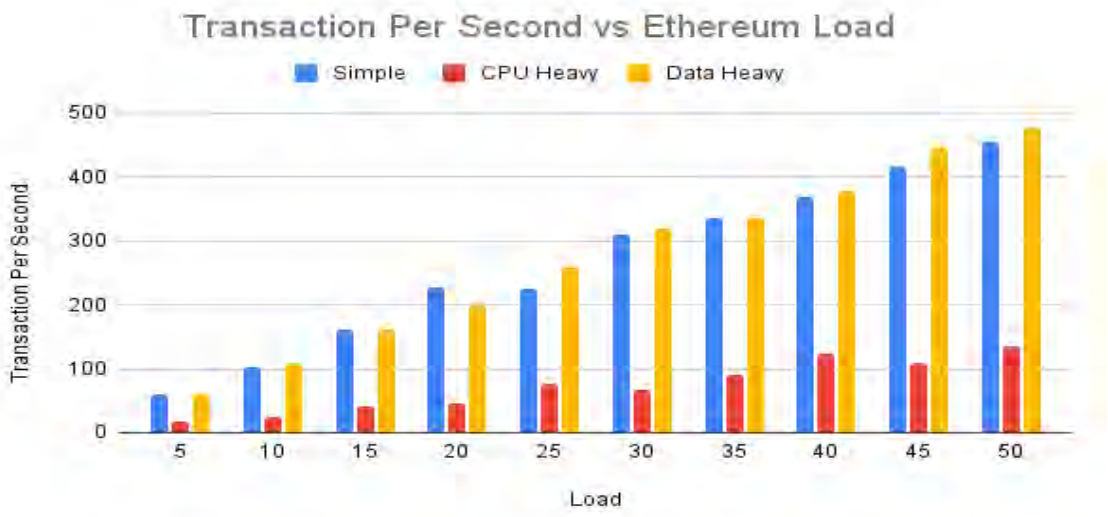


Figure 6.12: Transaction Per Second vs Ethereum Load

6.2.7 Overall Response Time vs Load

In Graph 6.12, Transactions Per Second (TPS) vs Ethereum Load illustrates the scalability of TPS vs load across the three function types. At load 50, the graph shows an enormous increase in TPS for simple and demanding functions, up to 455.58 and 476.50 TPS respectively. CPU-heavy functions also run much better as well, hitting 134.77 TPS at the highest load. Ethereum's effective scaling of the transaction throughput on a local network, in particular for simpler and data-intensive operations, is given by this.

Chapter 7

Discussion

In this chapter, we analyze our research objectives and compare them to our findings and research. We also discuss our developed framework's advantages, challenges we faced during the implementation, and limitations. We also acknowledge what we will be working and implementing in this research in the future.

7.1 Research Objective Analysis

- **RO1:** After a rigorous amount of research, testing, and benchmarking we have managed to determine the best metrics that are required to evaluate a blockchain application performance. Finally, we determined that Transaction Per Second (TPS), latency, wallet consumption, and overall response time are necessary parameters for evaluating the performance of applications deployed on public blockchain systems.

- **RO2:** As we have explained in our experimental discrepancies people have used different ways for their setup in benchmarking blockchain applications. We have used test networks and local networks to deploy and execute our benchmarks, and as a result, we have understood why testing in test networks of blockchain systems that mimic the components of the mainnet is very important in getting proper information. The evaluation that was done locally did not give us proper information.

- **RO3:** After analyzing all the benchmarking data, we were able to determine which evaluation metrics were better efficient for performance evaluation. Our experiments contribute towards gaining some insights into how efficient these resources are for Ethereum, Tron, and other similar projects. However, comparing it to other coins, Tron proved far more efficient in terms of CPU: when working with a computationally heavy task, Tron was much better. Whereas Ethereum imposed restrictions because of the gas limit per block of transactions and failed under high computational loads. In terms of wallet consumption, Tron was slightly more costly for simple and data-heavy tasks, while Ethereum was less costly for these same types of operations. On the other hand, though, Tron, ran stable costs for computationally intensive tasks, while Ethereum's costs were going off the charts. The network has

proven to be the second axis of difference, as varied as the complexity of tasks, and illustrates the importance of developers considering the network efficiency of an application. As we understand these patterns of resource utilization, developers can adjust their applications according to optimized efficiency and cost-effectiveness for different blockchain platforms.

- **RO4:** We prioritized parameters that directly influence user experience on public blockchains, such as transaction throughput (TPS) and confirmation latency. By focusing on these metrics, we assessed how each blockchain's performance affects end-users. Ethereum, while offering lower costs for simple transactions, exhibited higher latency and lower TPS compared to Tron, potentially impacting user experience in applications requiring quick confirmations. Tron demonstrated lower transaction latency and higher TPS, making it more suitable for applications where speed is critical. These differences emphasize the importance of aligning blockchain choice with the application's user experience requirements. Developers must balance cost considerations with performance needs to optimize user satisfaction.

- **RO5:** We compared Ethereum's PoS and Tron's DPoS under the same load conditions, which revealed obvious differences in performance. For example, the overall response time of Ethereum at load 25 is 19.094 08229 seconds and a TPS of 0.860 109 1593. However, Tron performs much better, with an overall response time of 1.78712 seconds and Tron TPS of 9.255831174. While Ethereum's response time is approximately 90.64% compared to Tron and certainly in terms of transaction throughput Tron is 976.1%. This in turn shows that DPoS is another strain of the scaling we must consider when contemplating or proposing scaling methods, allowing it to process exponentially more transactions per second while simultaneously staying orders of magnitude lower in response time. In contrast, Ethereum is faster but comes with slower transaction speeds and longer response times — and it's a bit more centralized. Since Tron is better prepared to handle the high transaction volume of high-performance applications with frequent microtransactions, Ethereum's PoS is preferable in cases where decentralization is more important.

7.2 Advantages

- **Comprehensive Comparative Analysis**: By evaluating both Tron and Ethereum in terms of several performance metrics, we gain a proper understanding of each platform's relative strengths and weaknesses.

- **Standardized Methodology**: This improved the reliability of our results as we used consistent experimental setup and equipment. This allowed for direct, raw, and fair comparisons between the platforms. Also, as the structure of smart contracts needs to be the same for both blockchain platforms, this makes it easy to get the benchmark data at the same time.

- **Real-World Applicability**: We deployed smart contracts on public test networks to ensure that our findings generalize to real-world blockchain applications and not just theoretical ones.

- **Focus on User Experience**: Our study addresses practical considerations important for developers by emphasizing metrics that directly affect end users, such as transaction latency and cost.

7.3 Challenges & Limitations

Challenges

- **Test Network Related Challenges:** As testing on a blockchain main network is too costly and risky, we used testnets for our evaluation. However, the accuracy of our data was hindered because in some cases test networks did not fully replicate mainnet conditions and had discrepancies. Additionally, it also required a significant amount of faucets to test inside testnets and the amount provided per day was too low for our analysis. We had to create 100 wallet addresses and manually collect faucets, which was challenging.

- **Variability in Network Performance:** It was difficult to maintain consistent conditions in test networks due to inherent fluctuations. Overall response time and throughput measurements were complicated due to these fluctuations.

- **Smart Contract Development Complexity:** Designing smart contracts that accurately reflect different types of workloads requires careful planning. Errors in code could lead to failed transactions or misleading results.

- **Tool Compatibility and Learning Curve:** Working with multiple blockchain platforms required proficiency with different tools and libraries. Differences in documentation quality and community support added complexity.

- **Data Collection and Synchronization:** Capturing and synchronizing performance data across platforms was complex and needed to be accurate. Developing consistent logging mechanisms required additional effort to ensure data accuracy.

Limitations

- **Test Network Constraints:** Since test networks do not fully reproduce mainnet conditions (mainly network congestion, fees, and miner behavior), performance may differ on the mainnet.

- **Limited Scope of Functions Tested:** Specifically, we researched three types of functions: simple, CPU heavy, and data heavy functions. We did not explore other types of functions.

- **Limitations in Consensus Mechanisms:** Ethereum and Tron's consensus mechanisms in particular were considered Proof of Stake (PoS) and Delegated Proof of Stake (DPoS). The framework was not tested on other consensus protocols, like Practical Byzantine Fault Tolerance (PBFT) or Proof of Authority (PoA), which restrict the applicability of the framework to such blockchain systems.

- **Smart Contract Structural Limitations:** Both of the smart contracts should have a similar ABI structure to be tested in your framework, testing between different structural applications will generate errors.

7.4 Future Work

Currently, our framework supports only two public blockchain testing, Ethereum and Tron. In our future work, we aim to include Solana and other public blockchains in our system. Ethereum and Tron have compatible and exchangeable smart contracts both can be written in solidity language. Solana smart contracts can be written in different languages even in solidity, so future implementation of Solana in our system will be our first priority. In our current system, users can only test their application by providing abi, smart contract address, and selecting test networks, but our second priority is to implement a sophisticated user interface where users can deploy and test their application instantly. Our third priority is to measure the CPU and memory usage of the distributed node which is processing the smart contracts function execution. As for Ethereum, a further exploration in the pool pending time is necessary because a substantial amount of time gets spend in the pool from the overall response time till any validator approves the transaction.

Chapter 8

Conclusion

Public blockchain systems have become essential in various real-world domains due to their transparency, security, and decentralized nature. However, evaluating their performance remains a significant challenge. Most of the existing works do give us meaningful insights but they lack in standardization of making a proper protocol. These methods often overlook crucial factors such as consensus mechanisms, and user experience, which are vital for understanding a blockchain system's effectiveness.

The performance evaluation framework we propose addresses these issues by providing standardized metrics and implementation. Our framework focuses on key performance indicators such as transactions per second, overall response time, wallet consumption, and resource utilization. By deploying smart contracts on test networks like Ethereum's Sepolia and Holesky, and Tron's Shasta and Nile, we were able to simulate real-world evaluations and benchmark applications without the risks and costs associated with mainnet deployments. Our experiments demonstrated how different consensus mechanisms Proof of Stake in Ethereum and Delegated Proof of Stake in Tron—impact performance under various workloads. The findings revealed that while Tron excels in handling high transaction loads with lower latency, Ethereum offers cost advantages for certain types of transactions, especially data heavy operations.

As trust is essential in both the physical and digital worlds, having reliable and efficient blockchain systems is crucial for the continued growth of decentralized applications. The main obstacle has been the lack of standardized evaluation methods, making it challenging to optimize blockchain performance and enhance user experience. By introducing our comprehensive framework, we aim to mitigate these issues and contribute to the development of more efficient, scalable, and user-centric public blockchain applications. Our model not only provides a better way to assess existing platforms but also sets the stage for future research in blockchain technology. We believe that our contributions can help bring about a new domain of research focused on performance optimization, ultimately making blockchain technology more accessible and beneficial for all users.

Bibliography

[1] S. Nakamoto, "Bitcoin whitepaper," *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, vol. 9, p. 15, 2008.

[2] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, pp. 22–23, 2013.

[3] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An introduction," *R3 CEV, August*, vol. 1, no. 15, p. 14, 2016.

[4] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 3–16.

[5] R. Yasaweerasinghelage, M. Staples, and I. Weber, "Predicting latency of blockchain-based systems using architectural modelling and simulation," in *2017 IEEE International Conference on Software Architecture (ICSA)*, IEEE, 2017, pp. 253–256.

[6] 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.

[7] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring ethereum network peers," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 91–104.

[8] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0. 8.13," *Whitepaper*, 2018.

[9] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu, "A detailed and real-time performance monitoring framework for blockchain systems," in *Proceedings of the 40th international conference on software engineering: software engineering in practice*, 2018, pp. 134–143.

[10] Z. Dong, E. Zheng, Y. Choon, and A. Y. Zomaya, "Dagbench: A performance evaluation framework for dag distributed ledgers," in *2019 IEEE 12th international conference on cloud computing (CLOUD)*, IEEE, 2019, pp. 264–271.

[11] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *16th USENIX symposium on networked systems design and implementation (NSDI 19)*, 2019, pp. 95–112.

[12] L. Yang, V. Bagaria, G. Wang, *et al.*, "Prism: Scaling bitcoin by 10,000 x," *arXiv preprint arXiv:1909.11261*, 2019.

[13] 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.

[14] M. Dabbagh, M. Kakavand, M. Tahir, and A. Amphawan, "Performance analysis of blockchain platforms: Empirical evaluation of hyperledger fabric and ethereum," in *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, IEEE, 2020, pp. 1–6.

[15] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance evaluation of blockchain systems: A systematic survey," *IEEE Access*, vol. 8, pp. 126 927–126 950, 2020.

[16] M. Firdaus, "A review of performance analyzing on public and private blockchain platforms," *OSF Preprints*, 2020.

[17] I. A. Omar, R. Jayaraman, K. Salah, M. C. E. Simsekler, I. Yaqoob, and S. Ellahham, "Ensuring protocol compliance and data transparency in clinical trials using blockchain smart contracts," *BMC Medical Research Methodology*, vol. 20, pp. 1–17, 2020.

[18] J. Sedlmeir, H. U. Buhl, G. Fridgen, and R. Keller, "The energy consumption of blockchain technology: Beyond myth," *Business & Information Systems Engineering*, vol. 62, no. 6, pp. 599–608, 2020.

[19] S. Woo, J. Song, S. Kim, Y. Kim, and S. Park, "Garet: Improving throughput using gas consumption-aware relocation in ethereum sharding environments," *Cluster Computing*, vol. 23, pp. 2235–2247, 2020.

[20] Z. Zheng, S. Xie, H.-N. Dai, *et al.*, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[21] H. Huang, W. Kong, S. Zhou, Z. Zheng, and S. Guo, "A survey of state-of-the-art on blockchains: Theories, modelings, and tools," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–42, 2021.

[22] F. Irresberger, K. John, P. Mueller, and F. Saleh, "The public blockchain ecosystem: An empirical analysis," *NYU Stern School of Business*, 2021.

[23] R. Jidrot and G. Perumal, *Workload characterization and performance evaluation of a blockchain implementation for managing federated cloud resources-assuming a peer-to-peer energy management use case*, 2021.

[24] M. Kaur, M. Z. Khan, S. Gupta, A. Noorwali, C. Chakraborty, and S. K. Pani, "Mbcp: Performance analysis of large scale mainstream blockchain consensus protocols," *IEEE Access*, vol. 9, pp. 80 931–80 944, 2021.

[25] B. Lashkari and P. Musilek, "A comprehensive review of blockchain consensus mechanisms," *IEEE access*, vol. 9, pp. 43 620–43 652, 2021.

[26] J. Pan, Z. Song, and W. Hao, "Development in consensus protocols: From pow to pos to dpos," in *2021 2nd International Conference on Computer Communication and Network Security (CCNS)*, IEEE, 2021, pp. 59–64.

[27] Q. Wang, R. Li, Q. Wang, and S. Chen, "Non-fungible token (nft): Overview, evaluation, opportunities and challenges," *arXiv preprint arXiv:2105.07447*, 2021.

[28] A. Ghosh, I. Sarkar, M. Dey, and A. Ghosh, "Artificial intelligence and blockchain: Implementation perspectives for healthcare beyond 5g," in *Blockchain Applications for Healthcare Informatics*, Elsevier, 2022, pp. 93–116.

[29] S. Honkamäki, "Evaluating ethereum development environments," M.S. thesis, S. Honkamäki, 2022.

[30] I. Alom, M. S. Ferdous, and M. J. M. Chowdhury, "Blockmeter: An application agnostic performance measurement framework for private blockchain platforms," *IEEE Transactions on Services Computing*, 2023.

[31] G. Mathur, "Ganache: A robust framework for efficient and secure storage of data on private ethereum blockchains," 2023.

[32] Etherscan, *Holesky etherscan explorer*, Accessed: 2024-10-15, 2024. [Online]. Available: https://holesky.etherscan.io/.

[33] Etherscan, *Sepolia etherscan explorer*, Accessed: 2024-10-15, 2024. [Online]. Available: https://sepolia.etherscan.io/.

[34] Tron Developers, *Tron developer documentation*, Accessed: 2024-10-15, 2024. [Online]. Available: https://developers.tron.network/.

[35] Tronex, *Nile testnet explorer*, Accessed: 2024-10-15, 2024. [Online]. Available: https://nileex.io/.

[36] Tronex, *Shasta testnet explorer*, Accessed: 2024-10-15, 2024. [Online]. Available: https://shasta.tronex.io/.