Omotade Atobatele 2216175

CS351: Project Specification

Development and Comparative Analysis of Blockchain Validation Methods

Introduction

Consensus mechanisms are fundamental to blockchain technology since they are often deployed in a decentralised manner, consensus algorithms ensure the security and integrity of the network. This is achieved by enabling distributed nodes across the network to approve the validity of transactions. Various consensus algorithms such as PoW(Proof of Work) and PoS(Proof of Stake), which Ethereum famously migrated to from PoW, each have their distinct advantages and trade-offs in terms of efficiency, energy consumption, decentralisation, scalability and security. This project aims to develop and implement consensus algorithms not typically used in blockchain contexts, to explore potential alternatives to current technologies.

Background

Blockchain by definition is a decentralised ledger of all transactions across a peer-to-peer network. Using this technology, participants can confirm transactions without needing a central clearing authority [1]. This decentralisation allows for enhanced transparency, security, and resilience against centralised points of failure. The concept of blockchain technology was first postulated in 1991 by Stuart Haber and W Scott Stornetta in their paper titled *'How to time-stamp a digital document'* in which they describe a cryptographically secured chain of blocks that is immutable once confirmed, preventing users from backdating documents or tampering with recorded information [2].

Though its most famous implementation to date is the facilitation of cryptocurrency, most notably Bitcoin, blockchain technology applications have expanded far beyond just digital currencies. Bitcoin was developed by an anonymous programmer (or group) under the pseudonym 'Satoshi Nakamoto' who introduced Bitcoin to the public in their whitepaper *Bitcoin: A Peer-to-Peer Electronic Cash System*. In their whitepaper, Satoshi notes the use of PoW to secure the network and validate transactions. PoW works by requiring nodes to solve complex cryptographic problems, specifically by " scanning for a value that when hashed, such as with SHA-256, the hash begins with several zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash" [3] making it computationally expensive and energy-intensive.

To summarise, in PoW, the blocks are validated by CPUs (nowadays GPUs and/or with specialised hardware like ASICs) completing calculations to find the 'golden' hash and earn the rewards in the form of newly minted cryptocurrency from network fees. There is a moving average of how difficult this is, if the blocks are being mined too quickly the difficulty is increased and vice-versa if it's taking too long. A self-adjusting difficulty mechanism allows a consistent rate of block production regardless of the total available computational power on the network. This system is crucial not only for the scalability but also for its integrity and security as to modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes [4].

However, while PoW has a proven track record so far of being a robust and secure method to maintain blockchain integrity it has a significant drawback: its high energy consumption. The extensive computational power required to solve the cryptographic problems results in high electricity consumption. The CBECI estimates that global electricity usage associated with Bitcoin mining ranged from 67 TWh to 240 TWh in 2023, with a point estimate of 120 TWh [5]. For perspective based on those estimates, it puts electricity used globally to mine BTC in the range of the total electricity consumption in Greece or Australia, respectively. BTC only being 1 of the many thousands of cryptocurrencies available.

Current Literature and Software

Currently, several cryptocurrencies utilise a different consensus algorithm to PoW used by Bitcoin, notably Ethereum, the second largest cryptocurrency by market capitalisation recently migrated to Ethereum 2.0 which saw many improvements to the protocol namely the aforementioned migration from PoW to PoS. Ethereum made the switch in December 2022 citing it as “more secure, less energy-intensive, and better for implementing new scaling solutions compared to the previous PoW architecture” [6].

PoS operates by “validators explicitly stake capital in the form of ETH into a smart contract on Ethereum. The validator is then responsible for checking that new blocks propagated over the network are valid and occasionally creating and propagating new blocks themselves. If they try to defraud the network (for example by proposing multiple blocks when they ought to send one or sending conflicting attestations), some or all of their staked ETH can be destroyed” [7].

Currently, there’s a consensus that PoS is a much more environmentally friendly alternative to PoW. Offering faster transactions and better scalability. However like with PoW it is not without its drawbacks, the primary concern is the risk it poses to decentralisation. There is a risk of a ‘rich get richer’ dynamic. For example, currently, the minimum stake required to become a validator on the Ethereum network is 32 ETH which at the time of writing is roughly equivalent to £56,000. This high entry barrier limits the number of individuals who can participate in network validation, effectively concentrating power in the hands of those who can afford it. Comparatively, users with access to a few hundred pounds can purchase dedicated mining hardware and join the network as a miner, making it more accessible to a broader range of participants. Arguably a more inclusive and decentralised approach to block validation.

The main concerns with PoS, then, are whether it can uphold the decentralised ethos that blockchain technology was founded upon and its robustness as it is not as tried and tested as PoW. As wealthier validators accumulate more rewards the risk of centralisation increases, undermining the core principle of distributed control of the network, one of the aspects that makes blockchain technology so powerful. This debate continues to shape discussions on the future of consensus mechanisms and blockchain governance.

Requirements

Functional Requirements:

Blockchain Creation:

- Develop multiple private blockchain networks, each using a different consensus algorithm (PoW, PoS, PBFT, etc.).
- Ensure each blockchain supports the basic structure (block creation, hashing, chaining, etc.).
- Implement cryptographic hash functions (e.g., SHA-256) for block verification.

Consensus Algorithms Implementation:

- Implement different consensus algorithms (PoW, PoS, DPoS, PBFT, PoA, etc.).
- Each blockchain should have the ability to validate transactions and form a consensus using its respective algorithm.

Transaction Management:

- Support the gathering and broadcasting of transactions on the network.
- Implement validation of transaction details (sender, recipient, amounts, etc.).

Immutable Ledger**:**

- Ensure immutability of the blockchain, where blocks and transactions cannot be altered or deleted once validated.
- Provide an audit trail for transaction validation and block creation.

Performance Metrics & Comparison:

- Gather performance metrics for each consensus algorithm (e.g., time to reach consensus, energy consumption, transaction throughput).
- Provide functionality to compare scalability, efficiency, and resource usage between the different consensus models.

User Interface (Optional):

- Implement a simple interface to view blockchain status, network participants, transactions, and block validations.

Non-Functional Requirements:

1. **Scalability:**
    a. Each blockchain must support scaling to a certain number of nodes and transactions without significant performance degradation.
    b. Test and compare the scalability limits of each consensus algorithm.

2. **Security:**
    a. Cryptographic methods (e.g., digital signatures, hashing) must be securely implemented to protect the integrity of data.
3. **Efficiency:**
    a. Optimise the consensus algorithms to minimise resource usage while maintaining security.
    b. Measure and compare the computational and energy efficiency of each consensus method.
4. **Decentralisation:**
    a. Maintain decentralisation by ensuring that no single node or participant can control the majority of the network.
5. **Environment Impact:**
    a. Assess and minimise the environmental impact of each consensus mechanism (e.g., PoW's energy consumption).
    b. Provide a comparison of the environmental costs between PoW and more energy-efficient models.
6. **Reliability:**
    a. Ensure fault tolerance within the P2P network so that the system can recover from node failures without data loss or downtime.
    b. Use consensus mechanisms that guarantee the network continues operating even when a portion of nodes are unreliable.

Objectives

1. Develop a robust and modular framework to deploy multiple consensus algorithms.
2. Compare the performance of each consensus mechanism in terms of efficiency, scalability, resource consumption, and environmental impact.
3. Provide insights into the trade-offs between decentralisation, security, and efficiency in different consensus models.
4. Highlight real-world applications for utilising different consensus mechanisms by testing in a private network environment.

Methods

To help achieve the objectives of the project, the following methods will be used. The overall project will be built using Python. Python will be used because of its extensive amount of libraries and frameworks that it offers, it is also cross-platform allowing it to be run across devices. The creator also has a lot of experience with the language in the past.

1. **Objective: Develop robust and modular frameworks to deploy multiple consensus algorithms.**
    Method:
    a. Use Python's `flask` framework to create a REST API for managing blockchain nodes and transactions.
    b. For each consensus algorithm, develop separate Python classes that encapsulate the logic for mining, forging, or validating blocks. These classes should be modular and reusable.

   Example: Create `PoW.py`, `PoS.py`, and `PBFT.py` modules.

   c. Store the state of each blockchain (blocks, transactions, etc.) using SQLite for simplicity or a distributed database like CouchDB if horizontal scaling is needed.

2. **Objective: Compare the performance of each consensus mechanism in terms of efficiency, scalability, resource consumption, and environmental impact.**
   Method:
   a. Instrument the consensus algorithms to log the number of operations (CPU cycles) and time taken for block validation. Eg. Use the `time` library to capture start and end times for validation.
   b. Monitor resource usage (memory, CPU) using Python libraries such as `psutil` to compare computational resource intensity between algorithms.
   c. Track energy usage indirectly by estimating the electricity consumption based on the system's performance characteristics and resource usage during the consensus process.

3. **Objective: Provide insights into the trade-offs between decentralisation, security, and efficiency in different consensus models.**
   Method:
   a. Simulate an operating blockchain network using Flask and Python's `threading` or `asyncio` libraries to run concurrent validators.
   b. Compare the decentralisation of each method by tracking the concentration of power in terms of staked assets or computational power.

4. **Objective: Highlight real-world applications for utilising different consensus mechanisms by testing in a private network environment.**
   Method:
   a. Deploy your private blockchain network using Flask, simulating a real-world decentralised application (DApp). **(Stretch)**
   b. Perform tests to show real-time performance in different scenarios (e.g., high transaction volumes, large number of nodes).
   c. Collect performance data for each blockchain and visualise it using Python's `matplotlib` or `plotly`.

Testing

1. **Objective: Develop robust and modular frameworks to deploy multiple consensus algorithms.**

   Testing Method:

   - **Unit Testing:**
     - Write unit tests for each consensus algorithm before implementing the code. Ensure that the PoW, PoS, and PBFT algorithms are correctly calculating the validation of transactions and adding blocks.
     - Use Python's `unittest` or `pytest` frameworks to create test cases for each function within the consensus classes.
     - Test cases should validate that the blockchain is correctly chaining blocks together and rejecting invalid transactions.
   - **Integration Testing:**
     - Test that the Flask API correctly handles requests for initiating consensus mechanisms and returns accurate results.
     - Ensure that the blockchain state is updated correctly when nodes broadcast transactions and receive responses from different consensus algorithms.

2. **Objective: Compare the performance of each consensus mechanism in terms of efficiency, scalability, resource consumption, and environmental impact.**

   Testing Method:

   - **Performance Testing:**
     - Write test cases that simulate high transaction volumes and measure the time taken to validate blocks in each consensus algorithm. Use Python's `timeit` module to test efficiency.
   - **Scalability Testing:**
     - Write tests to simulate an increasing number of nodes and transactions, and measure how well the system scales. Use Python's `asyncio` to simulate concurrent nodes interacting with the blockchain.
   - **Resource Usage Testing:**
     - Use Python's `psutil` library to track CPU and memory usage during block validation, and write tests to ensure each algorithm's resource consumption is within reasonable limits.

3. **Objective: Provide insights into the trade-offs between decentralisation, security, and efficiency in different consensus models.**

   Testing Method:

   - **Security Testing:**
     - Simulate common blockchain attacks and write tests to confirm the network's resilience. For each consensus algorithm, ensure that malicious nodes cannot take over the network. **(Stretch)**
   - **Decentralisation Testing:**
     - Test that no single node has disproportionate influence over the network.
   - **Efficiency Testing:**
     - Test how consensus mechanisms behave when the network is under heavy load. Write tests to confirm that the network maintains performance and consistency even with high transaction volumes.

4. **Objective: Highlight real-world applications for different consensus mechanisms by testing in a private network environment.**

   Testing Method:

   - **Functional Testing:**
     - Simulate real-world DApp scenarios, such as processing transactions in a decentralised voting system. Write tests to ensure transactions are properly validated and blocks are added in real-time.
   - **Usability Testing:**
     - Conduct usability tests by having peers and colleagues interact with the network. They will perform tasks like sending transactions and observing consensus validation.
     - Collect feedback through surveys or questionnaires to evaluate the user experience and improve the system's UI and performance.

Timetable

The project timetable is displayed below in the form of a Gantt chart which has been attached on the last page. Furthermore the timetable below dictates the hours that will be dedicated to the project each week. Subject to change based on whether the project dictates it or not.

	10am	11am	12pm	1pm	2pm	3pm	4pm	5pm	6pm	7pm	8pm	9pm	10pm	11pm
Mon	X	X	X											
Tue			X	X										
Wed						X	X	X	X					
Thu			X	X		X	X	X	X					
Fri														
Sat										X	X	X		
Sun											X	X		

Risk Assessment

The following risks and their mitigations associated with this project:

Risk	Mitigation
Personal Computer may break	Store all files in the cloud so they are accessible from other devices.
Software version change	Use a lower version than currently available
Inability to work due to illness	Over estimate development time to account for these

Resources

1. Git - Used for version control
2. GitHub - Store backup for project and access from other devices
3. Python - Used to write the backend software
4. HTML and CSS - Used for the frontend to facilitate transactions
5. SQLite - To store data in a database.
6. Flask
7. Psutil
8. Asyncio
9. Timeit
10. Unittest/PyTest
11. Matplotlib/Plotly

These resources may change over the course of the project.

Legal, Social, Ethical and Professional Issues

During the development of this project, some testing will be required from people other than the creator, but since these people will be colleagues and friends, there are no ethical issues to consider.

References

[1] Haber, S. and Stornetta, W.Scott. (1991). How to time-stamp a digital document. *Journal of Cryptology*, [online] 3(2). doi:https://doi.org/10.1007/bf00196791.

[2] Morey, M., McGrath, G. and Minato, H. (2024). *Tracking electricity consumption from U.S. cryptocurrency mining operations - U.S. Energy Information Administration (EIA)*. [online] www.eia.gov. Available at: https://www.eia.gov/todayinenergy/detail.php?id=61364.

[3] Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [online] Available at:
https://www.ussc.gov/sites/default/files/pdf/training/annual-national-training-seminar/2018/Emerging_Tech_Bitcoin_Crypto.pdf.

[4] Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [online] Available at:
https://www.ussc.gov/sites/default/files/pdf/training/annual-national-training-seminar/2018/Emerging_Tech_Bitcoin_Crypto.pdf.

[5] PwC (2023). *Making sense of bitcoin and blockchain*. [online] PwC. Available at: https://www.pwc.com/us/en/industries/financial-services/fintech/bitcoin-blockchain-cryptocurrency.html.

[6] Wackerow, P. (2022). *Proof-of-stake (PoS)*. [online] ethereum.org. Available at: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

[7] Wackerow, P. (2022). *Proof-of-stake (PoS)*. [online] ethereum.org. Available at: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

	Term 1										Christmas Holidays			
Week:	1	2	3	4	5	6	7	8	9	10	1	2	3	4
Write Project Specification	🟦	🟦												
Research Consensus Algorithms	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦						
Specification Deadline		◆												
Set up Development Environment			🟦											
Prototype Development - First module of framework				🟦	🟦	🟦								
Test initial prototype with basic consensus algorithms - First module (PoW.py)							🟦	🟦						
Write Progress Report								🟦	🟦					
Progress Report Deadline									◆					
Design Front-End and API									🟦	🟦				
Project Break											🟦	🟦	🟦	🟦

	Term 2										Easter Holidays and Term 3					
Week:	1	2	3	4	5	6	7	8	9	10	1	2	3	4	1	2
Develop Application Front-End and API	🟦	🟦														
Develop Separate consensus algorithms		🟦	🟦	🟦												
Unit Tests for each Algorithm				🟦	🟦											
Optimisation & Comparison					🟦	🟦										
Presentation Preparation						🟦	🟦	🟦								
Project Presentation									🟦	🟦						
Evaluate Project and Prototype								🟦	🟦	🟦	🟦	🟦	🟦			
Draft Final Report								🟦	🟦	🟦	🟦	🟦	🟦	🟦		
Proofread Final Report															🟦	🟦
Final Report Deadline																◆