The Academic College of Tel-Aviv-Yaffo

THE SCHOOL OF COMPUTER SCIENCE

Proof of Queue: A Secure and Efficient Consensus Mechanism for Blockchain Networks

April 2025

Thesis submitted in partial fulfillment of the requirements for the M.Sc. degree in the School of Computer Science of the Academic College of Tel-Aviv-Yaffo

By

Jonathan Shai Isakov

The research work for the thesis has been carried out under the supervision of

Dr. Mor Perry

Acknowledgments

This journey has been an incredible opportunity to learn, grow, and gain a deeper understanding of how meaningful research is conducted—and how new discoveries come to life. I am deeply grateful to my amazing wife, Ilana, whose unwavering support made this thesis possible. Thank you for letting me dedicate weekends and late nights to this work, even when I should have been helping you plan our wedding.

To my dog, Cali—I'm sorry for all the missed games of fetch. I hope the many naps you took beside me in the office were worth it.

Lastly, I would like to extend my sincere thanks to my outstanding advisor, Dr. Mor Perry. Mor, your sharp eye, thoughtful feedback, and tireless dedication through countless iterations shaped this work in every way. I would like to also thank Dr. Moshe Sulamy, thank you for encouraging me to challenge myself, ask questions, and pursue this idea.

Contents

Abstract

Blockchains and cryptocurrencies have become household names, largely due to their ability to facilitate decentralized financial transactions without governmental oversight. However, the consensus algorithms that underpin these systems—most notably Bitcoin's Proof of Work (PoW)—suffer from significant limitations, including high computational overhead, susceptibility to forks, and vulnerability to majority-based attacks. These weaknesses hinder the scalability and efficiency of blockchain networks.

In this work, we propose Proof of Queue (PoQ), a novel consensus algorithm designed to overcome these limitations. PoQ replaces computationally intensive miner selection processes with a deterministic, queue-based approach. The system uses a public, verifiable queue implemented as a binary tree, where miner identities are derived from public keys and ordered using the hash of the previous block. This mechanism pre-determines the block production sequence, effectively eliminating forks and minimizing network overhead.

PoQ improves resilience against adversarial control by penalizing malicious behavior and enabling recovery from attacks—even in scenarios where a single honest miner remains. Our results demonstrate that PoQ achieves substantial improvements in both performance and security, offering a scalable, trustless alternative to existing consensus mechanisms while preserving open participation.

1 Introduction

In 2009 Satoshi Nakomoto [34] developed the first blockchain, a decentralized financial system that allows members which do not trust each other under the byzantine fault tolerance guidelines [30] to partake in a money exchange system called Bitcoin. Bitcoin is a cryptocurrency that allows everyone who wishes to do so to trade using digital assets and currency utilizing the security of public and private keys [36] in combination with hash functions [41]. These mechanisms ensure that the transaction ledger is both secure and tamper-proof. Assuming the widely accepted $P \neq NP$, these cryptographic principals offer strong protection against fraudulent activities. Public and private keys enable secure verification, while hash functions preserve blockchain integrity by linking each block to its predecessor, making it computationally infeasible to alter past transactions undetected. Thus, blockchains and Bitcoin in particular can allow members who do not trust one another to agree on a chain of ownership.

Satoshi Nakomoto suggested using a proof of identity [13] under the guidelines of Proof of Work (PoW). As PoW name suggests the proof of honesty provided is the computational power behind the virtual identity. By trying to solve a hard to answer question the participant (also called miner in many instances) is required to lend its computational power to the algorithm and as a reward of success the participant will reap an agreed upon reward in the form of digital currency. As time moved on more and more proofs of identity developed each with their own draws and improvements. These suggestions while allowing a decentralized agreed upon chain of ownership are resource intensive, slow or vulnerable to majorities attacks and are heavily influenced by forks [10] [42] [4] [15].

In this work, we propose a novel consensus algorithm called Proof of Queue (PoQ). PoQ allows participants to register as future block publishers by joining a public queue, where their public keys serve as identifiers in the queue. The order in which participants advance in the queue is determined by an agreed-upon mechanism, enabling anyone with knowledge of the current queue, last block and time to determine the next block publisher without additional cryptographic verification. This approach streamlines block publishing by reducing the computational overhead required to validate each block and greatly reduces the risk and impact of forks, as the order of block production is predetermined. Additionally, because the queue is based on public keys (which also represent participants wallets), and since queue positions are a finite resource, the incentive for malicious behavior is diminished. Further more we show how in the event of a majority attack, a single honest participant in the queue can displace all malicious actors and restore integrity to the system by replacing them with honest members. Thus, PoQ mitigates the risk of majority-based attacks within the blockchain network while allowing faster block publication.

1.1 Background

In [30], the authors state that "a reliable computer system must be able to cope with the failure of one or more of its components." To illustrate this challenge, they introduce the byzantine generals thought experiment. The thought experiment illustrates the difficulties of creating a distributed and secure system where some of the members might be faulty or even malicious. In their results they show an algorithm that is capable of reaching a consensus as long as two thirds of the nodes are not faulty.

We use the definition of consensus from [21].

Definition 1. Consensus is the process of reaching agreement among a group of people or entities on a specific decision or action.

In our context consensus means that members of the same computer network agree on the value or course of action.

The Idea of a distributed and decentralized currency has been a goal since the 1980's. Though some algorithms were created none were able to act with consensus without a central authority. W. Dai's "b-money" [11] was the first to suggest using a hard to solve mathematical problem in order to regulate the creation of new currency without the need for a central authority. Another important step forward was made by Hal Fine [16] using b-moneys concept of money creation with the Hashcrash puzzle to create the first Proof of Work. But sadly even that fell short as he had to rely on a central authority in his solution [7]. In the paper [34], the anonymous researcher, or group of researchers, using the alias Satoshi Nakamoto, laid the foundation for what would later be called the blockchain. The blockchain is the first algorithm that was able to fully combine W. Dai's concept of money creation within a fully decentralized algorithm under the constrictions suggested at [30]. Nakamoto's algorithm, like those that followed, proposes a set of predefined actions that, if followed, enables a majority of honest participants to reach consensus and authenticate new transactions in the shared ledger.

We use the definition of Ledger from [7].

Definition 2. Ledger A state transition system where there is a "state" consisting of the ownership status of all existing bitcoins and a 'state transition function' that takes a state, a transaction and outputs a new state which is the result.

As such we derive that a blockchain should operate as a state machine recorder. Allowing the users to record transactions between members and to authenticate the creation of new funds. In a blockchain, consensus is used to guarantee that all nodes on the network agree on the current state of the network and the authenticity of transactions. A transaction Could be considered the request to transfer currency between user A and B, the result of the transaction should be Success if A has enough funds or Error if he lacks the needed funds to perform the action as can be seen in Algorithm 1.

In order to authenticate these transactions each input into the transaction needs to be signed using the private key [36] of the owner of the unspent coins. Since these transactions are not part of a centralized system keeping a database of these transactions is not a viable option. This prompted the usage of the Block, i.e, a set of transactions which are published together. In order to keep consensus of the order of transactions a timestamp was added [17]. In addition the hash of the previous block allows us to follow

Algorithm 1 Transaction

```
1: account initial state : {Bob: Z$, Alice:Y$}
2: A request from Bob to send Alice 20$
3: if Bob.funds >= 20$ then
4:     Create the transaction Tx = Bob ->Alice 20$
5:     account end state : {Bob: Z-20$  Alice:Y+20$}
6: else
7:     alert Error on the transaction
8:     account end state : {Bob: Z$, Alice:Y$}
9: end if
```

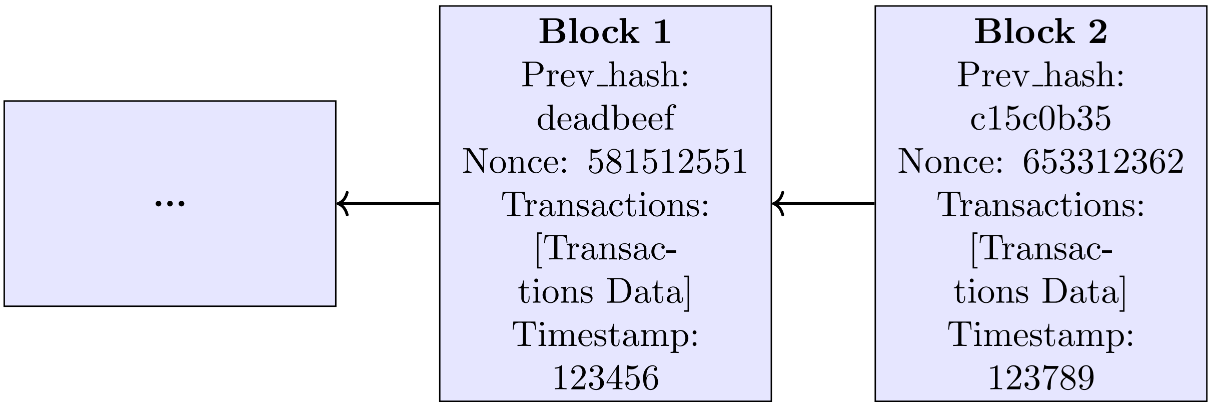

Figure 1: Blockchain depiction (adapted from [34]).

each transaction building the current state and validating its correctness. Lastly, in order to make the creation of new blocks hard (approximately one every 10 minutes [31]), The Nonce (appears in Figure 1) was introduced. The Nonce is a part of the Proof of Work implemented by Satoshi [34] and we discuss it further in subsection 1.1.1.

Definition 3. Block A set of transactions published together alongside the timestamp and previous blocks hash.

Thus, [34] provides the grounds for a successful blockchain as seen in Figure 1 as its structure allows blockchains to publish new transactions while keeping consensus:

1. It uses private and public keys to verify transactions.
2. It uses timestamps to provide a provable timeline for everyone in the chain to accept.
3. It uses the hash of the previous block to verify its continuity.
4. The majority can determine the correctness of the chain and affect it.

We now turn to explain why should a miner lend his computational power to the blockchain. Collecting transactions, validating them and attempting to publish new blocks is an expensive effort that requires network and computational effort. The solution to these was already suggested by W. dai at [11].

1. Each block prompts the minting of new coins, rewarded to the miner.
2. Granting the publishing miner a premium (a percentage of the transactions in each block).

Definition 4. Miner A member of the blockchain who attempts to publish new blocks.

We use the definition of Liveness from [2].

Definition 5. Liveness defines liveness as guaranteed service or starvation freedom, this definition stipulates that each member of a distributed network will have his request fulfilled eventually and that the system wont halt altogether.

[11] insures liveness, as independent and unrelated members of the network will each benefit from both publishing other peoples transactions as well as new blocks. This in turn incentives members to lend their computational power to insure both guaranteed service and starvation freedom as each miner will benefit only when satisfying these needs and do not stand to directly benefit from delaying block releases or not publishing a specific transaction.

The authors of the blockchain faced another issue, how to define a "participant" also known as a miner in the blockchain, as using non-physical objects like IP addresses could be faked [13]. Therefore in his algorithm later dubbed Proof of Work (PoW) Satoshi [34] chose to use the CPU power as a proof of identity meaning if a majority of the computational power comes to a consensus then the chain is authenticated (as each miner added his own block to the chain). As time moved on more and more ways to provide proof of identity have been made some of the popular methods today include Proof of Work, Proof of Stake, Delegated Proof of Stake and PBFT (these algorithms will be explained in depth under a dedicated subsection). These methods balance the speed of publishing new blocks, probability of a "Faulty" (in our case malicious) majority arising and the probability of errors in their chain.

Definition 6. Forking A split in the chain were two different blocks point to the same previous block.

Another issue was how to choose the correct chain. Since several different blocks can point to the same previous block [4]. This is a major issue onto itself as it wastes the resources of the blockchain (requiring everyone to keep multiple states at all time) and CPU of miners attempting to create new blocks on chains that wont be continued. In addition the ability to fork allows for a number of known attacks the main one being "double spending" where a member publishes a transaction in one fork and after receiving the goods in return quickly releases a number of blocks on a different chain in which the transaction did not happen, thus keeping his coins or transferring them to someone else [10].

In the paper [42] under the section "Taxonomy of blockchain systems" we see a comparison of different consensus algorithms. The authors compares private and public blockchains (also known as non-byzantine where one can assume no node is malicious) and between different blockchain algorithms. As we propose a mining algorithm that supports public blockchains we now explore them further.

Table 1: Comparison of Consensus Mechanisms.

Property	**Proof of Work**	**Proof of Stake**	**Practical Byzantine Fault Tolerance**	**Our Algorithm**
Node identity management	Open	Open	Permissioned	Open
Energy saving tolerance	No	Partial	Yes	Yes
Power of adversary	<25% computing power	<51% stake	<33.3% faulty replicas	99%
Example	Bitcoin	Peercoin	Hyperledger Fabric	None

This table compares the properties of various consensus mechanisms, highlighting the differences in node identity management (who can add new participants to the mining process), energy efficiency, tolerance against adversaries, and examples of systems employing these mechanisms. "Our Algorithm" represents a proposed method offering high adversary tolerance and energy efficiency while not restricting who can join as new miners.

1.1.1 Proof of Work

Proof of Work (PoW) is a method that requires each miner to try and solve a hard to answer question in order to publish a new block. The original Bitcoin algorithm [7] suggested the user create a Merkle tree, a structure that allows us to validate quickly a large number of transactions (as can be seen in

Figure 2). This structure allows miners to decrease the storage needed for the blockchain. In a Proof of Work (PoW) consensus mechanism, the *nonce* (short for "number used once" as appears in Figure 1) is a key part of the mining process. Miners continuously modify the nonce to produce new hashes. The goal is to find a nonce that, when hashed together with the other data, produces a hash that satisfies the difficulty requirement (e.g., a hash that starts with a certain number of leading zeros). Once a miner finds a valid nonce that meets the required difficulty, the block is added to the blockchain, and the miner is rewarded. Its important to note that the trail and error process is the source of huge resource waste [15] and slowness as its designed to produce a block every 10 minutes [31] limiting the number of transactions. The nature of the algorithm also means that as miners pull their resources together into "mining pools" [31] their ability to implement a 51% attack increases. Lately, such mining pools have become a source of worry for users of Bitcoin as the top 3 mining pools together control about 72% of the computational power in Bitcoin [39].

So in conclusion a miner will have to follow the following steps to publish his block under PoW:

- Miners take the block header data, which includes the previous block's hash, the current block's timestamp, the Merkle root of the transactions in the block, and the nonce.
- They hash this block header data with the nonce using a cryptographic hash function (e.g., SHA-256 in Bitcoin).
- If the resulting hash meets the target difficulty, the block is accepted by the network.
- If the hash does not meet the target, the miner adjusts the nonce and tries again, repeating the process until they find a valid hash.

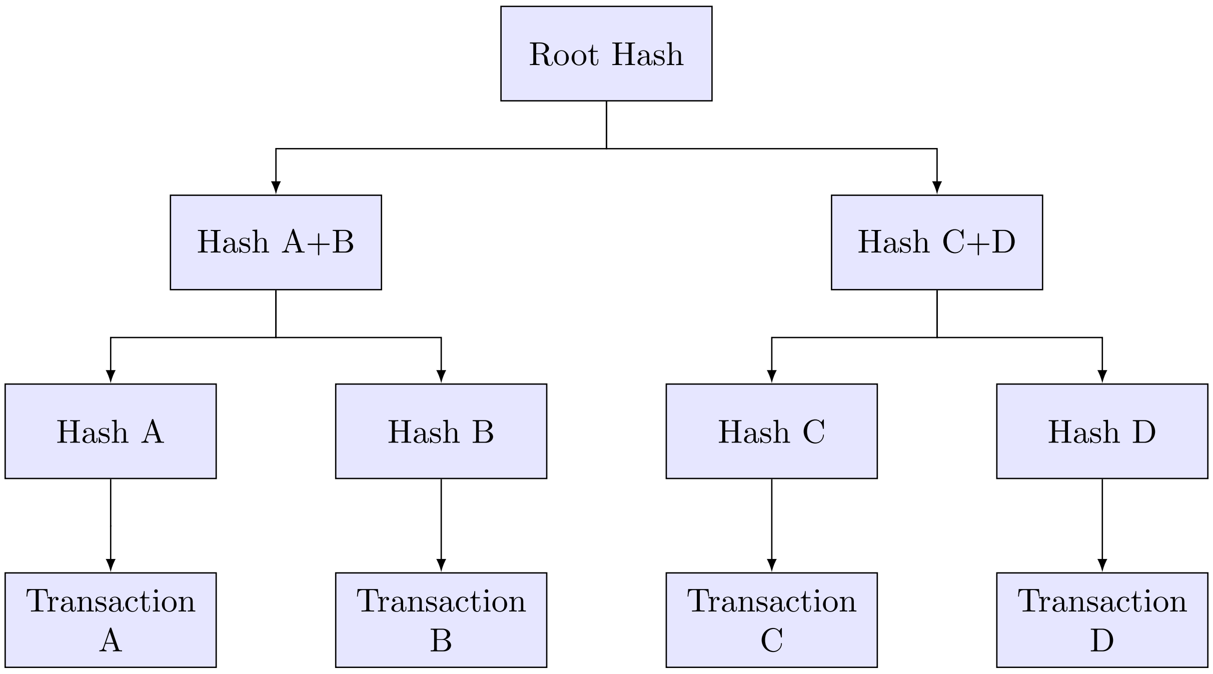

Figure 2: A simple diagram of a Merkle tree.

1.1.2 Proof of Stake

Proof of stake(PoS) [4] [26] is the most used current alternative for Proof of Work (PoW) as Ethereum the second largest coin transferred to this method [25]. Instead of using the computational power of its members to determine who gets to write new blocks it uses a pseudo-random lottery based on a ”stake” of coins the participant stands to lose for malicious behavior. The more coins a participant holds the higher his chances of being chosen as a miner (called validator under PoS) or even to propose new blocks himself. This method eliminates much of the redundant computation done in PoW. However as can be seen in [4] it does not eliminate forks but rather insures forks exist in the blockchain. Furthermore the speed of mining under these algorithms though increased from PoW is still about 20 transactions per second [29]too slow to truly become an alternative to modern financial systems. We now explore Ethereum’s Beacon Chain as explained in [25]. Ethereum’s Beacon Chain serves as the backbone of its Proof of Stake (PoS) consensus mechanism. The Beacon Chain is organized into *slots* and *epochs.* A slot is a 12-second period during which a new block may be added to the blockchain. An epoch is a collection of 32 slots (6.4 minutes). Validators must stay synchronized with the network’s timing to participate efficiently. The goal is to propose and confirm blocks during each slot, but some slots may remain empty. To become a validator, users must stake up to 32 ETH (name of the coin), which activates one validator. Validators earn rewards for submitting valid attestations and penalties for being offline or proposing invalid blocks. The rewards and penalties are designed to incentivize good behavior while minimizing risks for validators. Penalties for serious offenses, such as double voting, can lead to slashing, where the validator loses part or all of their staked ETH. Validators are chosen pseudo randomly to propose new blocks and participate in attesting (voting) on the validity of blocks. Attestations, weighted by the validator’s balance, are recorded on the Beacon Chain and help to determine the head of the blockchain. Validators are organized into *committees* to increase security. Each slot has a committee composed of at least 128 validators, and their role is to attest to the state of the blockchain and ensure that the proposed block is legitimate. A pseudorandom process called RANDAO is used to select proposers and shuffle validators into committees as can be seen in Figure 3. Ethereum’s PoS protocol uses *checkpoints* to finalize blocks. A checkpoint is the first block in an epoch. Validators cast votes for both the current checkpoint and the preceding one, which helps secure the blockchain against malicious behavior. A checkpoint is finalized once it garners a 2/3 super majority of votes from validators. Once finalized, blocks cannot be reverted.

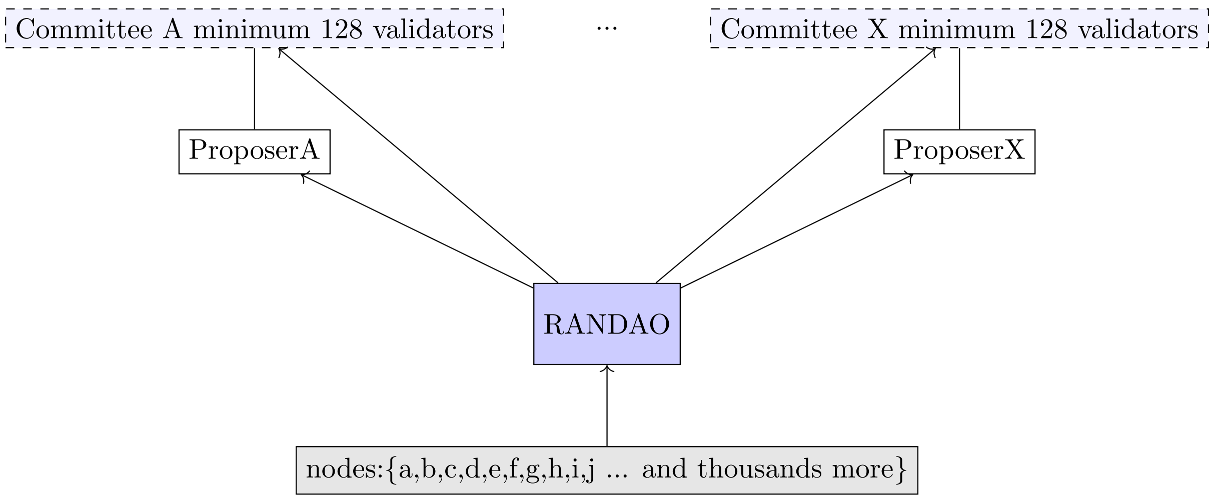
Committee A minimum 128 validators
...
Committee X minimum 128 validators
ProposerA
ProposerX
RANDAO
nodes:{a,b,c,d,e,f,g,h,i,j ... and thousands more}

Figure 3: At every epoch, a pseudorandom process RANDAO selects proposers for each slot, and shuffles validators to committees.

1.1.3 Delegated Proof of Stake

Delegated Proof of Stake (DPoS) is a variant of Proof of Stake (PoS) introduced by Dan Larimer in 2014 [28], designed to enhance scalability and reduce the energy demands associated with classical Proof of Work (PoW). In DPoS, token holders continuously vote to elect a fixed number of delegates (often called witnesses or block producers) who are entrusted to validate transactions and produce new blocks [28]. Because block production is concentrated in a small set of elected participants, DPoS enables faster block times and higher throughput relative to most PoW or PoS systems [35, 37]. Furthermore, a delegate that repeatedly fails to perform or acts maliciously can be voted out by stakeholders, creating a governance layer meant to enforce accountability and responsiveness [35].

However, DPoS has drawn criticism for potentially compromising decentralization. With a limited number of block producers, the network can become dependent on a small group, raising concerns about collusion or voter apathy among stakeholders [35, 37]. If influential stakeholders align in a way that cements certain delegates' positions, the protocol risks concentrating decision-making power [1]. Additionally, unlike some PoS mechanisms that implement strict slashing penalties for misbehavior, DPoS relies heavily on voter vigilance; its security and fairness hinge on stakeholders remaining active and informed during the delegate selection process [35, 37]. Despite these drawbacks, many modern blockchain platforms adopt DPoS to reap the benefits of high throughput, rapid finality, and flexible on-chain governance.

1.1.4 Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov first described Practical Byzantine Fault Tolerance (PBFT for short) in [9]. Later, with the emergence of blockchain and bitcoin, PBFT was transformed into a mining algorithm described in [20]. This algorithm is highly time-consuming as the publishing miner needs to wait for approval from two thirds of the network under the majority constraints in [27] to approve each new proposed block. Thus the algorithm becomes mostly implausible for large scale usage due to its high consumption of network bandwidth and long approval time for new blocks [24].

1.2 Our Results - Proof of Queue (PoQ)

In this work, we design a new consensus mechanism for blockchain. Our algorithm named Proof of Queue (PoQ) uses the blockchain's already existing infrastructure consisting of participants using public and private keys, publishers being chosen as a correlation of pseudo-random selection utilizing the random nature of Hash functions and transactions being published in Blocks holding a timestamp. In fact the idea of an agreed upon queue is even established in part in each epoch of a PoS.

We propose a new method of choosing and agreeing on the next block publisher as well as deciding on the correct chain. Our algorithm uses a queue of future miners, each miner on his turn publishes his block and gains the ability to fill the newly vacant position in the tail of a fixed-size queue by adding a member (he can add himself if he chooses to do so) to the queue. The queue which is public grants a fast and secure method for deciding the identity of the next block publisher.

We show that this method is capable of significantly reducing the resources needed for publishing new blocks while improving the number of transactions being published (if not matching them in comparison to PoW). This is done both by reducing the network overhead, the computation needed for verification of new blocks and eliminating forks when no malicious activity is involved. In addition we show how this new algorithm is capable of dealing with the 51 percentage attack in a way that theoretically allows a single honest miner to rebuild the entire queue by himself.

1.3 Related Work

The paper [10] highlights that though originally designed as a free decentralized currency the emergence of specialized mining equipment changed the blockchain to a business venture. As such and as PoS uses stakes instead of computational power we can start and suggest networks where not all participants need to own specialized equipment. Additionally, the resource-intensive nature of Bitcoin's consensus mechanism as discussed in [15], pushes the publisher to suggest a queue based blockchain. While they suggest implementing this as a method for choosing the next transaction to be published we believe the idea of a queue has better chances of success when designed to chose the next miner. It is also important to note the importance of timestamps to our solution, as proven in [17] it is impossible to create a consensus when the time of transactions is unclear. As such even when debating new ideas for attacks as seen in [14] the timing of the release of new blocks is the basis of mining attacks but not the forgery of timestamps themselves. Even in PoS when a miner is not in sync with the rest of the miners or is offline he might lose his turn or even be punished [25]. Lastly, the randomized nature of Hash functions as discussed in [41] along side the public and private key infrastructure [36] though used in PoW are a valuable tool that can provide each algorithm that chooses to use them with anonymity, randomness and security.

Blockchain networks operate on a peer-to-peer (P2P) architecture [38], where all nodes communicate directly without relying on centralized servers. This decentralized nature underpins blockchain's core principles of trustlessness and resilience, but it also presents challenges in ensuring efficient message propagation and maintaining consensus across distributed nodes. The P2P architecture inherently leads to higher propagation delays and potential network congestion, especially as the number of participants increases [5]. Improving blockchain networks is therefore crucial to address these scalability and performance bottlenecks while preserving decentralization. Current research focuses on optimizing message propagation to reduce latency [38], developing scalable consensus mechanisms [6], and implementing Layer 2 solutions like state channels and rollups [38] to process transactions off-chain. It is important to note that research on the underlying network itself is being conducted [5] where it shows that the main issue with block size increase is its need to propagate the network, thus knowing which IPs are about to release a new block and connecting to them/ neighbors directly will highly accelerate the propagation rate.

Beyond their traditional use as decentralized money ledgers, blockchain technology has emerged as a transformative solution across diverse industries. One prominent use case is supply chain management [18], where blockchain provides transparency, traceability, and immutability to track goods from origin to consumer, reducing fraud and improving efficiency. Another critical application lies in healthcare [19], enabling secure and decentralized storage of medical records, ensuring data integrity and privacy while facilitating seamless data sharing among authorized parties. In the realm of digital identity [8], blockchain allows individuals to manage and verify their identities securely, mitigating identity theft and providing a foundation for self-sovereign identity systems. Additionally, decentralized finance (DeFi) has unlocked opportunities for peer-to-peer lending, automated trading, and tokenized assets, reshaping financial services. Moreover, in voting systems [23], blockchain's transparency and immutability enhance the integrity of electoral processes. These diverse applications illustrate blockchain's potential to revolutionize industries by providing secure, decentralized, and transparent solutions, far beyond its origins as a cryptocurrency ledger.

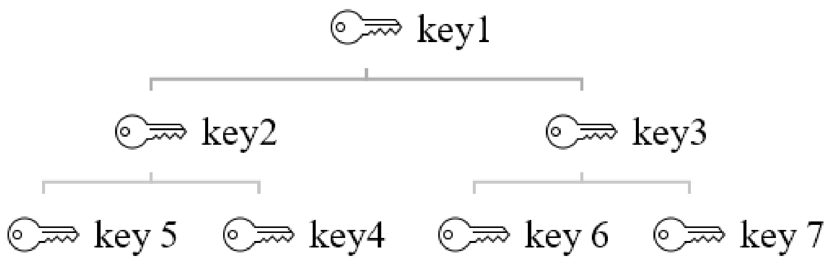

Figure 4: Representation of a queue

2 The Proposed Queue Mechanism

In this section, we present our proposed queue and how we aim to incorporate it using the existing blockchain structure.

2.1 The Queue

Our proposed changes derive their merit from the same source as [32] where it was shown that a queue holding the list of future transactions will improve the speed and accuracy of mining new blocks in mining pools. The key difference being that we keep a queue of future miners as opposed to future transactions.

Our proposed queue is a binary tree composed of the public keys of each one of the miners. Each time a miner successfully writes a block, he is popped out of the queue and is allowed one of two options. He may choose to write another miner at the tail of the queue or alternatively, he may write himself again into the queue, starting at the tail of the queue. Miners receive a reward for each block they mine in the same manner that Bitcoin rewards its miners [34]. Figure 4 shows such a tree, and Algorithm 2 shows how we can translate such a binary tree into a 1-dimensional array which we can add to each block as metadata (further explained in section 2.2 and shown in algorithm 2).

Algorithm 2 Child nodes function

```
1: function CHILDNODES(i)
2:     return $(2 * i) + 1, (2 * i) + 2$
3: end function
```

It is also important to look into where we rely on the current PoW algorithm [34] and its inner workings. Firstly, we will continue using the same network and message propagation methods. Changing the method by which other miners are discovered and by which information is shared in the blockchain is out of scope for this work though a simpler and faster method could be derived as part of the changes suggested in this work. Another important aspect where we rely on PoW is the structure of a block as described in [34]. We will suggest removing data that becomes obsolete by our new algorithm such as the Nonce but the structure of the block and the blockchain will remain the same as described in Figure 1. Furthermore we suggest the methods for rewarding miners, authenticating blocks and transactions remain the same as we do not see any benefit in changing them.

Another key part of our algorithm is the block publication and acceptance process that adds its own protections needed due to possible attacks (see details in section 3). As can be seen in Algorithm 3 the validation of new blocks is similar to [34] as it needs to validate the transactions and hash of the block but it also has to validate the queue publication (further explained in section 2.2). In addition a 10 minute time limit on new block publication is added as well as a method to deal with miners that disconnected

Algorithm 3 Accepting new blocks

```
if $publisher\_public\_key == queue[0]$ then
    if $block\_time - last\_block\_time < 10$ minutes then
        if validate transactions, block hash, and queue are all correct then
            accept block and provide miner with reward
        else
            discard the miner from the queue giving the next miner to propose miners in his place
        end if
    else
        assume miner lost his turn and move forward
    end if
else if $publisher\_public\_key < 50$ blocks after missed turn then
    if validate transactions, block hash, and queue is True then
        accept block and provide miner with reward
    else
        discard the miner from the queue giving the next miner to propose miners in his place
    end if
else if $publisher\_public\_key < 60$ blocks after missed turn then
    if validate transactions, block hash, and queue is True then
        run the lotto algorithm from proof of stake and accept block and provide miner with reward by
decision of its winner.
    else
        discard the miner from the queue giving the next miner to propose miners in his place
    end if
end if
recalculate the queue as needed with data from the block
```

or choose not to publish a block. Another method is introduced to accept out of order blocks in cases where a miner missed his turn which may happen due to malicious behavior and is further explained in section 3. Lastly, another key aspect of the algorithm is that blocks which fail the validation step invoke a punishment to their miner. The miner which published them is then removed from the queue and his place (or places) are provided to the next miner in line. Everyone in turn advance as if the malicious miner was promoted and the new replaces are added to the end of the queue.

Miners can advance in the queue if their parents have advanced. The form in which a miner is chosen to advance is the "distance" between the current block's hash published and the miner's public key. A lower distance means the miner moves up the tree. For further understanding of the protocol, please see Algorithm 4.

Lastly, it is important to note that our proposed improvements remain limited by the inherent speed at which messages propagate across blockchain networks. All ledger-based blockchain algorithms require nodes to verify the most recent block to prevent double-spending and validate correctness. This verification introduces a network-dependent delay that affects block publication times and cannot be fully eliminated by any consensus mechanism, including Proof of Work (PoW), Proof of Stake (PoS), or our proposed Proof of Queue (PoQ). While some prior research, such as the "Simulation Model for Blockchain Systems Using Queuing Theory" by Memon et al. [33], suggests methods to reduce delays, these solutions primarily address mining pools or trusted environments. Therefore, their applicability to fully decentralized and trustless public blockchains remains limited. Approaches like Delegated Proof of Stake (DPoS) could partially address message propagation issues but introduce significant security and centralization risks [35, 37], placing them beyond the scope of our current work.

Algorithm 4 Updating the queue

Description: This algorithm rebuilds a queue by traversing and reconstructing a tree. It assumes that each member has access to the last published block, which includes the following information:

- The publisher's public key or the public key of another miner.
- The block's hash.

It also requires the state of the tree as it existed prior to the publication of the block. The algorithm processes this information to rebuild the tree. At the end of its execution, the algorithm produces the updated tree, reflecting its state after the publication of the last block.

```
1: $Head \leftarrow$ top of the tree
2: Pop the top of the tree
3: Rebuild the tree with the following logic:
4: while $head.left$ and $head.right$ are not null do
5:     if $head.left.value - Last\_block > head.right.value - Last\_block$ then
6:         $Head.value \leftarrow head.right.value$
7:         $Head \leftarrow head.right$
8:     else
9:         $Head.value \leftarrow head.left.value$
10:        $Head \leftarrow head.left$
11:    end if
12: end while
13: $Head \leftarrow$ new_miner_public_key
```

2.2 Publishing the Queue

Definition 7. Genesis block The very first block upon which additional blocks in a blockchain are added.

In order to allow new miners to build the queue without following the chain to the Genesis block the algorithm attaches a part of the queue to each block. This section shows how we can attach the queue to each block and what steps the algorithm follows in order to achieve this task.

Each miner before releasing his block needs to follow the following set of instructions:

1. Turn the tree into an array (see Algorithm 2).
2. Look at the previous block and identify what is the index in the queue that was last published.
3. Attach to the block the $\frac{n}{1000}$ following places after that index looping to the beginning of the array when needed (n being the number of miners in the queue).

Algorithm 2 describes how we can turn a binary tree into a one-dimensional array that can be attached to each block.

Let us take, for example, the tree from Figure 4.

It is displayed as such in an array:

$$1 \quad 2 \quad 3 \quad 5 \quad 4 \quad 6 \quad 7 \quad \ldots$$

Therefore, the first block publishes 1,2 +(new_key), the next 3,5 + (new_key), the next 4,6 + (new_key), and so on.

2.3 Why Use a Binary Tree as Opposed To a Linked List/Array?

Definition 8. Linear queue A queue that follows the First In First Out (FIFO) method as depicted in Figure 5.

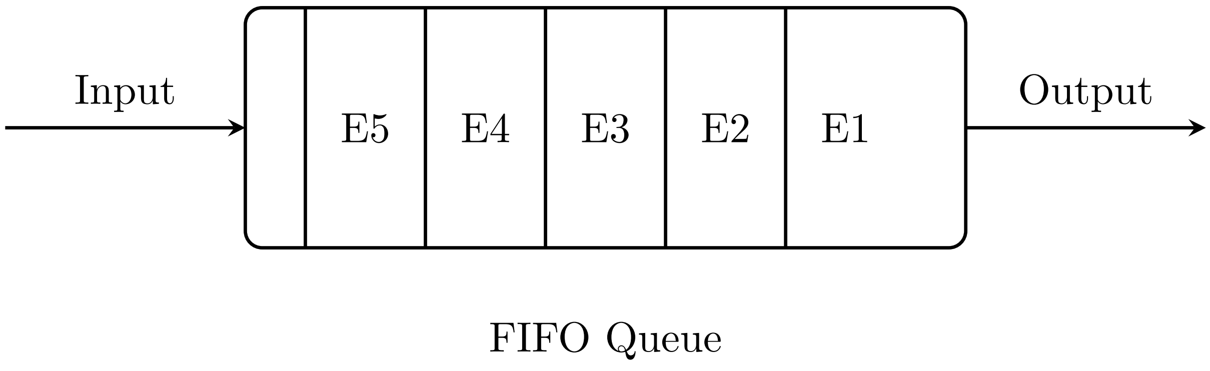

Figure 5: A FIFO Queue illustration, E5 is the last input to enter the queue and E1 is the first one to do so. E1 will be the first to pop out when the queue is full.

A linear queue does not require each miner to check the state of the blockchain actively as he knows the average speed of block publications and can cut costs by checking periodically to see if his turn is near thus saving computational resources. Though we aim to decrease the usage of CPU as opposed to Proof of Work such lethargic behavior poses a risk to both security and speed of mining as miners can miss their turn causing the line to hold until they check-in realize it is their turn and publish their block while failing to validate faulty or malicious blocks.

Theorem 2.1. Using a binary tree queue creates a strong incentive to continue and look for errors in the chain since each deviation from the algorithm will result in the block publisher being disqualified and the chain removing the attacking miner while keeping the bad block in the chain as evidence (further reducing forking in our algorithm).

Proof. The unpredictable nature of the advancement in the binary tree requires all miners to keep checking the blockchain to update their location in it. Since the hash of the block is unpredictable [41] a miner will have to check every block, recalculate its merkle tree [12], and validate its own location in the queue. This provides an incentive to not wait idly for one's turn but rather to continue and validate the blockchain process. □

Theorem 2.2. Using a non-linear queue improves the security of a blockchain without greatly impacting the speed in which we fix the queue after a 99% attack.

Proof. As seen in Theorem 2.1 a blockchain is continuously checked for faulty or malicious blocks, if one is discovered the miner who publishes this block is disqualified and is ejected from the queue as if he had successfully published a block only without the ability to add a new miner (as we do not want dishonest miners to add new miners to our queue). The next honest miner can add as many miners as were ejected as long as he can reference a faulty block that they published. As in a queue, there are a limited number of nodes and there is a limit on the time each node can wait until they publish their block (Theorem5.2) eventually if one exists even a single honest miner will be able to publish an honest block (see the section 5 for a more formal proof).

Now lets assume the worst case where the honest miner was the last one to enter our queue. In a linear queue we would have to wait for $(n-1)$minutes for the miner to reach the head of the queue (n being the number of miners). Consider a binary tree of height h, where the root is at level 1 and the bottom level is at level h. We are interested in a particular node at the bottom and want to determine the expected number of iterations required for this node to become the top node. To get our chosen bottom node to the top after h iterations, we must make the correct binary choice at each of the $h-1$

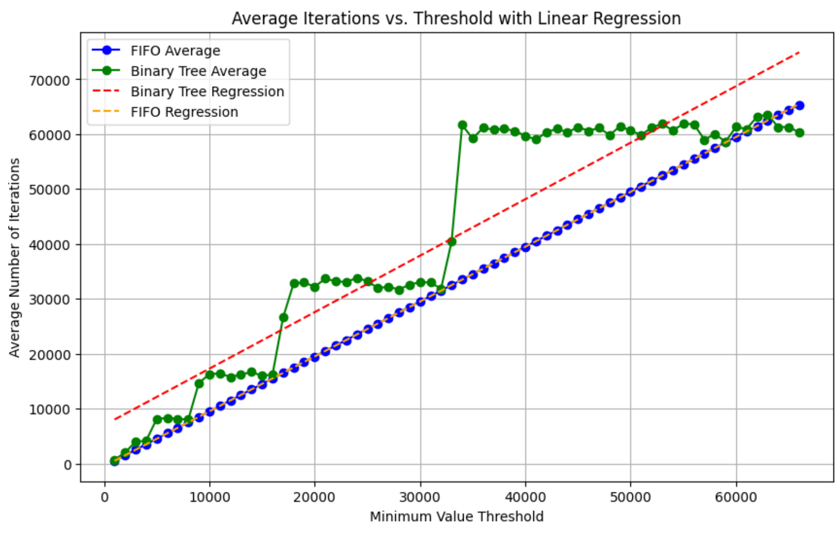

Figure 6: Representation of the average number of iterations it takes in average for a node within the 1000 range to reach the head of the line with FIFO queue and our binary tree.

intermediate nodes along its unique path making $h-1$ the best case. In the worst case it will be the last one chosen meaning it will still perform as well as the FIFO queue and the average case somewhere in between but still better than a FIFO queue.

We now analyze the average number of iterations required for the first honest miner to reach the head of the queue. Let X denote the miner's position in the queue, where he may be the first or the only one present. We compare two queue structures: a FIFO queue and a tree-based queue with 16 levels, containing $2^{16}-1=65,535$ nodes. The FIFO queue allows a miner in position X to reach the head of the queue.

To validate our findings, we conducted 50,000 independent and random iterations to demonstrate that our algorithm does not require exponential computing time. Instead, its performance is comparable to that of the FIFO queue in terms of the number of iterations needed for X to reach the head. The corresponding Python implementation, which can be found on GitHub [22], runs this test by tracking how many iterations a node at position X requires to reach the head. The results, along with each node's initial position, are recorded.

From this data, we generated Figure 6, which illustrates the average number of iterations required for successive groups of 1,000 nodes (e.g., 0–999, 1000–1999, etc.) to reach the head of the queue. Using a queue size of 2^{16} and 50,000 iterations, we applied linear regression to examine the relationship between the number of iterations needed to reach the head and the queue size, comparing both the FIFO and our proposed algorithm.

As can be seen in Figure 6 the FIFO queue has a gradient of approximately 1, meaning that when a random miner X is chosen in the queue it will take him (as expected) that exact number of iterations to reach the queues head. On the other hand when our Binary tree algorithm is used the gradient is approximately 1 as well, Meaning either methods do not provide an advantage one over the other when n the queues length is sufficiently large (as the key difference between them is the intercept).

$\square$

3 Possible Attack Vectors and Mitigations

In this section, we show different attack vectors aiming at hindering the implementation of queue mechanism protocols and the possible solutions.

3.1 Holding Up the Queue

Since validation of the blockchain using the block's hash before is still mandatory, holding up the queue [10] as a malicious actor remains the greatest threat to our proposed algorithm. Since no one can write ahead of the next in the queue, he can choose not to release any block, bringing the entire chain to hold still as can be seen in Figure 7.

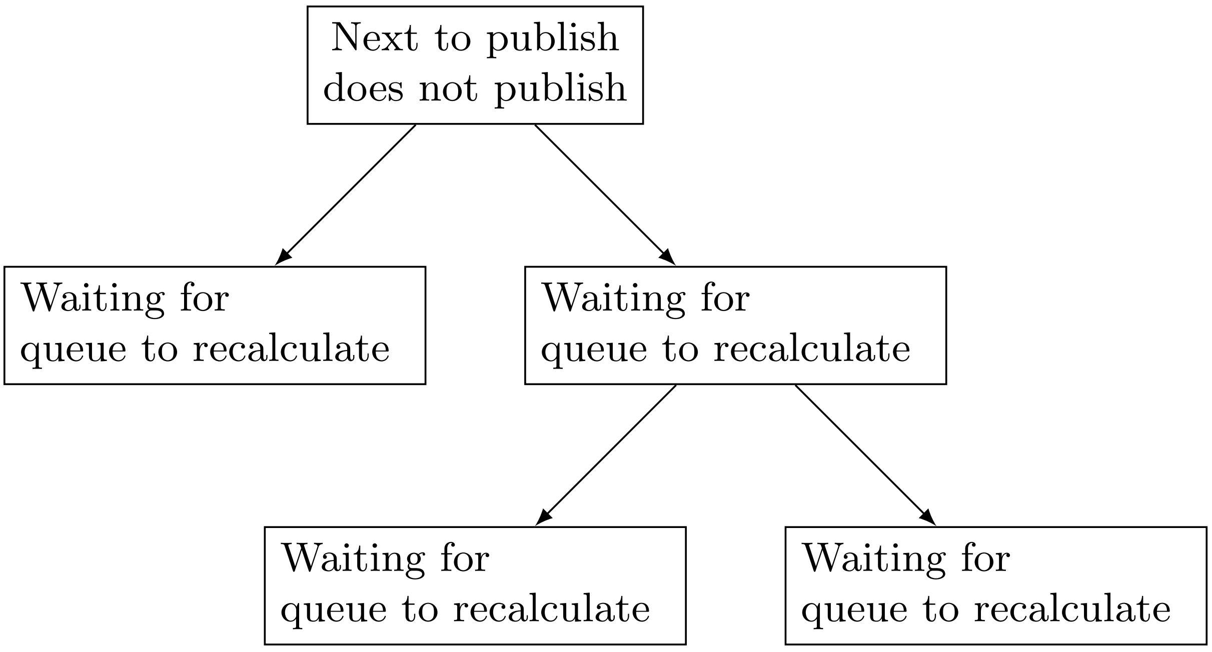
Next to publish does not publish
Waiting for queue to recalculate
Waiting for queue to recalculate
Waiting for queue to recalculate
Waiting for queue to recalculate

Figure 7: A malicious actor choosing not to publish creating a standstill as everyone is stuck in the following state.

Theorem 3.1. Our algorithm limits holdup attacks and improves speed of publishing new blocks.

Proof. If a block is not published within the next 10 minutes after the timestamp of the last block, the miner may be eliminated from the tree, thus releasing the holdup and allowing the chain to continue. Furthermore, there is no incentive to hold up the queue since each miner is rewarded with three incentives upon releasing its block. Since PoW is designed to produce a block once every 10 minutes we are at least as fast as PoW with the mean time being faster as a miner is rewarded only when publishing new blocks there is no incentive for delays.

1. He gains the authority to add a new miner to the tree.
2. He collects a premium for each transaction he writes.
3. He is given a certain number of coins for his role in the mining process, like Bitcoin awards for successful miners.

These incentives combined with the possibility of "punishment" are enough in theory to push miners to publish as soon as they can.

It is important to note however that a late published block is the only way to create a fork in the Proof of Queue algorithm. The forking issue exists with PoS (Proof of Stake) as well, Both of them allow miners with more power (one in the form of more funds, the other in the form of more slots in the queue) to mine more blocks. The only difference is that mining new blocks in PoS provides direct power to the miner through the creation of more funds, which he can use to gain more power and influence the chain in a greater way (through purchasing more validator slots). On the other hand, with queue-based mining, the miner does not gain any more power through the mining of new blocks since, in exchange for mining a new block, a miner should receive only a single new slot at the bottom of the queue and only in rare occasions will a miner receive the ability to fill more than one slot thus gaining more power over the queue.

□

More about fake time stamping can be found in the section 3.4.

3.2 Controlling 50 Percent of the Network

In Bitcoin, if a malicious actor controls 50 percent of the network, it is doomed. Since Bitcoin requires a majority to agree upon the next block, 50% of the computation power may choose what is correct and ignore malicious blocks(the same goes to some extent with PoS) [3]

Theorem 3.2. A queue algorithm allows even one honest miner to recover the entire chain.

Proof. This will happen by eliminating all the other miners that publish bad/illegal blocks or waiting elimination as discussed in section 2.4.1 holding up the queue. Then, the next honest miner can choose not to add just one miner at the end of his turn but instead add miners for all the malicious miners he can point out. This is done by following the algorithm and requires no outside intervention, just time as can be seen in the liveness section 5. □

3.3 Publishing Fake/Wrong Blocks

Theorem 3.3. Our algorithm can notice and react to fake/wrong blocks.

Proof. This is countered by everyone having a full report of the following things: the hash of the last block, the full tree of miners, and the timestamp of the last block. Using this, it is impossible to fake a block on the same premises as the PoW used by Bitcoin. □

3.4 Publishing Fake Timestamps

Our algorithm is extremely sensitive to fake/ incorrect timestamps as the protection against holding up the queue opens the door for a direct misuse of the timestamp as a malicious tool against other miners. This can in turn:

1. Cause the next in the queue to miss his. This is accomplished by a dishonest miner waiting till the end of our allotted time, then publishing a block stating he published it immediately after the last block was published. Thus, tricking all the honest miners to think that the next in line missed his turn.

2. Publishing blocks in a delay to slow down the network. A malicious actor may wait for the set number of blocks stated in the algorithm designed to deal with the issue above (Algorithm 7) and only then release his block. Causing the entire network to republish the blocks afterward.

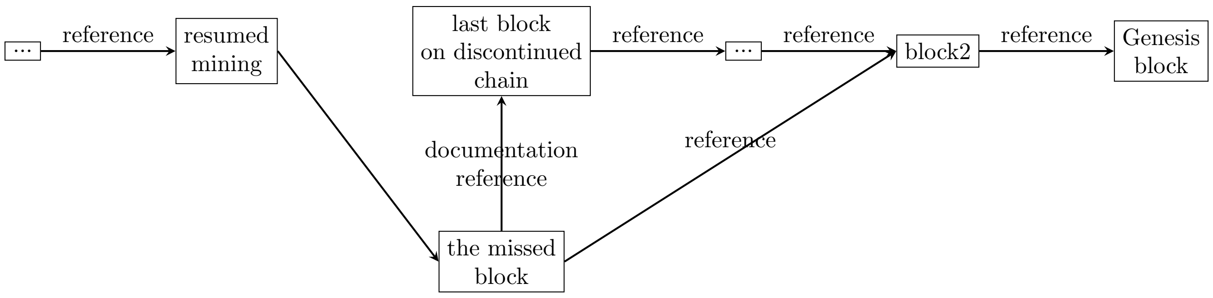

Figure 8: A depiction of a blockchain with an accepted missing block. Please notice the documented reference to the discontinued chain and the reference to the block from which the missing block was derived.

To deal with the issue we suggest Algorithm 5 which is displayed in Figure 8. By limiting the number of times each miner can invoke it we hope to eliminate the malicious incentive to do so. In addition, we believe the high publishing rate possible with Proof of Queue will discourage attackers from this form of attack as it can recover easily. Finally, the set number of blocks that allow contention will allow us to prevent double-spending as we can wait for that number of blocks to consider a transaction complete and final.

Theorem 3.4. Our algorithm can deal with fake/wrong timestamp publications.

Proof. 1. We allow every miner to contest the skip of his turn for up to a set number of blocks.

2. Not allowing a delayed miner to publish another delayed block for the next chosen number of blocks. Creating an incentive not to do so maliciously, or the actor might miss his next turn to mine. Since there is only a set number of miners (the size of the queue), missing an opportunity to mine is a big issue that may result in a great penalty, reducing the incentive to do so maliciously.

□

Thus we come to another conclusion, in PoS, forking becomes non-economical in the sense that mining on a number of different forks can result in a penalty. In queue-based mining, the miner can choose to withhold his transaction and later post it creating "wasted" blocks that will now lead to nowhere. Since this is mitigated through the improvement of mining speed and punishment of the miner, both methods are equally effective from our point of view.

Algorithm 5 Sending out-of-order blocks

```
1: if My turn was skipped then
2:     if The last block published <= 60 blocks after my turn then
3:         Publish a new block with a reference to the current last block
4:     end if
5: end if
```

4 Keeping Consensus

Consensus in distributed systems refers to the process by which multiple independent nodes or processes in a network agree on a single data value or course of action. This agreement is crucial for ensuring consistency across the system, particularly in environments where nodes may fail, messages may be delayed, or components may behave unpredictably. Therefor consensus is fundamental for maintaining the integrity and reliability of distributed applications, such as databases, blockchain networks, and collaborative systems, where coordinated decision-making is essential for correct operation [40]. In our context, consensus refers to all participants agreeing on the state of the queue. However, the algorithm allows for discrepancies to occur when a miner contests the skipping of their turn, which poses a challenge to maintaining consensus.

This means that while some members of the chain may receive a certain number of blocks prior to the contested block other members may have received a different number of blocks. This may happen due to delays in the network or block propagation working incorrectly.

For example lets consider a case where Alice, Bob and Carol are miners in our blockchain. Bob is directly connected to Carol while Alice is connected through a number of other Miners. When a contested block is sent by Alice and a normal block is sent by Carol at exactly the same time Bob and Alice wont have a consensus about the state of the blockchain. Bob will think that Carols block should be considered in the count for "Number of blocks missed" while Alice will dispute this.

To preserve consensus, we rely on the following assumptions:

1. Out-of-order blocks are only accepted if fewer than 60 blocks have been added since the dispute occurred (this number is suggested and not tested in real-life scenarios).

2. The miner in question did not previously send a block and is now attempting to modify it.

If these assumptions are met, the following steps are taken to maintain consensus:

1. If between 50 and 60 blocks have been mined, a lottery test from proof-of-stake algorithm is conducted between the publishing miners and the missed miner, as detailed in Algorithm 6. The winner of this lottery decides whether to allow or reject the publication of the contested block.

2. To keep consensus each miner can call for either a lottery or disqualification on the basis of the number of published blocks. This long requires him to provide a reference to the block he is mentioning (in the form of its hash and timestamp). This will prompt everyone to request that blocks hash and verify if the conditions for a lottery or disqualification have been met. This allows us to deal with cases where some nodes have yet to receive the exact block that crosses the threshold.

3. If fewer than 50 blocks have passed, all miners will accept the new block automatically and mining will resume as normal from the new position 5.

4. If it was determined that the disputed block should be disqualified than the miner is removed from the queue allowing the next miner in line to fill the missing positions with miners of his choosing.

The steps above are shown in the algorithm 7. This in turn helps us to provide the needed consensus for our network as it allows every and all members to agree/ disagree on whether a block should be accepted out of turn and if not provides an explanation to the reason.

Algorithm 6 Proof of Stakes lottery like algorithm

```
Input: List of miners with their public keys $\{(S_1, key_1), (S_2, key_2), \ldots, (S_n, key_n)\}$
Input: hash value of the block that is contested$H_{ref}$
Output: Selected validator $S_{winner}$

Initialize: $min\_offset \leftarrow \infty$
$S_{winner} \leftarrow$ null
for each miner $\{(S_i, key_i)\}$ in the list of miners do
    $H_i \leftarrow \text{Hash}(key_i)$                        ▷ Compute the hash of the staker
    $offset_i \leftarrow |H_i - H_{ref}|$                ▷ Calculate the offset from the reference hash
    if $offset_i < min\_offset$ then
        $min\_offset \leftarrow offset_i$
        $S_{winner} \leftarrow S_i$
    end if
end for
return $S_{winner}$                  ▷ Return the miner with the closest key as the validator
```

Algorithm 7 Accepting out-of-order blocks

```
if verify block correctnes and transactions are valid then
    if new recived block out of order then
        if publishers original position < 50 blocks after current block then
            Continue the chain from that location
        else if publishers original position < 60 blocks after current block then
            send a call for lottery with the hash of the latest known block and a timestamp showing it was published prior to the contested block to determine if to accept it or not and run 6
        else
            Send a disqualification notice with the hash of the latest known block and a timestamp showing it was published prior to the contested block
            Remove the disqualified blocks miner from the queue
        end if
    end if
end if
```

5 Proving Liveness

In [27] it is stated that liveness is a property that stipulates that a 'good thing' happens during execution. The paper [2] further expands on the idea giving us an example of liveness as "guaranteed service", meaning "every request for service is satisfied eventually, the 'good thing' is receiving service. " or as "starvation freedom, which states that a process makes progress infinitely often, the 'good thing' is making progress.".

To prove liveness we will prove the following theorems:

Theorem 5.1. The Queue will allow everyone in line to participate providing liveness as guaranteed service.

Proof. Given that the hash of a block is unknown and random and that public keys are created in the same manner producing a random number then letting α be the hash of the last block and β_1 being the value of public key 1 and β_2 being the value of the second pubic key, then they have the same probability to be chosen. This means each node's right or left child has the same probability of advancing. For a perfect binary tree with N nodes (every node has 0 or 2 children and all leaves have the same level),N is:

$$N = 2^{h+1} - 1$$

To prove each miner will eventually be chosen we calculate the expectancy of a miner to reach the root of the binary tree after a set number of iterations. Each iteration of our algorithm consists of:

1. Removing the root node (level 1).
2. Replacing that vacant position by *one* of its two children (with probability 1/2 each).
3. Continuing this random replacement procedure down the tree: whenever a node is chosen to move up, its old position is vacated and then filled by one of its two children with probability 1/2 each, and so on.
4. At the bottom level (level H), the vacated leaf is replaced by a *new* node so that the tree remains a complete binary tree.

Thus, in each iteration exactly one random root-to-leaf path is chosen (each left/right choice is 1/2). Every node on that path "bubbles up" one level, and the leaf at the end of that path gets replaced.

Goal: Determine the expected number of such iterations before our tracked leaf (initially at level H) is chosen in a sequence of moves all the way to level 1 (the root) for the *first time.*

If at some point our tracked node is at level k $(1 < k \leq H)$, then the probability that the random root-to-leaf path passes through that node is

$$\left(\tfrac{1}{2}\right)^{k-1},$$

since a path from the root (level 1) down to level k has exactly $(k-1)$ independent left-or-right choices, each with probability 1/2.

Let T_k denote the expected number of iterations for the tracked node to go from level k to level 1. Clearly, $T_1 = 0$ (if the node is already the root, we are done). For $k \geq 2$, in each iteration:

With probability $\left(\frac{1}{2}\right)^{k-1}$, the node is chosen and moves up from level k to level $k-1$.

With probability $1 - \left(\frac{1}{2}\right)^{k-1}$, the node stays at level k.

Hence, the standard expectation argument (for a geometric waiting time in each level-up step) gives

$$T_k \;=\; 1 \;+\; \left(\tfrac{1}{2}\right)^{k-1} T_{k-1} \;+\; \left[1-\left(\tfrac{1}{2}\right)^{k-1}\right] T_k.$$

Rearranging,

$$T_k \;-\; \left[1-\left(\tfrac{1}{2}\right)^{k-1}\right] T_k \;=\; 1 \;+\; \left(\tfrac{1}{2}\right)^{k-1} T_{k-1},$$

$$\left(\tfrac{1}{2}\right)^{k-1} T_k \;=\; 1 \;+\; \left(\tfrac{1}{2}\right)^{k-1} T_{k-1} \quad\Longrightarrow\quad T_k \;=\; \frac{1}{\left(\tfrac{1}{2}\right)^{k-1}} \;+\; T_{k-1} \;=\; 2^{k-1} \;+\; T_{k-1}.$$

Given $T_1 = 0$, we therefore obtain

$$T_k \;=\; \sum_{j=2}^{k} 2^{\,j-1} \;=\; \sum_{n=1}^{k-1} 2^n \;=\; 2^k \;-\; 2.$$

For a node initially at level H, its expected time T_H (in iterations) to *first* reach the root (level 1) is

$$\boxed{T_H \;=\; 2^H \;-\; 2.}$$

and since,

$$H \;=\; \log_2(N+1).$$

For a complete binary tree with N nodes,

$$\boxed{T(N) \;=\; N \;-\; 1.}$$

since this is a polynomial run time is a function of N we assume that each miner will be chosen eventually making this an algorithm that stands in the requirement for liveness as a guaranteed service. □

Theorem 5.2. The Queue will always allow the publishing of new blocks proving liveness in the form of starvation-freedom.

Proof. In each iteration a block is created, given that said block can be either accepted or rejected we need to prove that eventually (in polynomial time) a non-faulty block will be created allowing us to further the chain. Given that we proved in theorem 3.2 that given a single honest miner the network will produce within a polynomial time a new "correct" block that case is already proved. In the case of no honest miners each block can become a non-faulty one guaranteeing that the chain will grow and produce new blocks. □

6 Conclusion and Future Work

In the work, we present a new and novel algorithm that solves the Byzantine issue underlining blockchain. The algorithm named PoQ (Proof of Queue) uses the randomized nature of both the hash function and the public key of each miner to decide upon the next miner out of a set queue that acts as a binary tree. We saw how using a binary tree in the queue solves many security concerns plaguing other blockchain algorithms and how we mitigate these concerns. Lastly, we defined and proved liveness and consensus in the context of our algorithm and laid the ground for future work needed to further develop the algorithm.

We list here some directions for future research:

1. Is it possible to incorporate a queue for transactions into the algorithm to make it even faster?

2. What improvements would such a solution offer over other algorithms in a real-world scenario?

3. What is the optimal delay for accepting out-of-order blocks?

4. What is the best size for the queue?

5. What is the ideal size of the queue that can be published with each block?

6. Should we add a randomization method to the process of choosing the next miners who will have an incentive to claim keys close to the middle to enhance their success rate (games theory - Ice cream vendor problem)?

References

[1] Tron: Advanced decentralized blockchain platform, 2017.

[2] Bowen Alpern and Fred B. Schneider. Defining liveness. *Department of Computer Science, Cornell University*, 1985.

[3] Fredy Andres Aponte-Novoa, Ana Lucila Sandoval Orozco, Ricardo Villanueva-Polanco, and Pedro Wightman. The 51% attack on blockchains: A mining behavior study. *IEEE Access*, 2021.

[4] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. January 2017.

[5] Wei Bi, Huawei Yang, and Maolin Zheng. An accelerated method for message propagation in blockchain networks. *Seele Tech Corporation*, 2024.

[6] BitShares. Delegated proof of stake (dpos), n.d.

[7] Vitalik Buterin and Gavin Wood. Ethereum: A next-generation smart contract and decentralized application platform, 2014.

[8] Alexandru-Cristian Careja and Nicolae Tapus. Digital identity using blockchain technology. *Procedia Computer Science*, 2023.

[9] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. February 1999.

[10] Nicolas T. Courtois and Lear Bahack. On subversive miner strategies and block withholding attack in bitcoin digital currency. January 2014.

[11] W. Dai. b-money. 1998.

[12] Haitz Sáez de Ocáriz Borde. An overview of trees in blockchain technology: Merkle trees and merkle patricia tries. Department of Engineering, University of Cambridge, February 2022.

[13] John R. Douceur. The sybil attack. 2002.

[14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *arXiv preprint arXiv:1311.0243*, 2013.

[15] Minghong Fang and Jia Liu. Toward low-cost and stable blockchain networks. July 2020.

[16] Hal Finney. Reusable proofs of work, 2005.

[17] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[18] Vishal Gaur and Abhinav Gaiha. Building a transparent supply chain: Blockchain can enhance trust, efficiency, and speed. *Harvard Business Review*, May–June 2020.

[19] Abid Haleem, Mohd Javaid, Ravi Pratap Singh, Rajiv Suman, and Shanay Rab. Blockchain technology applications in healthcare: An overview. *Materials Today: Proceedings*, 2021.

[20] Himanshi. Byzantine fault tolerance (bft) in blockchain. January 2023.

[21] Ziad Hussein, May A. Salama, and Sahar A. El-Rahman. Evolution of blockchain consensus algorithms: A review on the latest milestones of blockchain consensus algorithms. *Cybersecurity*, 6(30), 2023.

[22] Jonathan Shai Isakov. Proof-of-queue, 2025. GitHub Repository: https://github.com/jonisakov/Proof-of-Queue.

[23] Beulah Jayakumari, S Lilly Sheeba, Maya Eapen, Jani Anbarasi, Vinayakumar Ravi, A Suganya, and Malathy Jawahar. E-voting system using cloud-based hybrid blockchain technology. *Journal of Cloud Computing*, 2024.

[24] Wei Jiang, Xiaoyan Wu, Minghao Song, Jian Qin, and Zhiwei Jia. Improved pbft algorithm based on comprehensive evaluation model. *Applied Sciences*, 13(2):1117, 2023.

[25] JosephC. The beacon chain ethereum 2.0 explainer you need to read first. December 2022.

[26] Sunny King and Scott Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. aug 2012.

[27] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, March 1977.

[28] Daniel Larimer. Delegated proof-of-stake (dpos) consensus, 2014.

[29] Cristian Lepore, Michela Ceria, Andrea Visconti, Udai Pratap Rao, Kaushal Arvindbhai Shah, and Luca Zanolini. A survey on blockchain consensus with a performance comparison of pow, pos and pure pos. *Mathematics*, 8(10):1782, 2020.

[30] Marshall Pease Leslie Lamport, Robert Shostak. The byzantine generals problem. July 1982.

[31] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S. Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 919–927. IFAAMAS, International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[32] Quan-Lin Li, Jing-Yu Ma, and Yan-Xia Chang. Blockchain queueing theory, 2018.

[33] Raheel Ahmed Memon, Jian Ping Li, and Junaid Ahmed. Simulation model for blockchain systems using queuing theory. February 2019.

[34] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. March 2009.

[35] Bradley Peak. Delegated proof-of-stake (dpos), explained, 2023.

[36] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[37] Gaurav Roy. What is delegated proof-of-stake (dpos)?, 2022.

[38] Han Song, Zhongche Qu, and Yihao Wei. Advancing blockchain scalability: An introduction to layer 1 and layer 2 solutions. *arXiv preprint arXiv:2406.13855*, 2024.

[39] The Investopedia Team. 51% attack: Definition, who is at risk, example, and cost, 2024.

[40] J. Turek and D. Shasha. The many faces of consensus in distributed systems. *Computer*, 25(6):8–17, jun 1992.

[41] Vladimir Omar Calderon Yaksic. A study on hash functions for cryptography. 2003.

[42] Zibin Zheng and Shaoan Xie. Blockchain challenges and opportunities: a survey. October 2018.

תקציר

במהלך העשור האחרון, טכנולוגיית הבלוקצ'יין ומטבעות הקריפטו הפכו למונחים שגורים בשיח הציבורי. השימוש במטבעות מבוזרים, כגון ביטקוין, מאפשר ביצוע עסקאות פיננסיות ללא צורך בגורם מתווך או גוף ריבוני, באופן המבטיח פרטיות, עצמאות ואבטחת מידע. עם זאת, הבסיס שעליו נשענות רשתות אלו – אלגוריתמי הקונצנזוס – סובל ממגבלות מהותיות. האלגוריתם הפופולרי ביותר Proof of Work (PoW) , שהוצג לראשונה על ידי סטושי נקאמוטו במסגרת ביטקוין, דורש השקעת משאבי חישוב אדירים לצורך השתתפות בקונצנזוס. שיטה זו מובילה לבזבוז אנרגיה, לקצב איטי של אישור בלוקים, לרגישות לפיצולים (forks) ולעמידות חלקית בלבד בפני מתקפות רוב.

לאורך השנים הוצעו אלטרנטיבות ל־PoW כגון Delegated Proof of Stake, Proof of Stake ו־PBFT אך גם אלו סובלות ממורכבויות, נקודות תורפה או ויתורים על ביזור אמיתי. מטרת עבודה זו היא להציע אלגוריתם קונצנזוס חדש בשם Proof of Queue (PoQ), שמטרתו לספק מענה למגבלות הקיימות תוך שמירה על רשת פתוחה, מבוזרת וללא אמון הדדי בין משתתפיה.

ב־PoQ, תהליך בחירת כותב הבלוק הבא נעשה באופן דטרמיניסטי מתוך תור ציבורי ומבוזר המאורגן במבנה של עץ בינארי. כל משתתף המעוניין לכרות בלוקים נרשם מראש לתור באמצעות מפתח ציבורי, והסדר שבו הם יסודרו בתור נקבע לפי שילוב של מיקום קודם בתור וערך הגיבוב (hash) של הבלוק האחרון. שיטה זו מבטלת את הצורך בתחרות חישובית, מאפשרת חיזוי זהות הכורה הבא, ומונעת לחלוטין מצב של פיצול בשרשרת הבלוקים, שכן סדר כריית הבלוקים ידוע מראש.

בנוסף PoQ, מטפל במתקפות רוב בדרך חדשנית: גם אם רוב התור מאויש על ידי משתתפים זדוניים, משתתף הגון אחד בלבד יוכל עם הזמן להסירם מהתור, להחליפם במשתתפים ישרים ולהשיב את תקינות הרשת. השיטה כוללת מנגנון קניסה למי שאינו פועל לפי הכללים, תוך שמירה על שקיפות מלאה ויכולת אימות על ידי כלל המשתתפים.

תוצאות המחקר מראות כי PoQ מציע שיפור משמעותי ביעילות, במהירות אישור בלוקים, בצריכת משאבים, וברמת האבטחה – וכל זאת מבלי לוותר על עקרונות הביזור. אלגוריתם זה עשוי להוות בסיס לדור הבא של רשתות בלוקצ'יין ציבוריות, המשלבות בין אמינות גבוהה לבין נגישות והשתתפות רחבה.

המכללה האקדמית תל אביב-יפו

בית הספר למדעי החשב

הוכחה בצורת תור: מנגנון יעיל ובטוח להסכמה עבור רשתות בלוקצ'יין

אפריל 2025

חיבור זה הוגש כחלק מהדרישות לקבלת התואר "מוסמך" – M.Sc.

במכללה האקדמית תל אביב-יפו

על ידי

יונתן שי איזקוב

העבודה הוכנה בהדרכתה של

ד"ר מור פרי