Data Science and Engineering
https://doi.org/10.1007/s41019-024-00251-0

RESEARCH PAPER

Erdos: A Novel Blockchain Consensus Algorithm with Equitable Node Selection and Deterministic Block Finalization

Buti Sello[1] · **Jianming Yong**[2] · **Xiaohui Tao**[3]

Received: 18 February 2024 / Revised: 18 April 2024 / Accepted: 23 May 2024

Abstract
The introduction of blockchain technology has brought about significant transformation in the realm of digital transactions, providing a secure and transparent platform for peer-to-peer interactions that cannot be tampered with. The decentralised and distributed nature of blockchains guarantees the integrity and authenticity of the data, eliminating the need for intermediaries. The applications of this technology are not limited to the financial sector, but extend to various areas, such as supply chain management, identity verification, and governance. At the core of these blockchains is the consensus mechanism, which plays a crucial role in ensuring the reliability and integrity of a system. Consensus mechanisms are essential for achieving an agreement amongst network participants regarding the validity of transactions and the order in which they are recorded on the blockchain. By incorporating consensus mechanisms, blockchains ensure that all honest nodes in the network reach a consensus on whether to accept or reject a block, based on predefined rules and criteria. The aim of this study is to introduce a novel consensus mechanism named Erdos, which seeks to address the shortcomings of existing consensus algorithms, such as the Proof of Work and Proof of Stake. Erdos emphasises security, decentralisation, and fairness. One notable feature of this mechanism is its equitable node-selection algorithm, which ensures equal opportunities for all nodes to engage in block creation and validation. In addition, Erdos implements a deterministic block finalisation process that guarantees the integrity and authenticity of the blockchain. The main contribution of this research lies in its innovative approach to deterministic block finalisation, which effectively mitigates the various security risks associated with blockchain systems.

Keywords Blockchain deterministic finality · Blockchain fork-resistance · Blockchain security · Decentralised ledger technology

1 Introduction

The paradigm shift brought about by blockchain technology cannot be overlooked in our understanding of digital trust and transactional transparency. Central to this groundbreaking technology is the pivotal concept of the consensus algorithms. These algorithms play a crucial role in enabling disparate and decentralised nodes to reach a consensus on the state of a blockchain ledger, thereby guaranteeing the accurate and honest confirmation of transactions and the proper addition of blocks. This guarantees the immutability of transactions.

The utilisation of blockchain technology extends beyond the realm of cryptocurrency, encompassing various domains, such as supply chain management, identity verification, governance, and numerous other sectors. This broad applicability is due to the robust consensus algorithms employed by blockchain, such as Proof of Work (PoW) and Proof of Stake

✉ Buti Sello
Buti.Sello@unisq.edu.au

Jianming Yong
Jianming.Yong@unisq.edu.au

Xiaohui Tao
Xiaohui.Tao@unisq.edu.au

[1] School of Business, University of Southern Queensland, West St, Toowoomba, QLD 4350, Australia

[2] School of Business, University of Southern Queensland, Sinnathamby Blvd, Springfield, QLD 4300, Australia

[3] School of Mathematics, Physics and Computing, University of Southern Queensland, West St, Toowoomba, QLD 4350, Australia

Published online: 06 June 2024

(PoS), which guarantee enhanced security while simultaneously ensuring efficient transaction processing within the network. Nevertheless, the existing consensus algorithms suffer from certain limitations. For instance, the Proof of Work algorithm is not only energy-intensive but also susceptible to centralisation, posing significant environmental and security risks. On the other hand, the Proof of Stake algorithm presents challenges related to the concentration of wealth. These drawbacks impede widespread network participation and prevent the attainment of a harmonious equilibrium among decentralisation, security, and scalability within the realm of blockchain technology.

To address these obstacles, we proposed a novel consensus mechanism called Erdos. This mechanism prioritises security, decentralisation, and fairness, and its comparison with PoW and PoS is shown in Fig. 1. It incorporates a node-selection method that promotes equal opportunities for all nodes to engage in block creation and validation. In addition, it implements a deterministic process for finalising the blocks, ensuring the integrity and authenticity of the blockchain. This study makes a significant contribution to the field of blockchain technology by introducing a novel method for achieving deterministic block finalisation. This approach effectively mitigates various security risks inherent in blockchain systems.

This study has a clear organizational structure. The introduction is followed by Sect. 2, which offers a comprehensive background and examines previous research on blockchain, consensus mechanisms, and related challenges. Section 3 provides an in-depth overview of the Erdos Consensus, and Sect. 4 discusses the intricacies of the Erdos architecture. Section 5 presents a thorough discussion of our research and future work, followed by the conclusion.

2 Related Work

2.1 An Overview of Blockchain Technology

Blockchain, or Distributed Ledger Technology (DLT), commonly associated with Bitcoin, introduced in 2008 by Satoshi Nakamoto, is a tamper-proof distributed ledger system known for its decentralisation, immutability, and transparency. Its utility has rapidly evolved from its initial use in cryptocurrencies to a versatile tool applicable to diverse sectors, including healthcare, banking, food safety, and supply chain management [7]. There are various types of blockchains, such as public, private, and consortium blockchains, each characterised by unique access controls and governance models. There is further categorisation, namely permissioned or permissionless blockchains. They differ fundamentally in terms of accessibility and governance mechanisms. Permission blockchains, typically private or consortium blockchains, restrict network access only to authorised participants and manage consensus through a limited number of nodes, thereby enhancing scalability, efficiency, and supervisory control [10].

However, permissionless blockchains are open to anyone where individuals can participate in consensus and validate transactions without prior permissions, making them capable of being highly decentralised [31]. While open participation promotes a higher level of privacy and decentralisation, as it removes the necessity of a third-party authority to ensure the integrity of the chain, it also presents challenges in a trustless environment. The absence of centralised oversight to enforce flexible and adaptive controls, such as security, scalability, and general governance, leads to difficulties in updating and upgrading systems efficiently, even

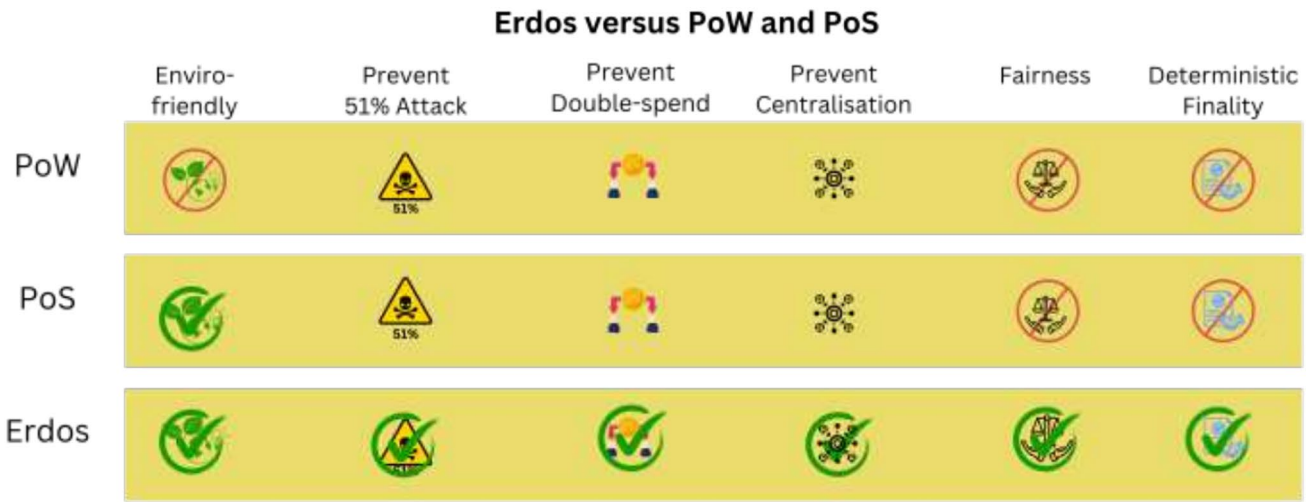

Fig. 1 Erdos versus PoW and PoS

hindering rapid evolution, giving rise to the need for some form of autonomous governance across the network [30]; hence, blockchain consensus mechanisms have emerged to fill this gap.

2.2 Consensus Mechanisms

A consensus algorithm is pivotal in blockchain systems as it serves as a collaborative framework that confirms the legitimacy and order of transactions without the need for a central authority [2]. Various consensus algorithms, such as Proof of Work (PoW) and Proof of Stake (PoS), possess distinct advantages and drawbacks, impacting aspects such as security, throughput, scalability, latency, and energy efficiency. They are critical for achieving agreement across distributed systems, particularly in the presence of failures or malicious actors [29]. Consensus mechanisms allow decisions and agreement on the state of the system, despite such adversities. By incorporating consensus mechanisms, the blockchain ensures that every honest node in the network agrees on the acceptance or rejection of a block based on predefined rules and criteria [10]. These mechanisms also serve the purpose of determining which node will have the authority to add the next block to the blockchain and receive a reward for this service to the network. This reward is the key to ensuring the liveness of the chain. The concept of liveness in consensus mechanisms refers to the ability of the system to continue to operate and make progress, ensuring that transactions and operations are eventually completed despite issues such as network delays or node failures. Liveness is highlighted as one of the fundamental components of any consensus algorithm [5, 8, 29]. This component is crucial for indicating that all nodes in the network are capable of reaching a consensus and confirming transactions within a reasonable timeframe, ensuring that the system remains active and responsive. Liveness, along with safety, agreement, termination, and fault tolerance, form the backbone of a robust consensus algorithm which is necessary for maintaining the overall health and effectiveness of blockchain applications.

2.2.1 Fairness

However, the reward introduces the problem of fairness and equity in the right to produce and add a block to the blockchain and receive rewards for one's efforts. Sahin et al. asserted that fairness and equity in the incentive structure of a consensus mechanism are core to the cardinal principles of blockchain [23]. For example, in PoW, mining a block requires a miner to solve a difficult cryptographic puzzle. This requires significant computational power, thereby making it difficult for a solo miner with limited computing power to obtain the opportunity to mine a block. This leaves only those that can afford powerful and expensive computing power with an unfair advantage [26]. The same applies to PoS as participants who have financial resources to afford putting forth a higher stake are also advantaged above those with fewer resources bringing us to the old adage "the rich get richer". The excessive computational power required for PoW also presents the issue of energy consumption.

2.2.2 Energy Consumption

The Proof of Work (PoW) consensus mechanism, which is one of the most commonly utilised in blockchain technologies such as Bitcoin, dominates the energy consumption in cryptocurrencies owing to factors such as redundant operations, inefficient mining devices, and the type of energy sources [14] and is particularly energy-intensive. This is because of the requirement for extensive computational effort to solve complex mathematical problems, which in turn results in high electricity utilisation [2, 15].

2.2.3 Centralisation

Consensus mechanisms play pivotal roles in how centralized or decentralized a blockchain network is. For instance, the analysis by Rebello et al. highlights the security vulnerabilities associated with consensus mechanisms, such as susceptibility to 51% attacks in PoW or the potential for collusion in PoS systems, and stresses that these vulnerabilities can be exacerbated by the degree of centralisation in the network. For example, if a small number of participants control significant computing power in PoW or hold large stakes in PoS, the network becomes more centralised and, thus, more vulnerable to attacks [20]. Although centralised mechanisms can provide some benefits in scalability and transaction throughput, it comes at the cost of distributed trust and security that the blockchain is meant to provide [9, 11, 27, 28]. Susceptibility to threats, such as the aforementioned 51% attack, is non-negligible when considering the possible enormity of the threat.

2.2.4 51% Attack

A 51% attack occurs when an entity gains control over more than 50% of a blockchain's hashing power, allowing them to manipulate the network by double-spending coins or preventing other miners from confirming transactions [1].This type of attack can be especially detrimental, as it also allows attackers to fork the blockchain and create a private version,

which can undermine the integrity and trust of the blockchain network [13]. These attacks can significantly devalue the associated cryptocurrency as trust in its stability and security is diminished [22].

2.2.5 Double Spend

Double spending can occur in several ways, but notably via a Race Attack or 51% attack.

In a Race Attack, the attacker aims to use the same currency token for multiple recipients, similar to signing a check for two individuals using the same available funds. This attack can succeed, for example, because Bitcoin's PoW has a 10 min time lag to confirm a transaction. On average, Bitcoin adds a new block within approximately 10 min [4, 16]. Within that time, transactions that carry higher transaction fees are prioritised by miners when selecting transactions from the transaction pool to be included in a new block. This presents a ripe opportunity for race attacks [24].

The anatomy of this attack is as follows: Suppose an attacker/user only has $ 10 balance in their blockchain wallet and initiates a transaction with a merchant, for example, a coffee shop, worth $ 10. The transaction is broadcast to the blockchain, and is thus placed in an unconfirmed transaction pool. At the same time, using the Unspent Transaction Output (UTXO) method initiates another transaction for the same $ 10 but makes this transaction more attractive to miners by ensuring that the transaction fees are much higher than the original transaction to the merchant. In the UTXO model, a transaction output is considered "unspent" until it is used as an input in a new transaction. In this case, both transactions end up in the unconfirmed transaction pool. Miners are more likely to select the second transaction because they have higher fees and include it in the next block [12]. The merchant, who may not have waited for transaction confirmation, would have proceeded to provide the goods "purchased". On the 10 min time elapse for the new block to be added to the chain, the fraudulent transaction has been

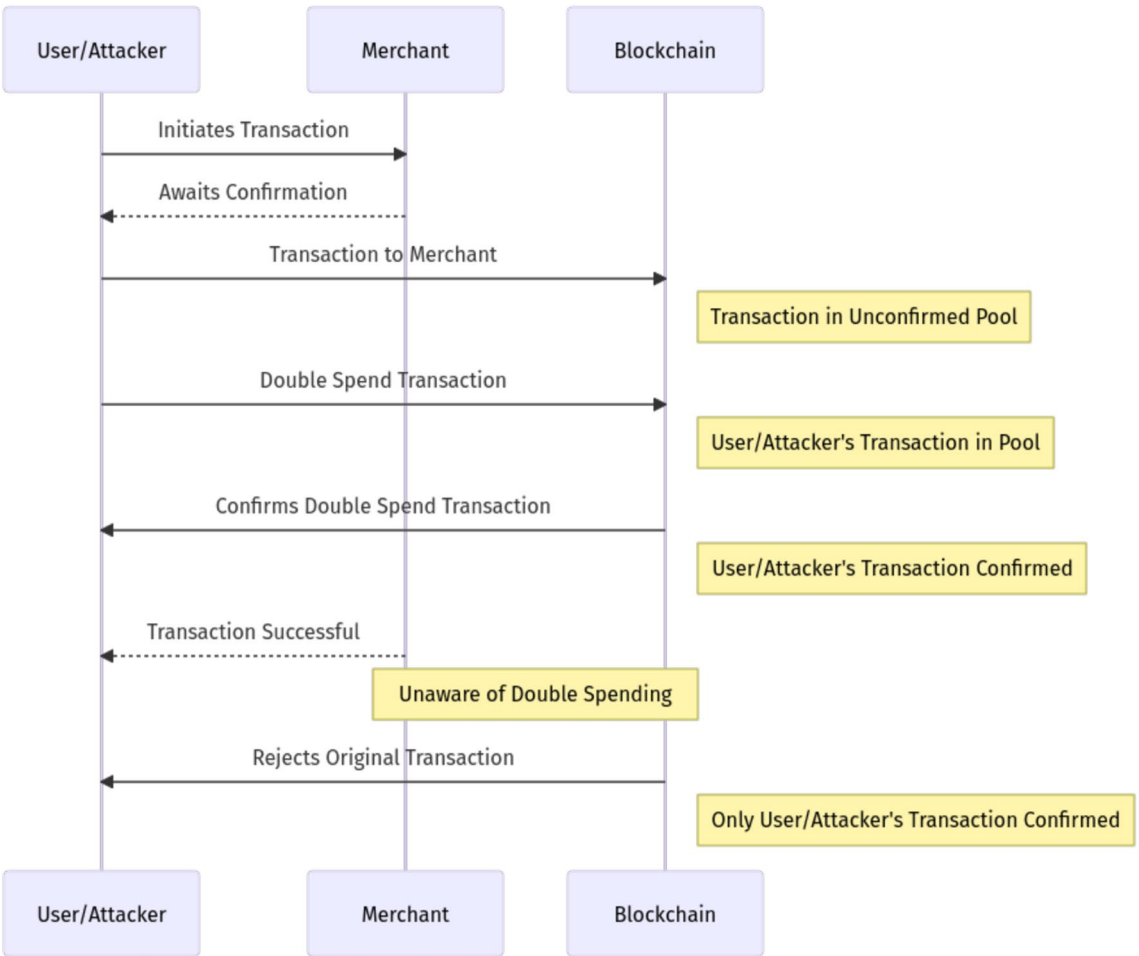

Fig. 2 Double-spend example

included in the block. Eventually, the original transaction to the merchant makes it into being selected for inclusion in the next block, and it fails to validate regardless of the fact that goods or services may have already been issued as shown in Fig. 2.

As mentioned earlier, the 51% attack has many serious security issues for blockchains but also provides further vulnerability with regard to double spending [3]. In the context of double spending, the mechanism of attack can be conducted as follows: the attacker or a group of attackers aims to acquire over 50% of the network's hash computing power. It is worth noting that this proves challenging in large blockchain networks because it is costly to gain over 50% of the hash power. However, the smaller chains are particularly vulnerable. After gaining 50% of hash power, attacker/s initiates a legitimate transaction to a merchant or other party. This transaction is processed and confirmed normally in a new block. Because attacker/s have more computing power in the network, they can begin a secret fork, that is, mining blocks in secret, thus creating a private fork of the chain [1]. In this secret private fork, they can include transactions that can send the already spent currency back to themselves, in essence, reversing the original transaction. This chain can grow faster to become the longest chain because attacker/s possess more computing power than the rest of the network [17]. Because the PoW protocol dictates that where there are multiple forks of a chain, the longest chain is the valid one [6, 25], the secret, fraudulent fork will be adopted as the legitimate fork, invalidating the original legitimate transaction.

2.2.6 Longest Fork and Probabilistic Block Finalisation

The feature of immutability is core to blockchain technology. Probabilistic block finalisation, made possible by the longest fork issue, goes against the core principle of immutability in the blockchain technology.

The longest-fork issue arises when multiple blocks are mined simultaneously, resulting in the creation of multiple competing chains. Risks include selfish mining, block orphanings, and chain reorganisation, often referred to as chain reorg issues.

Selfish mining is a strategy in which miners attempt to gain an advantage by keeping new blocks private instead of immediately broadcasting them to the network. By doing so, they can potentially increase their chances of mining the next block, leading to a longer chain and higher reward. Block orphaning, on the other hand, is a situation where a valid block, which was added to a forked chain, becomes obsolete and is discarded because another valid block on a different fork chain becomes dominant.

Chain reorg refers to a situation in which a previously accepted chain is replaced by an alternative, longer chain. This of course, being the basis of Nakamoto-based blockchains' way of dealing with consensus, where the longest fork is considered the valid one and thus becomes the preferred fork, as it ensures the highest level of security and validity, except where it is a malicious chain reorg. A malicious chain reorg can open the chain to various other vulnerabilities.

2.2.7 Scalability

Scalability is a critical concern in blockchain technology and refers to a network's ability to handle an increasing load of transactions efficiently [21]. Scalability in blockchain networks, particularly PoW-based chains, has been attributed to the speed of transaction processing, which does not increase proportionally with the addition of additional resources [19]. In Bitcoin, a new block is created and added to the chain approximately every ten minutes [18].

3 Erdos Overview

The Erdos Consensus has taken a novel approach to address some of the challenges facing blockchain technology. Particular variance from the common consensus mechanisms is as follows:

Transaction pool: All transactions are collected in a pool, and each transaction includes a timestamp that is utilised for the selection of the forger and vetters. The selection of the forger and vetters is initiated when the number of transactions in the transaction pool reaches a predetermined algorithmic threshold.
Forger and vetter selection: During the creation of a new block, a forger and two vetters are chosen. The selection process involves using the timestamp from the last transaction in the pool and a seven-digit subset of the hash to select the forger and vetters from a list of eligible nodes. Nodes are deemed eligible if they have not participated as a sender or recipient in any transactions of the current block, and if their local blockchain is up-to-date by comparing the hash of the last block in each node's blockchain with the hash of the last block in the forger, vetter1, and vetter2's blockchains.

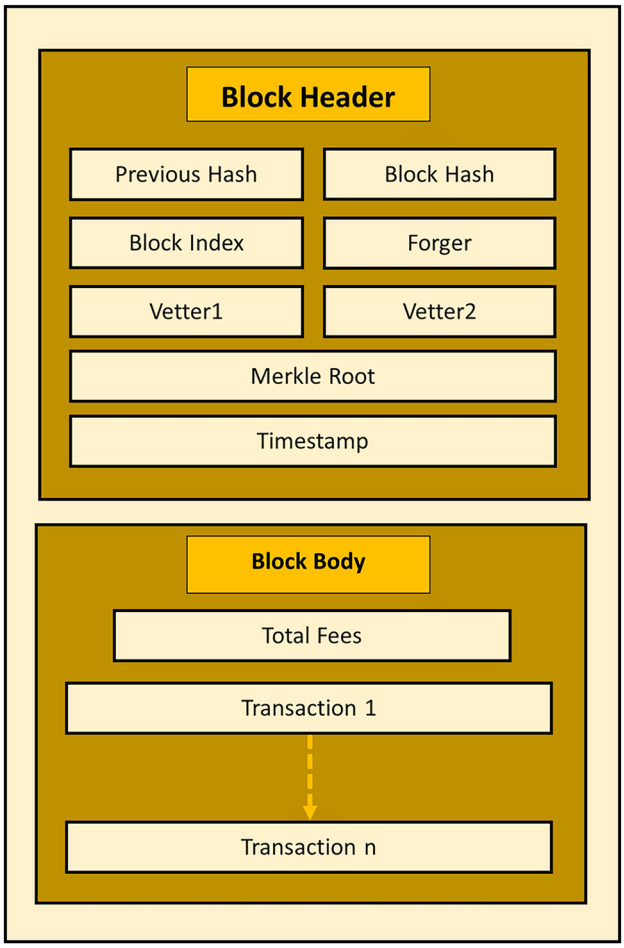

Fig. 3 Erdos block structure

Block creation: The forger constructs a new block containing transactions from the pool and total transaction fees. The block contains the hash of the previous block and addresses of the forger and vetters. The 'Block' class has been structured to include the forger and vetters as attributes, which are included in the calculation of the block's hash. In addition, a Merkle tree of the included transactions is generated. Figure 3 shows the full block structure.

Block verification: The two vetters verify the new block. If both vetters agree that a block is valid, it is added to the blockchain. The verification process is conducted in two phases. First, after a forger has created a prospective block with all the necessary block details, the first vetting commences and is performed by Vetter1. Upon successful vetting by Vetter1, the forger adds reward transactions to the block. Rewards are allocated as predetermined percentages of the total block transaction fees, which are fixed amounts per transaction. These rewards are distributed to the forger, Vetter1, and Vetter2 as well as a calculated surplus sent in a separate transaction to the network for blockchain network management and maintenance. A new Merkle tree, including the new transactions, is created and replaces the original one, and a new hash is calculated. This updated block is then sent to Vetter2 for the second vetting. The second vetting confirms the modified block. Upon successful confirmation, the forger appends the block to the chain and all wallets related to the transactions in the new block are updated.

Blockchain update: All nodes in the network update their blockchain to include the new block, ensuring that all nodes have the same version of the blockchain, which is essential for the security and reliability of the network.

4 Erdos Architecture

4.1 Forger and Vetter Selection

The key to the uniqueness of the Erdos algorithm is the selection of a node that adds a new block to the blockchain. In contrast, the Proof of Work employs the energy-inefficient method of mining which, regardless of both extensive and wasteful resource consumption, still results in the possibility of selfish mining, double spending, and a 51% attack risk due to the non-deterministic finalisation of a block. The Erdos algorithm selects three random nodes using multiple layers of randomness, including the generation of a unique subset, as shown in Fig. 4.

These three nodes consist of a forger and two vetters, who perform the following tasks:

The forger is ultimately responsible for creating a new prospective block, which then undergoes the first vetting conducted by Vetter1. Upon passing the first vetting, the forger then adds reward transactions to the transactions in the prospective block, after which a second vetting, conducted by Vetter2, verifies that the newly added reward transactions are valid. With the second vetting successful, the forger proceeds to add a newly created block to the blockchain.

The fact that only one forger, one vetter1, and one vetter2 can ever be selected in the creation of the next block to be added to the chain, and no other block can be added until the process of this forging is completed, eliminates the risk of having a longer chain resulting from selfish mining.

Algorithm 1 Select Forger and Two Vetters

Require: $self.transaction\_pool$, $self.nodes$, $self.is\_last\_block\_up\_to\_date$
Ensure: A tuple containing the forger and two vetters, or an error message

function SELECTFORGERANDTWOVETTERS
  $last\_tx\_timestamp \leftarrow$ **String**$(self.transaction\_pool[-1].timestamp)$
  $initial\_hash \leftarrow$ **SHA256Hash**$(last\_tx\_timestamp)$
  $start\_idx \leftarrow$ **RandomInteger**$(0,$ **length**$(initial\_hash) - 7)$
  $subset \leftarrow$ **Substring**$(initial\_hash, start\_idx, start\_idx + 7)$
  $subset\_hash \leftarrow$ **SHA256Hash**$(subset)$
  $excl\_addresses \leftarrow$ {**tx.sender for tx in** $self.transaction\_pool$}∪
    {**tx.recipient for tx in** $self.transaction\_pool$}
  $eligible\_nodes \leftarrow$ [**node for node in** $self.nodes$**if node not in**
    $excl\_addresses$ **and** $self.is\_last\_block\_up\_to\_date(node)$]
  if length$(eligible\_nodes) < 3$ **then**
    return {'error': 'Not enough eligible nodes'}
  end if
  $closest\_nodes \leftarrow$ **new MinHeap**()
  for each $node$ **in** $eligible\_nodes$ **do**
    $distance \leftarrow$ **AbsoluteValue**(**Integer**(**SHA256Hash**$(node), 16)-$
      Integer$(subset\_hash, 16))$
    HeapPush$(closest\_nodes, (-distance, node))$
    if length$(closest\_nodes) > 3$ **then**
      HeapPop$(closest\_nodes)$
    end if
  end for
  $forger \leftarrow$ **HeapPop**$(closest\_nodes)$
  $vetter1 \leftarrow$ **HeapPop**$(closest\_nodes)$
  $vetter2 \leftarrow$ **HeapPop**$(closest\_nodes)$
  return $(forger, vetter1, vetter2)$
end function

The function detailed in Algorithm 1 introduces a method for selecting forgers and vetters based on the latest transaction timestamp and a randomised process with the aim of ensuring fairness and unpredictability in the selection process. There is an additional check to ensure that only nodes with the latest blockchain information are eligible for selection. It avoids potential biases by excluding recent transaction participants and relies on cryptographic hashing for secure and random selections, thereby contributing to the overall integrity and decentralisation of the blockchain network.

Step 1 Hashing the Last Transaction Timestamp: The function begins by extracting the timestamp of the last transaction in the transaction pool. This timestamp is converted into a string and hashed using the SHA256 algorithm to produce an initial hash.
Step 2 Generating a Random Subset Index: A random starting index is generated within the range of the initial hash's length minus seven. This index is used to select a seven-digit subset from the initial hash.
Step 3 Hashing the Subset: The selected seven-digit subset of the initial hash is hashed again using SHA256 to create a subset hash.
Step 4 Excluding Addresses: The function compiles a set of excluded addresses, comprising the senders and recipients of all transactions in the current block. These addresses are excluded from the selection process to maintain impartiality.
Step 5 Identifying Eligible Nodes: The function creates a list of eligible nodes, including all nodes in the network, except those with excluded addresses. Additionally, it ensures that only nodes with up-to-date information about the last block are considered, thereby enhancing the integrity of the process.
Step 6 Selecting Forger and Vetters: This function employs a min-heap to identify the nodes closest to the subset hash. The closest node is chosen as the forger, and the next two closest nodes are selected as vetters.

Step 7 Validation of Eligible Nodes: If fewer than three eligible nodes, the function returns an error message, indicating insufficient nodes for selection.
Step 8 Returning the Selection: The function returns a tuple containing the addresses of the selected forger and two vetters.

4.2 First Vetting

Algorithm 2 describes a function that evaluates whether a given block in a blockchain network passes the initial criterion set for vetting by the first designated vetter. It is crucial to maintain the integrity and security of the blockchain, ensuring that only blocks that meet the defined criteria are considered valid and are added to the chain

Step 1 Node Address Verification: Initially, the function checks whether the address of the current node matches the address of the first vetter (vetter1). If not, the function returns false, indicating that this node is not authorised to perform the first vetting.
Step 2 Transaction Verification: The function iterates through each transaction (tx) in the block. Two checks are performed for each transaction. The first is signature verification, which checks whether the transaction's signature is valid by using the verify_signature method of the sender's wallet. This ensures that the transaction was indeed initiated by the sender and has not been tampered with. The second is Transaction Validity, which checks if the transaction is valid based on prescribed criteria; if any transaction fails, the function returns false.

Algorithm 2 First Level Vetting of a Block

Require: $block$ (block to be vetted), $forger$ (address of the forger), $vetter1$ (address of the first vetter), $vetter2$ (address of the second vetter), $self.node\_address$ (address of the current node), $self.node\_wallets$ (wallets associated with nodes), $self.chain$ (current blockchain), $last\_transaction\_pool$ (transactions in the last block)
Ensure: Boolean value indicating the result of the vetting process

```
function FirstVetting(block, forger, vetter1, vetter2)
    if self.node_address ≠ vetter1 then
        return false
    end if
    for each tx in block.transactions do
        if not self.node_wallets[tx.sender].VerifySignature(tx, tx.signature) or not self.IsValidTransaction(tx) then
            return false
        end if
    end for
    if block.forger ≠ forger or block.vetter2 ≠ vetter2 then
        return false
    end if
    last_block ← self.chain[−1]
    if block.previous_hash ≠ last_block.hash or block.index ≠ last_block.index+1 then
        return false
    end if
    if block.transaction_fees ≠ SumOfFees(block.transactions) then
        return false
    end if
    merkle_tree ← self.CreateMerkleTree(last_transaction_pool)
    if merkle_tree ≠ block.merkle_tree then
        return false
    end if
    return true
end function
```

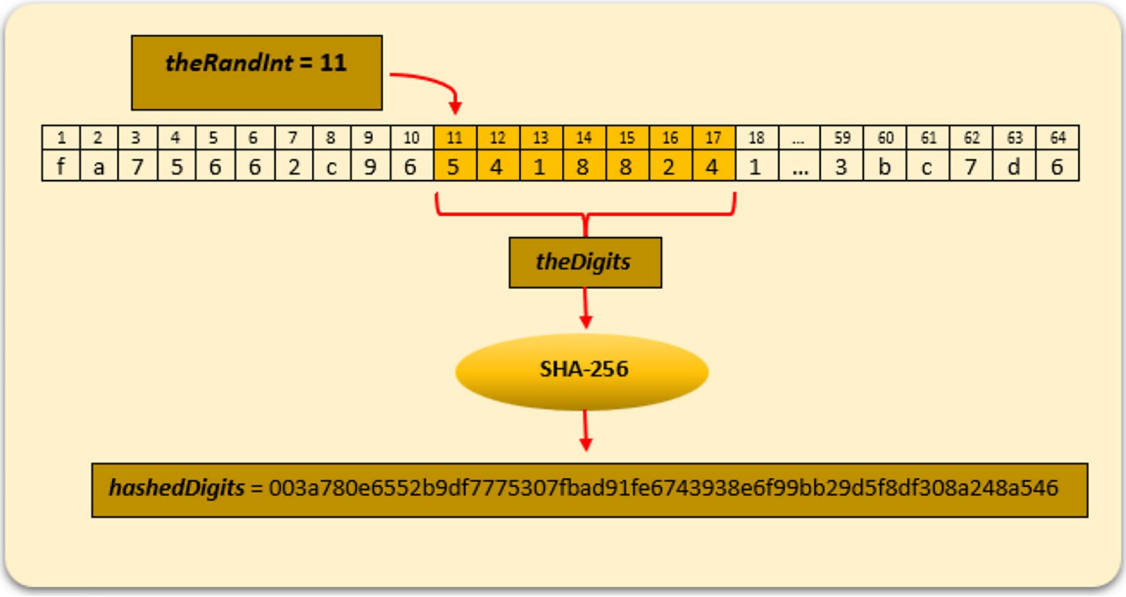

Fig. 4 Subset_Hash generation

Step 3 Forger and Vetter Verification: The function verifies that the forger and the second vetter (vetter2) of the block are the same as those specified in the inputs. If there is a mismatch, it returns false. This step ensures that the block was proposed and vetted by the correct nodes.
Step 4 Block Consistency Check: The function retrieves the last block from the current blockchain (self.chain) and first, it compares the previous_hash of the current block with that of the last block. These must match in order for the block to be considered valid. Second, it checks whether the index of the current block is exactly one more than that of the last block, ensuring the correct sequence in the blockchain. Any discrepancy in these checks results in false-return values.

Step 5 Transaction Fee Calculation: The function calculates the total transaction fees in the block and compares it with the transaction_fees field of the block. If there is a mismatch, it returns false.
Step 6 Merkle Tree Validation: The function constructs a Merkle tree using the transactions from the last transaction pool and compares it with the Merkle tree included in the block. If they do not match, it returns False.
Step 7 Successful Vetting: If all checks pass, the function returns True, indicating that the block has successfully passed the first level of vetting.

4.3 Second Vetting

Algorithm 3 Second Level Vetting of a Block

Require: $new - block$ (block to be vetted), $forger$ (address of the forger), $vetter1$ (address of the first vetter), $vetter2$ (address of the second vetter), $self.node - address$ (address of the current node)
Ensure: Boolean value indicating the result of the vetting process

```
function SecondVetting(new_block, forger, vetter1, vetter2)
    if self.node_address ≠ vetter2 then
        return false
    end if
    new_merkle_tree ← self.CreateMerkleTree(GetTransactionsDict(
        new_block.transactions))
    if new_merkle_tree ≠ new_block.merkle_tree then
        return false
    end if
    return true
end function
```

The Second_Vetting function detailed in Algorithm 3 is designed to perform a secondary-level validation on a new block in a blockchain network. It specifically focuses on verifying the integrity of the block's Merkle tree. It acts as a safeguard, ensuring that only blocks with verified and consistent transaction data are accepted into the blockchain.

Step 1 Vetter Verification: The function begins by confirming whether the address of the current node matches that of the second vetter (vetter2). This step ensures that only the designated second vetter can perform this level of vetting. If the addresses do not match, the function immediately returns false, indicating that the node is not authorised to vet the block.

Step 2 Merkle Tree Reconstruction: The function reconstructs the Merkle tree from the transactions present in the new block. This is achieved by converting each transaction (tx) into a dictionary format (tx.__dict__), and then using these dictionaries to create a new Merkle tree.

Step 3 Merkle Tree Verification: The function then compares the newly constructed Merkle tree with the Merkle tree already present in the new_block. This comparison is crucial because it ensures the integrity and consistency of the transaction data within the block. If the Merkle trees do not match, it signifies a discrepancy in the transaction data, and the function returns false.

Step 4 Successful Vetting: If the reconstructed Merkle tree matches that in the block, the function returns True. This indicates that the block passed the second level of vetting, verifying the integrity of its transactions and their organisation within the Merkle tree.

4.4 Finalisation

Algorithm 4 Finalize Block

Require: new_block, $forger$, $vetter1$, $vetter2$, $self.chain$, $self.transaction\_pool$, $self.node\_wallets$
Ensure: A string indicating the result of the block finalization

```
function FinalizeBlock(new_block, forger, vetter1, vetter2)
    total_fees ← new_block.transaction_fees
    reward ← (total_fees × 0.75)/3
    surplus ← total_fees − (reward × 3)
    Create reward transactions for forger, vetter1, vetter2
    Create surplus transaction for the network
    Add transactions to new_block.transactions
    new_merkle_tree ← CreateMerkleTree(transaction dictionaries)
    new_block.merkle_tree ← new_merkle_tree
    is_valid ← SecondVetting(new_block, forger, vetter1, vetter2)
    if not is_valid then
        return "Second vetting failed."
    end if
    Recalculate new_block.hash and update self.chain
    Clear self.transaction_pool
    Update wallet balances for forger, vetter1, vetter2, "Network"
    return "Block finalized and added to the blockchain."
end function
```

Algorithm 4 describes the finalize_block function that is responsible for finalising a block in a blockchain network. It achieves this by distributing rewards, updating the blockchain with the new block, and managing wallet balances.

Step 1 Reward Calculation: The function starts by calculating the total transaction fees in the new_block. It then computes the reward for the forger and the two vetters, which is 75% of the total fees divided equally between them. The remaining 25% of fees, termed surplus, is allocated to the network. These values were set arbitrarily for the purposes of this study.
Step 2 Creating Reward and Surplus Transactions: Reward transactions are created for forger, vetter1, and vetter2. Additionally, a surplus transaction is created for the network, where the network itself is both the sender and recipient.
Step 3 Adding Transactions to the Block: All new rewards and surplus transactions are added to the new_block.
Step 4 Merkle Tree Update: A new Merkle Tree, including all transactions in the new_block, is created and assigned to the block, ensuring accurate representation of all transactions.
Step 5 Second Vetting: The updated block undergoes a second vetting process. If the block fails, the function returns a message, indicating that the second vetting has failed.
Step 6 Block Hash Calculation: Assuming that the second vetting is successful, the hash of new_block is recalculated to reflect the inclusion of the new transactions.
Step 7 Updating the Blockchain: The newly finalised block is appended to the blockchain (self.chain) and the transaction pool (self.transaction_pool) is cleared.
Step 8 Updating Wallet Balances: The wallets of the forger, vetter1, and vetter2 are updated with their respective rewards. In addition, the wallet of the network is updated with the surplus amount.
Step 9 Final Output: The function returns a message indicating successful finalisation and addition of the block to the blockchain.

This function ensures the integrity of the blockchain by finalising the blocks using a rigorous process of reward distribution, Merkle tree construction, vetting, and blockchain update. This is a critical step in maintaining the consistency of the blockchain and incentivising nodes participating in the block validation and creation process. Figure 5 shows the block creation process from start to finish.

The add_transaction_to_pool function described in Algorithm 5 adds new transactions to the transaction pool in the blockchain network. It validates transactions, manages the transaction pool, and triggers block creation when the pool reaches its capacity.

Step 1 Transaction Signature and Validation: Initially, the function generates a signature for the transaction using the sender's wallet, and assigns this signature to the transaction. It then verifies the validity of the transaction. If

4.5 Transaction Pooling

Algorithm 5 Add Transaction to Pool

Require: $transaction$, $self.transaction\_pool$, $self.node\_wallets$
Ensure: A string indicating the result of adding the transaction

function AddTransactionToPool($transaction$)
  $signature \leftarrow self.node\_wallets[transaction.sender]$.
    SignTransaction($transaction$)
  $transaction.signature \leftarrow signature$
  if not $self$.**IsValidTransaction**($transaction$) **then**
    return "Error: Invalid transaction."
  end if
  $self.transaction\_pool$.**append**($transaction$)
  if length($self.transaction\_pool$) ≥ 5 **then** ▷ Assuming max pool size is 5
    $merkle\_tree \leftarrow self$.**CreateMerkleTree**($self.transaction\_pool$)
    $merkle\_tree\_str \leftarrow$ **json.dumps**($merkle\_tree$)
    $(forger, vetter1, vetter2) \leftarrow self$.**SelectForgerAndTwoVetters**()
    $(new\_block, transaction\_pool\_dict) \leftarrow self$.**CreateNewBlock**($forger$, $vetter1, vetter2$)
    if $self$.**FirstVetting**($new\_block, forger, vetter1, vetter2, merkle\_tree\_str$) **then**
      $finalize\_result \leftarrow self$.**FinalizeBlock**($new\_block$, $forger, vetter1, vetter2$)
      if $finalize\_result$ = "Block finalized and added to the blockchain." **then**
        return "New block has been added to the blockchain."
      else
        return "New block failed second round of vetting."
      end if
    else
      return "New block failed first round of vetting."
    end if
  end if
  return "Transaction added to pool."
end function

the transaction is invalid, the function returns an error message indicating an invalid transaction.
Step 2 Adding to the Transaction Pool: If transaction is valid, it is appended to the pool.
Step 3 Checking the Pool Size and Block Creation: The function checks whether the transaction pool has reached its maximum capacity (assumed to be five transactions in this case). This value was chosen arbitrarily for this study. If the pool is full, the function initiates the block-creation process by first creating a Merkle tree from the transactions in the pool. Then, the Merkle tree is converted into a string format, after which the function selects the forger and two vetters using the select_forger_and_two_vetters method. A new block is created using the selected forger and vetters.
Step 4 Block Vetting and Finalisation: If the vetting by both vetters is successful, the block is finalised using the finalize_block method and the block is added to the blockchain, the function returns a message stating that a new block has been added to the blockchain. In the event that the block fails in the second round of vetting during finalisation, the function returns a message indicating this failure.
Step 5 Return Message: If the transaction pool has not reached its maximum capacity, the function simply returns a message confirming that the transaction has been added to the pool.

4.6 Node Chain Synchronisation

To overcome the longest-fork issue when synchronising node chains, the normal path for obtaining the longest chain from the blockchain node list is used. The identified longest chain must be verified, and this is done by checking it against the last block's forger and two vetters, as these are the most reliable nodes because they had been vetted for the process of producing the previous block. Algorithm 6 illustrates how this is achieved.

Algorithm 6 Check If Last Block is Up-To-Date

Require: $node$, $self.previous\_forger$, $self.previous\_vetter1$, $self.previous\_vetter2$, $self.hash$
Ensure: Boolean indicating if the last block of the node is up-to-date

function IsLastBlockUpToDate($node$)
  $last\_block\_hash \leftarrow self.\mathbf{Hash}(node.blockchain[-1])$
  $forger\_last\_hash \leftarrow self.\mathbf{Hash}(self.previous\_forger.blockchain[-1])$
  $vetter1\_last\_hash \leftarrow self.\mathbf{Hash}(self.previous\_vetter1.blockchain[-1])$
  $vetter2\_last\_hash \leftarrow self.\mathbf{Hash}(self.previous\_vetter2.blockchain[-1])$
  $is\_up\_to\_date \leftarrow (last\_block\_hash = forger\_last\_hash)$**and**
    $(last\_block\_hash = vetter1\_last\_hash)$**and**
    $(last\_block\_hash = vetter2\_last\_hash)$
  return $is\_up\_to\_date$
end function

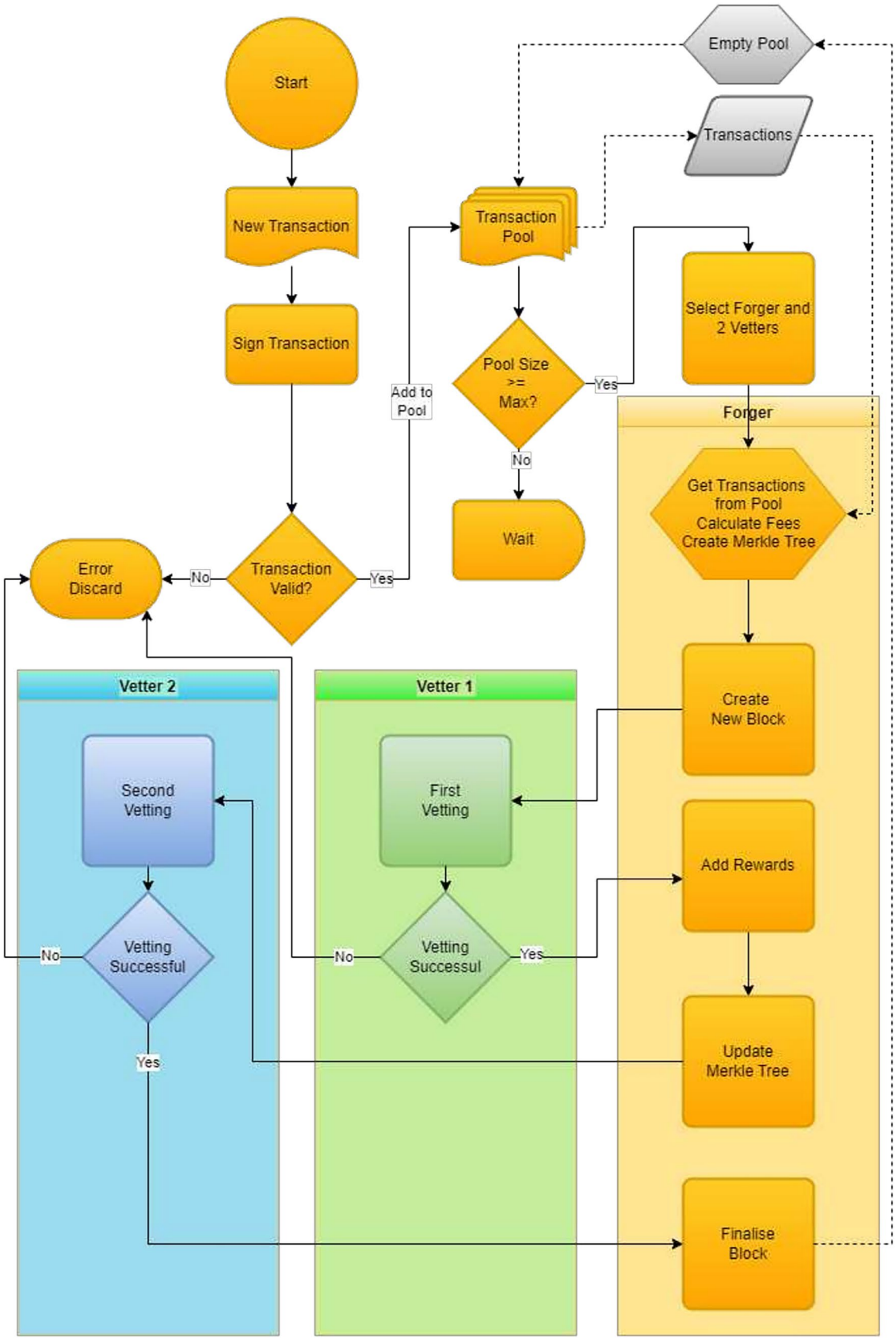

Fig. 5 Erdos block creation overview

This function verifies the current status of a node's blockchain by comparing its latest block with those of key network participants: the previous forger and two vetters. This method is employed to ensure consistency across different nodes in the blockchain network.

Step 1 Hash Extraction: The function begins by obtaining the hash of the last block in the given node's blockchain. This hash serves as the node's current Blockchain status identifier.
Step 2 Comparison with Key Nodes: The function retrieves the hash of the last block from the blockchains of the previous forger, previous_vetter1, and previous_vetter2.
Step 3 Up-to-Date Validation: The function checks whether all three hashes (from the forger, vetter1, and vetter2) match the node's last block hash. If they all match, it indicates that the node's blockchain is up-to-date with the key participants' blockchains.
Step 4 Returning the Status: The function returns TRUE if the node's last block is up-to-date, meaning that it matches the last blocks of the forger and both vetters, and FALSE if there is a mismatch, indicating that the blockchain of the node might not be synchronised with the rest of the network.

This function is crucial in that it ensures that a node's blockchain is aligned with that of the main validators (forger and vetters). This helps to mitigate issues related to blockchain forks and synchronisation discrepancies, which are common challenges in distributed ledger systems.

This function is crucial in that it ensures that a node's blockchain is aligned with those of the main validators (forger and vetters). This helps mitigate issues related to blockchain forks and synchronisation discrepancies, which are common challenges in distributed ledger systems.

5 Discussion

Erdos presents a novel consensus mechanism that addresses the key concerns surrounding security, scalability, decentralisation, and fairness. While PoW and PoS provide high security through computational work and stake size, respectively, they remain vulnerable to 51% attacks. Erdos tackles this problem by relying on randomly chosen vetters, independent of mining power or stake, and deterministic block finalisation. This eliminates the advantage of majority control and mitigates the risk of forking, which is crucial for preventing 51% attacks.

Double spending, another potential security threat, is eliminated in Erdos through the involvement of two vetters alongside the forger for each block. This selection process ensures that no single node has an undue influence, and the vetting process itself acts as a safeguard against malicious transactions. Even if a forger attempts to sneak in fraudulent transactions, the vetters have a final say before finalisation, effectively preventing double spending.

Scalability is often a bottleneck in consensus mechanisms. In the case of Erdos, the forger and vetter selection processes could be potential choke points. The proof-of-concept deployment, which was written in Python and deployed in a local development environment, shows an efficient algorithm with O(n + m) complexity, where n is the number of transactions and m is the number of nodes. Testing a large-scale implementation could further shed light on Erdos's capabilities.

Decentralisation is a core principle of blockchain technology, and Erdos delivers this promise through a random vetter selection process. Every participant has an equal chance of being chosen, eliminating the possibility of a central authority or bias.

Fairness is another key aspect of well-designed consensus mechanisms. Erdos ensures a fair reward distribution by granting all participants an equal opportunity to be selected as a forger or vetter. Only those involved in transactions or with outdated blockchains are excluded from participation in a given block cycle to ensure a transparent and equitable system.

5.1 Future Work

Although the proposed Erdos consensus algorithm presents a promising approach to address the challenges of security, decentralisation, and fairness in blockchain systems, several areas warrant further exploration and research.

5.1.1 Large-scale Implementation and Performance Evaluation

This study introduced the theoretical foundation of the Erdos algorithm; however, it is crucial to implement and evaluate its performance on a large-scale blockchain network. Such implementation would enable a comprehensive assessment of the scalability, robustness, and practical performance of the algorithm under real-world conditions. This involves deploying the Erdos algorithm on a distributed network with a substantial number of nodes and conducting rigorous performance tests to evaluate metrics, such as transaction throughput, latency, and resource utilisation. Additionally, this large-scale implementation would provide insights into the behaviour of the algorithm and potential bottlenecks when operating at scale.

5.1.2 Formal Security Analysis and Potential Attack Vectors

Although the Erdos algorithm introduces novel mechanisms to enhance security and mitigate potential threats, thorough formal security analysis is essential. This analysis should aim to identify potential attack vectors and vulnerabilities, such as Sybil attacks, Eclipse attacks, and Distributed Denial of Service (DDoS) attacks. By formally modelling and analysing the algorithm's security properties, potential weaknesses can be uncovered and insights can be provided towards developing countermeasures or algorithmic improvements to enhance the overall security and resilience of the Erdos consensus mechanism.

5.1.3 Governance and Decision-making Mechanisms

As blockchain networks continue to grow and evolve, effective governance and decision-making mechanisms have become increasingly crucial. Future research could explore how the Erdos algorithm can be extended or adapted to support on-chain governance, voting mechanisms, and decentralised decision-making processes. This would involve developing frameworks and protocols that enable network participants to propose and vote on changes or updates to the blockchain system, while maintaining the principles of decentralisation, transparency, and fairness. Such governance mechanisms could facilitate the long-term sustainability and continuous improvement of blockchain networks based on the Erdos consensus algorithm.

By addressing these future research directions, the Erdos consensus algorithm can be further refined, validated, and adapted to meet the ever-evolving demands of the blockchain technology and its diverse applications.

6 Conclusion

The Erdos consensus algorithm is a significant contribution to the field of blockchain technology as it addresses key issues of security, decentralization, and fairness. By employing a deterministic block finalization process, the algorithm eliminates risks such as double-spending and provides fork resistance, thus ensuring the authenticity of the blockchain. Furthermore, the multi-tiered vetting process and fair reward distribution mechanism help to mitigate centralization.

However, as with any new consensus mechanism, it is important to thoroughly test and gradually implement the Erdos algorithm to validate its potential in future blockchain applications. As it has not been tested on a large-scale network, the performance and robustness of the algorithm in a real-world situation remain undetermined.

Author Contributions B. Sello: conception, design and writing. J. Yong & X. Tao: review, edit, consultation.

Funding The authors did not receive support from any organization for the submitted work.

Data Availibility Not Applicable.

Code Availibility Not Applicable.

Declarations

Conflict of interest The authors have no Conflict of interest to declare that are relevant to the content of this article.

Ethical Approval Not Applicable.

Consent to Participate Not Applicable.

Consent for Publication Not Applicable.

References

1. Akbar NA, Muneer A, Elhakim N, Fati SM (2021) Distributed hybrid double-spending attack prevention mechanism for proof-of-work and proof-of-stake blockchain consensuses. Future Internet 13(11):285
2. Alam S (2023) The current state of blockchain consensus mechanism: issues and future works. Int J Adv Comput Sci Appl 14(8):84–94
3. Apostolaki M, Zohar A, Vanbever L (2017) Hijacking bitcoin: Routing attacks on cryptocurrencies. pp 375–392
4. Bissias G, Levine B (2020-01). Bobtail: improved blockchain security with low-variance mining
5. Blackshear S, Chursin A, Danezis G, Kichidis A, Kokoris-Kogias L, Li X, Zhang L (2023) Sui lutris: a blockchain combining broadcast and consensus. arXiv.org,
6. Bünz B, Kiffer L, Luu L, Zamani M (2020) FlyClient: Super-light clients for cryptocurrencies. pp 928–946
7. Dong S, Abbas K, Li M, Kamruzzaman J (2023) Blockchain technology and application: an overview. PeerJ Comput Sci 9:e1705–e1705
8. Dou H, Yin L, Lu Y, Xu J (2022) A probabilistic proof-of-stake protocol with fast confirmation. J Inf Aecur Appl 68:103268
9. Gencer AE , Basu S, Eyal I, Renesse Rv, Sirer EG (2018) Decentralization in bitcoin and ethereum networks. arXiv.org
10. Guru A, Mohapatra H, Altrjman C, Yadav A (2023) A survey on consensus protocols and attacks on blockchain technology. Appl Sci. https://doi.org/10.3390/app13042604

11. Halpin H (2020) Deconstructing the decentralization trilemma. In: Proceedings of the 17th international joint conference on e-business and telecommunications. SCITEPRESS - Science and Technology Publications. https://doi.org/10.5220/0009892405050512
12. Jiang Y, Liu X, Dai J (2020) A novel pricing mechanism for user coalition in blockchain. Wirel Commun Mobile Comput. https://doi.org/10.1155/2020/8885179
13. Kaur M, Khan MZ, Gupta S, Noorwali A, Chakraborty C, Pani SK (2021) Performance analysis of large scale mainstream blockchain consensus protocols. Ieee Access. https://doi.org/10.1109/access.2021.3085187
14. Kohli V, Chakravarty S, Chamola V, Sangwan KS, Zeadally S (2023) An analysis of energy consumption and carbon footprints of cryptocurrencies and possible solutions. Digital Commun Netw 9(1):79–89
15. Lamriji Y, Kasri M, Makkaoui KE, Beni-Hssane A (2023) A comparative study of consensus algorithms for blockchain. In: 2023 3rd international conference on innovative research in applied science, engineering and technology (IRASET) (pp. 1–8). IEEE
16. Lin C, Ma N, Wang X, Liu Z, Chen J, Ji, S (2018) Rapido: a layer2 payment system for decentralized currencies. arXiv.org
17. Moroz DJ , Aronoff DJ, Narula N, Parkes DC (2020) Double-spend counterattacks: threat of retaliation in proof-of-work systems. arXiv.org
18. Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. Decentralized business review
19. Noh SW, Shin SJ, Rhee KH (2020) PyRos: a state channel-based access control system for a public blockchain network. Secur Commun Netw. https://doi.org/10.1155/2020/8891183
20. Rebello GAF, Camilo GF, Guimarães LCB, de Souza LAC, Thomaz GA, Duarte OCMB (2022) A security and performance analysis of proof-based consensus protocols. Annal Télécommun 77(7):517–537
21. RoÅman N, Corn M, Åkulj G, Diaci J, PodrÅaj P (2022) Scalability solutions in blockchain-supported manufacturing: a survey. StrojniÅiki vestnik. J Mech Eng 68:585–609. https://doi.org/10.5545/sv-jme.2022.355
22. Saad M, Spaulding J, Njilla L, Kamhoua CA, Shetty S, Nyang D, Mohaisen A (2020) Exploring the attack surface of blockchain: a comprehensive survey. Ieee Commun Surv Tutorials. https://doi.org/10.1109/comst.2020.2975999
23. Sahin H, Akkaya K, Ganapati S (2022) Optimal incentive mechanisms for fair and equitable rewards in PoS blockchains. In: 2022 IEEE international performance, computing, and communications conference (IPCCC) pp 367–373. IEEE. ISSN: 2374-9628
24. Schreiber Z (2020) k-root-n: An efficient algorithm for avoiding short term double-spending alongside distributed ledger technologies such as blockchain. Information 11:2. https://doi.org/10.3390/info11020090
25. Shi E (2019) Analysis of deterministic longest-chain protocols. pp 122–12213
26. Wang S, Qu X, Hu Q, Wang X, Cheng X (2023) An uncertainty- and collusion-proof voting consensus mechanism in blockchain. IEEE/ACM Trans Netw 31(5):1–13
27. Wang W, Hoang DT, Hu P, Xiong Z, Niyato D, Wang P, Kim DI (2019) A survey on consensus mechanisms and mining strategy management in blockchain networks. IEEE Access 7:22328–22370
28. Wilhelmi F, Guerra E, Dini P (2022) On the decentralization of blockchain-enabled asynchronous federated learning
29. Yadav AK, Singh K, Amin AH, Almutairi L, Alsenani TR, Ahmadian A (2023) A comparative study on consensus mechanism with security threats and future scopes: blockchain. Comput Commun 201:102–115
30. Yadav AS, Singh N, Kushwaha DS (2023) Evolution of blockchain and consensus mechanisms & its real-world applications. Multimed Tools Appl 82(22):34363–34408
31. Zhou S, Li K, Xiao L, Cai J, Liang W, Castiglione A (2023) A systematic review of consensus mechanisms in blockchain. Mathematics (Basel) 11(10):2248