Article | Received 5 February 2025; Accepted 14 April 2025; Published 27 April 2025
https://doi.org/10.55092/blockchain20250008

Agent-based modeling of Ethereum consensus short-range reorganization attacks

Junhuan Zhang*, Zhengyong Zhao and Ran Ji

School of Economics and Management, Beihang University, China

* Correspondence author; E-mail: junhuan_zhang@buaa.edu.cn.

Highlights:

- Simulation of ex-ante, ex-post and fine-grained reorganization attack agents' decision-making dynamics.
- The impact of blockchain network scale and structure on short-range reorganization attacks.

Abstract: Blockchain technology establishes trust among participants through technical means. However, some malicious nodes may compromise this trust through short-range reorganization attacks for their interest. This paper develops an agent-based model to systematically analyze Proof-of-Stake short-range reorganization attacks, where three types of agents interact through distributed consensus mechanisms with ex-ante, fine-grained, and ex-post reorganization attack strategies. Through rigorous simulation of agent decision-making dynamics, we identify that: (1) Compared with ex-ante reorganization, the ratio of malicious nodes required for ex-post reorganization is much larger. (2) Increasing the node number increases the difficulty of ex-ante and ex-post reorganization. (3) The number of nodes affects ex-post reorganization attacks more significantly than ex-ante attacks. (4) Fine-grained reorganization significantly reduces attack difficulty.

Keywords: blockchain; proof of stake; abnormal behavior; short-range reorganization attacks

1. Introduction

Since the notion of blockchain was introduced in 2008, it has gone through a long development period. Today, it creates the blockchain industry and attracts investments from geeks, scholars, and institutions, who jointly build a prosperous blockchain ecosystem. However, as the blockchain industry develops, its security concerns become more pronounced. According to statistics from blockchain security research institutions, in 2023 alone, 464 security incidents occurred, causing losses of up to 2.486 billion US dollars [1].

As the core of blockchain technology, the consensus algorithm plays a key role in ensuring the blockchain's decentralized, secure, and stable operation. There are currently three mainstream

blockchain consensus algorithms: Proof-of-Work (PoW), represented by Bitcoin; Proof-of-Stake (PoS), represented by Ethereum; and Byzantine Fault Tolerance (BFT), represented by HyperLedger Fabric. Among them, the PoW algorithm is one of the earliest consensus algorithms applied to blockchain. Its essence lies in nodes competing to generate blocks through computing power, thereby earning block rewards while ensuring the network operates normally. However, the increase in computing power depends on high-energy-consuming chips and the electrical energy to drive the chips. According to estimates by mainstream energy-tracking research institutions, in 2021, the energy consumption of the Bitcoin network is equivalent to that of countries such as Sweden and Thailand [2]. Therefore, the PoW algorithm has a serious energy waste problem. BFT-like consensus algorithms are mainly applicable to consortium blockchains. Due to their decentralized nature, this consensus algorithm can only be applied to small-scale consortium blockchains. In summary, the PoS algorithm is very suitable for public large-scale blockchain infrastructure from energy consumption and decentralization perspectives. The Ethereum blockchain represents the broadest user base and the highest market value. Its consensus algorithm successfully switched from PoW to PoS in September 2022, marking that the PoS algorithm, as an alternative technology to the PoW algorithm and representing the future direction of blockchain consensus, has officially entered the mainstream vision. According to CoinMarketCap statistics, as of March 9, 2024, Ethereum's market value has surged to $473.2 billion, constituting 18.2% of the total cryptocurrency market capitalization. Consequently, delving into the anomalies within the POS algorithm assumes paramount significance for ensuring the seamless functionality of the blockchain. In the following elaboration, the Ethereum consensus algorithm refers to the Ethereum PoS algorithm.

Short-range reorganization attacks are more well-known among the many attack methods targeting the Ethereum consensus algorithm. They are relatively easy to implement but yield deleterious consequences such as double spending and front-running attacks. As a result, it is of great significance to model and analyze this attack method. This paper studies the Ethereum PoS algorithm and known attack models, which includes establishing a relatively comprehensive short-range reorganization attacks model based on PoS consensus by integrating the characteristics and attack mode of the consensus algorithm.

The structure of this paper is shown below: Chapter 1 is an introduction to Ethereum and PoS. Chapter 2 is the literature review on the PoS consensus algorithm and attack methods. Chapter 3 introduces the basic settings of the Ethereum consensus. Chapter 4 introduces the known attack methods, including short-range reorganization attacks, and constructs a comprehensive attack model. Chapter 5 analyzes the effectiveness of these attack strategies through simulation experiments, considering factors like malicious node ratio and honest node forking degree. Chapter 6 summarises the findings and highlights the potential for improving Ethereum's PoS algorithm.

The main contributions of this paper are in the following aspects: (1) After a comprehensive analysis of the various factors that contribute to the success of short-range reorganization attacks, this paper successfully establishes a relatively complete attack model based on the multi-agent concept. (2) Based on this model, the impact of malicious node ratio, honest node forking degree, and short-range reorganization block length on attack results are analyzed for the first time.

2. Related literature

2.1. Proof of stake consensus algorithm

The consensus algorithm is the cornerstone of blockchain technology, ensuring that the node ledgers participating in the blockchain network reach a consensus. In open, large-scale networks, blockchain typically utilizes two primary consensus algorithms: the PoW and the PoS algorithms. The PoW algorithm requires nodes to compete for block generation through computing power. The greater the computing power, the higher the probability of creating new blocks. Meanwhile, PoS demands that nodes compete for block generation based on their stake. The larger the stake, the higher the likelihood of producing blocks.

The PoS algorithm can be traced back to 2012. Given the high energy consumption and security issues related to the vulnerability of the Bitcoin network to 51% attacks [3], King *et al.* study a blockchain based on PoS, referred to as PeerCoin [4]. PeerCoin utilizes the PoS to replace Bitcoin's PoW algorithm. Its PoS algorithm integrates the concepts of coin age and PoW, adopting a chained consensus algorithm to enhance the sustainability and security of the chain. Subsequently, many PoS algorithms have been proposed [5]. Considering the energy consumption issues of PoW blockchain and its vulnerability to the 'nothing at stake' attack [6], Kwon proposes the Tendermint consensus algorithm [7], which is the first algorithm to combine PoS with Byzantine fault tolerance (BFT) [8]. To address the energy consumption issue, Gilad *et al.* [9] suggest an Algorand consensus algorithm based on a Verifiable Random Function (VRF) [10]. This algorithm employs a two-layer consensus architecture that balances scalability and decentralization in the blockchain, providing a reliable foundation for distributed applications and achieving significant breakthroughs in security and efficiency. Ethereum's transition from PoW to PoS consensus was initiated by Buterin and Griffith [11] through the Casper Finality Friendly Gadget (Casper FFG), which introduced a BFT-style finality layer over Ethereum's PoW chain. Building on this hybrid framework, Buterin *et al.* analyzed incentive structures, proposing a dynamic staking reward model that inversely correlated validator returns with total network participation to optimize security [12]. To resolve latency in finality and eliminate PoW dependency, Buterin *et al.* [13] further enhance Ethereum PoS by combining the LMD GHOST algorithm [14] and Casper FFG, resulting in the Gasper consensus algorithm. A systematic comparison highlights that while PoW offers the strongest formal security guarantees, PoS systems like Gasper can achieve similar security via checkpoint finality and dynamic availability layers [15].

2.2. Proof of stake consensus attack methods

Ethereum, the world's second-largest public blockchain by market cap, is often compared to a "dark forest" from "The Three-Body Problem," where visibility frequently leads to destruction [16]. Its PoS algorithm, which is relatively new and complex compared to PoW, has been the focus of various studies proposing different attack methods, including short-range reorganization attacks, finality delay attacks, and balance attacks.

Neuder *et al.* [17] propose two methods for launching attacks that require less than one-third of the total stake value. One method involves short-range reorganizations of the underlying chain to initiate double-spending and front-running attacks. The paper constructs models to simulate and calculate the

probability of success in reorganizing blocks of varying lengths. The other method involves attackers using malicious nodes to delay block finality. They also built models to simulate and calculate the probability of successful attacks under different block delays. Schwarz-Schilling *et al.* optimize these two attack methods through adversarial networks, achieving successful attacks with fewer controlled nodes [18]. They combine these two improved attack techniques to develop a third type of attack. This attack allows attackers with less stake, who cannot control network message propagation, to cause blockchain reorganizations through remote attacks. Honest and rational malicious parties can exploit this attack to increase their gains or delay block confirmation times, thereby threatening the security of Ethereum's PoS consensus. Additionally, this attack may lead to voting processing congestion and consensus instability. Kai *et al.* study a new type of attack called saving attacks, which prevents nodes from reaching consensus [19]. In saving attacks, attackers retain their block-producing power during temporary consensus failures and later use it to induce another consensus failure, increasing the delay in block finality. The authors investigate the impact of saving attacks on various fork choice algorithms, including the one used by Ethereum's PoS. They simulate saving attacks on the longest chain rule, GHOST, and LMD GHOST, finding that saving attacks had a significant negative impact on consensus.

When the blockchain network operates under a partially synchronous adversarial environment [20], attackers know when honest nodes execute fork choice algorithms and can relay messages to these nodes before a certain time, while honest nodes cannot immediately update each other's messages. Neu *et al.* propose a balancing attack method in which attackers divide the honest node set into distinct groups while ambiguously proposing blocks, sending two different blocks to various parts of the honest node set, thereby splitting the chain into two forked chains [21]. Attackers influence the network's fork choice algorithm by selectively releasing proof messages that favor one fork, making it appear to have the most validator support. They prevent block finality by maintaining a balanced distribution of validators between the two forks. The probability of success for the attack correlates with the total stake value controlled by the attacker; even controlling just 1% of the total stake can provide an opportunity for the attack.

Buterin proposed a method to enhance proposer weight in response to balancing attacks [22]. When honest nodes promptly receive the block corresponding to the latest time slot, the weight of that block is increased to a specific percentage of the overall weight of the committee for that time slot. Subsequently, Ethereum's PoS fork choice algorithm adopted the LMD GHOST protocol from the Casper CBC consensus algorithm [23]. This algorithm only accepts the most recent voting messages when nodes send multiple votes for a block. Neu *et al.* then improved the balancing attack strategy against Ethereum's consensus, utilizing the characteristic of LMD GHOST that only accepts the latest messages as voting for attacks [24]. This attack requires only a constant number of ambiguous votes to surpass the proposer's elevated weight. The cost of this attack is limited, and a single execution can lead to a permanent split among honest nodes.

In response to the aforementioned variants of balancing attacks, some scholars proposed removing LMD from the LMD GHOST fork choice algorithm and only using the GHOST mechanism, but this could lead to more serious problems. Neu *et al.* propose an avalanche attack method against the GHOST mechanism in PoS [24], which combines selfish mining [25] and balancing attacks. This attack exploits specific weaknesses in the ambiguity of GHOST rules and PoS, allowing attackers to reuse uncle blocks in GHOST, thus contributing ambiguous blocks to multiple ancestor blocks' weight. Ultimately, Ethereum core developers updated the fork choice algorithm to defend against this attack on LMD GHOST,

completely excluding ambiguous validators. The new fork choice algorithm also downgraded the future voting weight of ambiguous validators. This improvement effectively prevented the aforementioned balancing attacks while maintaining defense against avalanche attacks.

In the Ethereum PoS algorithm, the probability of short-range reorganization attacks is relatively high, and related literature has extensively explored this issue. The primary harms caused by such attacks include double-spending and front-running incidents. Concerning double-spending attacks, Rosenfeld details the specific process: the attacker first broadcasts a transaction across the blockchain network to pay digital currency to a merchant [26]. After the merchant receives the digital currency, they release the goods, after which the double-spending attacker initiates a new transaction that conflicts with the initial transaction to return the digital currency to themselves. They utilize a majority of the blockchain network's computing power or stake to publish a forked chain block, thus reversing the old transaction and confirming the new one. Ultimately, the merchant loses both the digital currency and the goods, while the attacker gains both. Eskandari *et al.* describe front-running as an entity profiting by obtaining privileged market information about upcoming transactions in advance [27]. With the development of blockchain technology, front-running has re-emerged in new forms. Daian *et al.* study front-running behavior in blockchain DEXs, finding that attackers use bots to probe the Ethereum network for transactions with captureable value [28]. Once such transactions are identified, they issue a transaction that can replace the attacked transaction by increasing the transaction fee to profit.

3. Ethereum consensus basic settings

The overall architecture of the Ethereum consensus algorithm can be roughly divided into two layers [13]. The first layer involves the node staking process, where regular nodes stake a certain amount of ETH to become validating nodes. Validating nodes have the authority to propose blocks and vote, and they can receive corresponding rewards after participating in consensus correctly. Unlike PoW chains and other chain-based PoS algorithms, nodes must stake before participating in consensus, which helps prevent malicious behavior. When a node is found to engage in malicious activity, the consensus algorithm punishes it by reducing its staked ETH. The second layer is the node voting process. Once a regular node becomes a validating node, it must solve two consensus problems: the fork selection problem and the block confirmation problem. Ethereum consensus employs the GHOST algorithm to address the fork selection problem. This algorithm is based on the heaviest-weight chain rule, selecting the chain with the most node support as the canonical chain. After determining the canonical chain, the proposer links the new blockchain to the head of this chain. Additionally, the consensus mechanism utilizes the Casper FFG algorithm to finalize the block. Once the block is finalized, it cannot be rolled back unless the attacker is willing to risk incurring a significant fine of ETH. The Casper FFG algorithm is inspired by the Byzantine fault tolerance algorithm. Once most nodes cast their votes twice, the block is confirmed. The vast majority of nodes refers to at least 2/3 of the total number of nodes. This ratio ensures that the system can tolerate several node failures without compromising overall reliability and security.

In the consensus algorithm, commonly used terms and their explanations are shown in Table 1:

Table 1. Definition of consensus basic terms.

Terms	Definition
Canonical Chain	The heaviest chain branch.
Regular Node	Participant in the network, responsible for receiving and broadcasting transactions but not directly participating in the consensus process.
Validating Node	A node with sufficient stake, participating in the consensus process, is responsible for verifying transactions and packaging blocks.
Epoch	A period of time on the chain, which is used to divide the rounds of validating nodes. Each epoch contains multiple slots.
Slot	A small period within an epoch, which is used to perform consensus activities of the validating node, including proposing blocks and voting.
Committee	A group of validating nodes responsible for reaching consensus and generating blocks within a specific slot.
Proposer	A special validating node in the committee is responsible for proposing new block proposals within the slot.
Attestation	The attitude of the validating nodes toward the block proposal during the consensus process, reaching consensus and determining the final block through voting.

3.1. Node staking process

There are many ways to stake Ethereum: solo staking, staking as a service, and pooled staking. Solo staking directly locks the held ETH in the network, requiring self-management of node operation and staking processes. Staking as a service involves entrusting a professional staking service provider to manage nodes and staking, saving time and energy. Pooled staking, on the other hand, involves multiple coin holders collaborating in staking, sharing risks and rewards. This article discusses the most influential and safest method, solo staking.

Figure 1 shows the solo staking method process, which is divided into three stages: preparation, application, and activation stage. The first stage is the preparation stage, which involves several key steps. First, according to the requirements of the Ethereum mainnet, the staker should have at least 32 ETH and a dedicated computer connected to the internet ~24/7. Next, the staker must run a full node on their hardware to synchronize Ethereum's historical block data, allowing them to propose and verify blocks during the consensus process. Subsequently, the staker generates a public-private key pair for node validation based on the aggregate signature cryptographic algorithm, where the private key signs messages during consensus, and the public key represents the staker's identity within the Ethereum network. Additionally, the staker generates another public-private key pair for withdrawals, which is necessary for creating a withdrawal certificate. This procedure serves as a cryptographic guarantee to ensure that the staked funds can be withdrawn safely. It's important to configure the private key used

for signing on a dedicated computer so that the Ethereum node program can sign consensus messages whenever needed. Therefore, for security reasons, the private key for signing and the private key for withdrawals should remain separate.

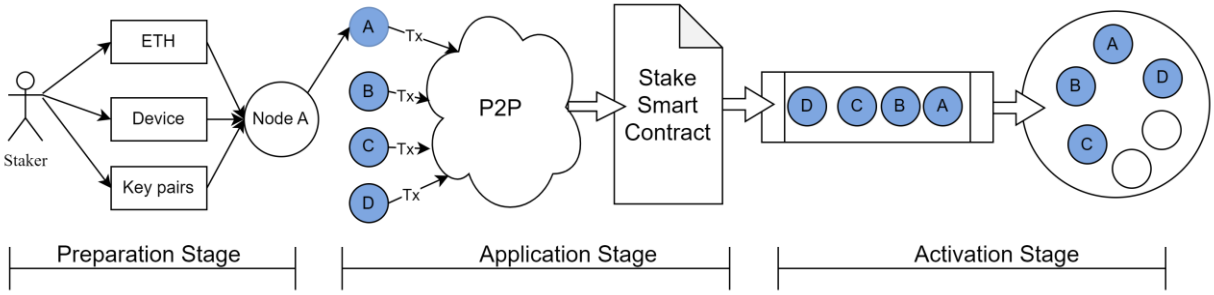

Figure 1. Ethereum PoS staking process.

The second stage is the application stage. Once the staker has completed the preliminary preparations, it can start submitting the staking application to the staking contract. First, the staker packages the prepared validator public key, withdrawal voucher, and the amount of ETH into fixed data. This data is set as an input parameter to the Ethereum transaction. Then, the staker signs the Ethereum staking transaction and sends it to the Ethereum network, which enters the transaction verification pool through broadcasting. When the new block time arrives, the transaction is packaged into the new block by the validating node already in the validating node set. Subsequently, the staking contract receives the staker's ETH, executes the staking transaction, and determines the validity of the data in the staking transaction. If the data is valid and the amount of ETH exceeds the minimum staking amount, a number is assigned to the validating node and add it to the validating node set.

The third stage is the activation stage. Under normal circumstances, a validating node is activated when it is added to the validating node set. Since it takes at least two epochs for Ethereum's blocks to be finally confirmed, for safety reasons, the node must wait at least approximately 12.8 minutes before verifying the block. After completing the staking process, a regular node can become a validating node and participate in the consensus process. Typically, the staker stakes the minimum amount (32 ETH) to maximize benefits. If the staker possesses a large amount of ETH, they may apply for as many validating nodes as possible. Therefore, in the subsequent discussion, scenarios involving stake value or node count are consistent- the greater the number of nodes, the higher the total stake value.

3.2. Epoch and slot

Currently, the number of nodes participating in the Ethereum mainnet is about 500,000. The consensus based on the Byzantine fault-tolerant algorithm requires nodes to participate in voting. However, if all nodes must verify and vote for each block, the efficiency of consensus's network transmission and block verification will be significantly reduced. Therefore, the Ethereum consensus introduces the concepts of epoch and slot [13]. Epoch is a period on the chain used to divide the rounds of validating nodes. Each epoch is divided into several slots. The slot is used to execute the consensus activities of the validating node. Each slot is evenly allocated a part of the validating nodes. These validating nodes form the

committee of the slot and are responsible for proposing blocks and voting. The consensus that introduces the design of epoch and slot can be called the slot-based PoS algorithm.

Figure 2 shows the differences between the slot-based PoS chain and the traditional PoW chain. In Figure 2, all nodes in the Bitcoin chain participate in the block generation process, and 12 blocks have been generated. The Ethereum chain consists of 16 validating nodes. Each epoch contains 4 slots, and each slot is assigned 4 validating nodes. The consensus has run for 3 epochs, or 12 slots, and has also generated 12 blocks. The number of committees C_i in the i-th slot is shown in Equation 3.1:

$$C_i = \frac{N}{S} + \begin{cases} 1 & if\ i \leq N\ mod\ S \\ 0 & if\ i > N\ mod\ S \end{cases} \tag{3.1}$$

Where N is the total number of validating nodes; S is the number of slots in a single epoch.

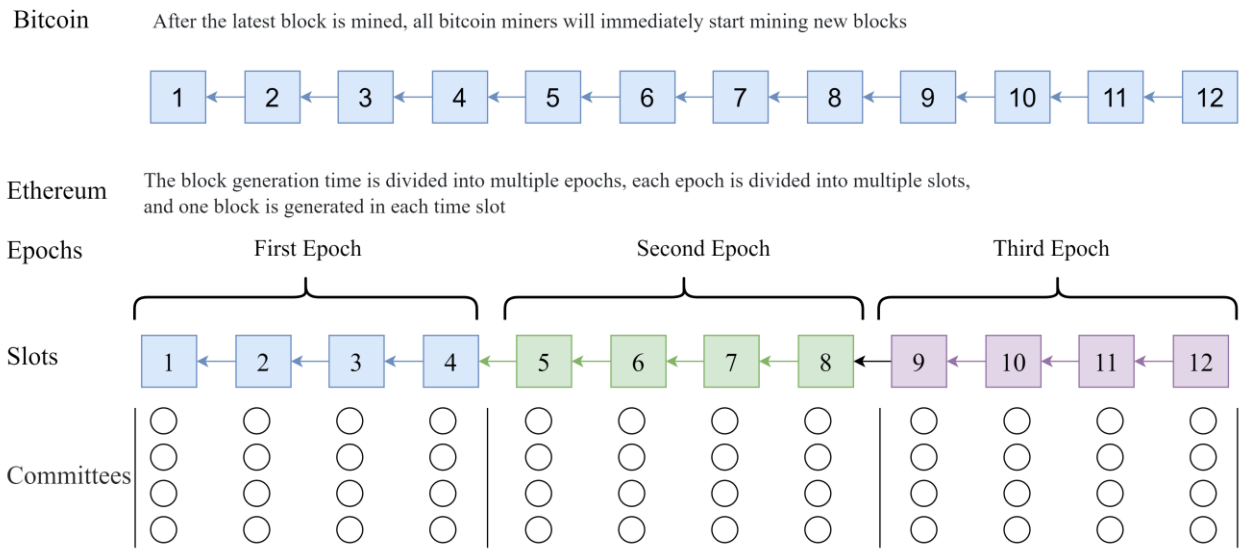

Figure 2. Difference between PoW chain and slot-based PoS chain.

According to Buterin *et al.* [13], the committee and proposers are randomly selected, and the generation process is shown in Equation 3.2:

$$commitee = compute\_committee(seed, validators, slot) \tag{3.2}$$

Where $seed$ is the random seed, which is determined 2 epochs in advance to prevent grinding attacks; $validators$ is the set of validating nodes.

The proposer of the slot is the key role of the slot committee and is responsible for the packaging of the block, which generation process is shown in Equation 3.3:

$$proposer = compute\_propose(seed, commitee) \tag{3.3}$$

3.3. Fork selection algorithm

The fork selection algorithm of Ethereum is based on the GHOST algorithm. In the following discussion, GHOST specifically refers to Ethereum's fork selection algorithm. The GHOST fork selection algorithm is designed to select the block with the greatest weight in the blockchain. Its fundamental concept is to choose the block with the highest observation weight in a greedy fashion, relying on the latest message from the validator, thus creating a secure and highly consistent blockchain [14].

Figure 3 illustrates the GHOST fork selection algorithm. In Figure 3, the root of the tree represents the genesis block. The weight of each node is equal to the sum of the weights of all its descendant nodes. The fork selection algorithm starts from the genesis block, and by searching the tree, the child node with the largest weight value is selected as the parent node for the next search round in each step. This process is repeated until the leaf node is reached. The weight calculation of the node in the chain is shown in Equation 3.4:

$$w_b = \sum_{c \in C(b)} w_c + \sum_{v \in V(b)} S(v) \tag{3.4}$$

Where w_b represents the weight of block b, $C(b)$ represents the set of all subblocks of block b; w_c represents the weight of block c; $V(b)$ represents all the validating nodes of block b; $S(v)$ represents the weight of block v.

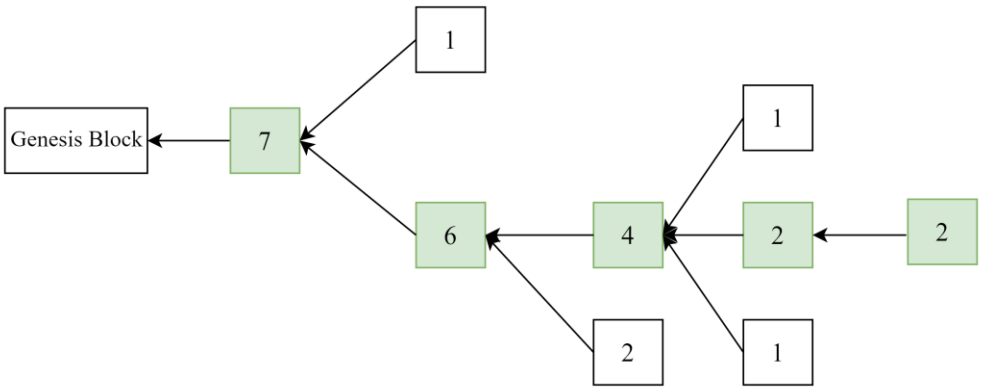

Figure 3. GHOST fork selection algorithm.

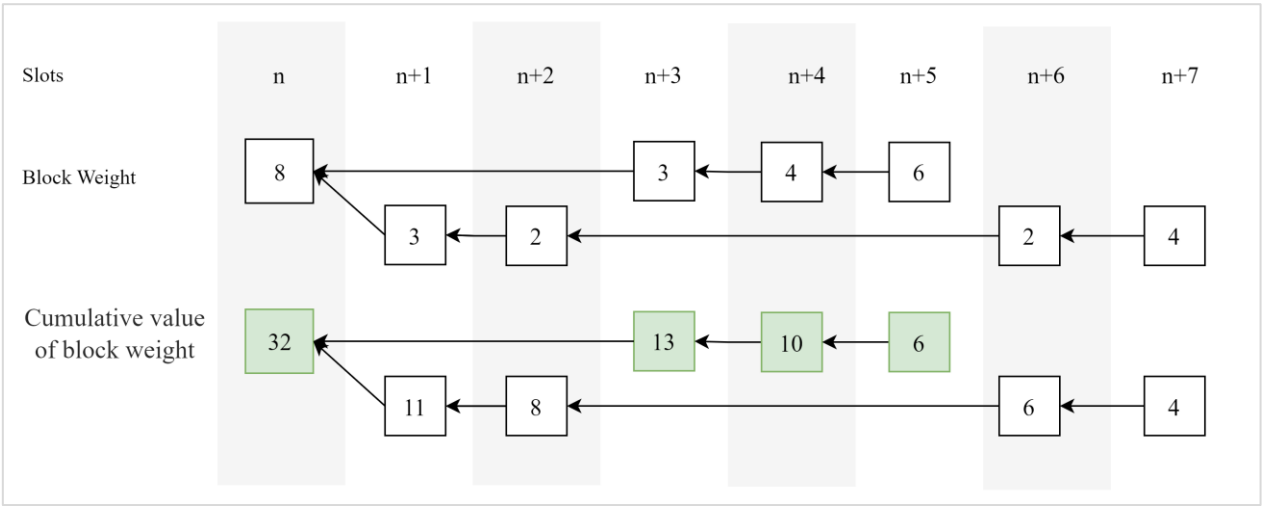

Figure 4. Ethereum forking chain.

Figure 4 shows the forking chain view of Ethereum forks combined with the slot design. It is divided into three parts from top to bottom. First, the top shows the slot number, and the Ethereum consensus is completed within 8 slots. Second, the middle part shows the block voting situation. The votes for the n-th slot reached 8 votes. Starting from the n-th slot, two forked chains $Chain(n+5)$ and $Chain(n+7)$ appeared. These two forked chains represent the chains with the blocks of the $(n+5)$-th slot and the $(n+7)$-th slot as the chain ends. The chain is composed of blocks, as shown in Equation 3.5:

$$Chain(n+5) = B_n, B_{n+3}, B_{n+4}, B_{n+5} \tag{3.5}$$

Where B_{n+5} represents the block generated by slot $n+5$, and the forked chain $Chain(n+5)$ is composed of 4 blockchains.

Finally, the bottom shows the cumulative number of votes, and the number of votes for the root block reaches 32, representing the total number of votes for the entire epoch. Based on the GHOST algorithm, starting from the block in slot n, the next block selects the block B_{n+3} generated by the $(n+3)$-th slot, and the final block of the fork is the block generated by the $(n+5)$-th slot. Therefore, from the view of this node, the green part represents the canonical chain. The weight calculation method of this chain is shown in Equation 3.6:

$$W_{n+5} = WS_n + \sum_{i=3}^{5} WS_{n+i} \tag{3.6}$$

Where W_{n+5} represents the weight of $Chain(n+5)$, which traces back from the block of the $(n+5)$-th slot to the justified block B_n calculated by the Ethereum consensus, and accumulates the weights of these blocks.

Therefore, if the node is the proposer of the $(n+8)$-th slot, the next block points to the block generated by the $(n+5)$-th slot, not the $(n+7)$-th slot, because $W_{n+5} = 21$, which is greater than $W_{n+7} = 19$.

To defend against short-range reorganization attacks, the Ethereum main net has implemented the algorithm of proposer weight enhancement and cross-slot voting weight reduction. The enhanced proposer weight $\widetilde{W_n}$ is calculated in Equation 3.7:

$$\widetilde{W_n} = \begin{cases} W_n + W_{ce} * \Delta w, & if\ 0 \le t_B < \frac{T}{3} \\ W_n, & if\ \frac{T}{3} \le t_B \le T \end{cases} \tag{3.7}$$

Where T is the duration of the time slot. t_B represents the time at which the block is received, starting from this time slot. W_{ce} is the weight of the committee for this time slot, which is the sum of the weights of all validating nodes in the committee. Δw is the ratio of weight enhancement. W_n is the weight of $Chain(n)$.

The reduced cross-slot voting weight WA is calculated in Equation 3.8:

$$WA = \begin{cases} 1, & if\ m = n \\ \Delta wa, & if\ m > n \end{cases} \tag{3.8}$$

Where m refers to the m-th slot when committee N_m votes, and n refers to the n-th slot when the voted B_n is generated. WA means that if a committee votes for a block prematurely, its voting weight should be reduced by Δwa.

3.4. Finality algorithm

Casper FFG is a block finality algorithm introduced by Ethereum consensus. The algorithm is based on the two-phase commit model of Byzantine fault tolerance [11]. Its first phase is justification. Validators stake a certain amount of ether and then participate in block verification. If the vast majority of validators agree that a block is valid, the block is called justified, marking the initial stage of consensus. In the finalization phase, validators continue to verify and add additional votes. If the vast majority of validators agree on a block and all blocks before the block have been justified, the block is confirmed as

final and finality is achieved. Through this design, Casper FFG ensures the reliability and security of consensus in the network and provides a more efficient and scalable consensus mechanism for Ethereum.

Figure 5 shows the finality algorithm, in which the number of Validating Nodes is 60, and a single epoch is divided into 32 slots. After 3 epochs, the consensus state is divided into 3 states: Finalized, Justified, and None. The block numbers from 1 to 32 of the first epoch have been confirmed twice, so they are in the finalized state, and these 32 blocks cannot be rolled back at will. The block numbers from 33 to 64 of the second epoch have been voted by the majority of Validating Nodes once, so they are in the justified state, which can be rolled back. The third epoch is currently voting, and it cannot be determined whether it is one of the two states. However, when the number of votes reaches or exists 40, the third epoch is upgraded to the justified state, and the second is upgraded to the finalized state. The calculation method of the block state is shown in Equation 3.9:

$$S(b) = \begin{cases} Justified, \ if\ S(a) = Finalized\ or\ Justified\ and\ sm(a \to b) \\ Finalized, \ \ if\ S(b) = Justified\ and\ sm(b \to c)\ and\ h(b) = h(c) + 1 \end{cases} \tag{3.9}$$

Where $S(b)$ represents the state function of block b, $sm(a \to b)$ represents the number of votes for block a to b has reached the majority, that is, the number of votes is greater than or equal to 2/3 of the total votes, and $h(b)$ represents the height of block b. If block $(a \to b)$ is voted by most validators and block a has been finalized or reasonable, then block b is confirmed as reasonable, and the genesis block defaults to the finalized state. If block $(b \to c)$ is voted by most validators, block b has been confirmed as reasonable, and the height of block b is equal to the height of block c plus 1, then block b is confirmed as finalized.

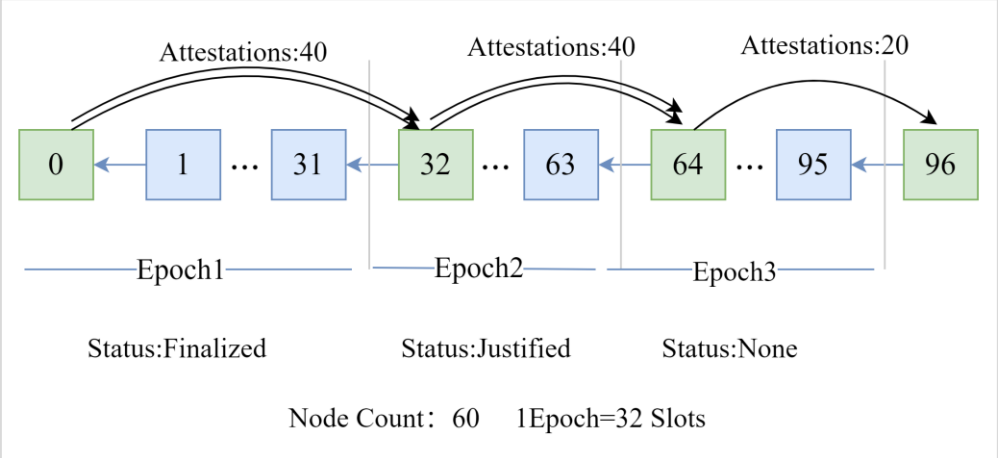

Figure 5. Casper FFG finality algorithm.

4. Short-range reorganization attack model

The Ethereum consensus protocol implements finality assurance through the Casper FFG framework, which imposes a security threshold requiring attackers to control over 1/3 of the staked ETH to execute long-range reorganization attacks against finalized blocks. Short-range reorganization attacks against non-finalized blocks are easier to implement without the 1/3 stake threshold. For non-finalized blocks, the consensus process follows the GHOST fork choice algorithm, which emphasizes recent attestations. This section formalizes a comprehensive short-range reorganization attack model based on multi-agents and systematically elaborates how attackers initiate short-range reorganizations by hiding blocks and

publishing them at the right time, taking advantage of network delays or the proposer's advantage (such as controlling the number of validators). This section first defines the roles of nodes, then explains how to precisely control the message-sending time and its promoting effect on the attack, and finally elaborates in detail how to conduct ex-ante reorganization, refined reorganization, and ex-post reorganization.

4.1. Node agent definition

Five main agents are involved in launching short-range reorganization attacks: proposer node, validator, attacker, honest node and malicious node. Different entities may assume multiple roles simultaneously, and malicious nodes may have various behaviors, some of which are punished. First, let us introduce the different roles:

(1) Proposer node P: In the slot, a unique node is pre-selected from the committee through a random algorithm as the proposer node P, responsible for proposing blocks.

(2) Validator V: In the slot, nodes other than the proposer node are pre-selected from the committee through a random algorithm as validators, responsible for voting on the blocks proposed by the proposer node P.

(3) Attacker A: The entity that initiates the attack can control any abnormal behavior of one or more nodes to achieve the purpose of short-range reorganization attacks.

(4) Honest node H: The node that proposes or votes for blocks according to the GHOST fork selection algorithm during the consensus process.

(5) Malicious node M: During the consensus process, the node that proposes or votes according to the attacker's instructions.

When malicious nodes act as proposers, they can choose to issue conflicting blocks, but such behavior will be punished. On the other hand, as proposers, they can also choose not to generate blocks, or keep blocks and release them when needed, and this behavior will not be punished. When malicious nodes act as validators and vote, if they violate the GHOST rule and vote for conflicting blocks, they will be punished; however, if they vote for a chain with a lower weight, this behavior will not be punished. In addition, when malicious nodes vote, the voting message can be broadcast to specific target nodes when necessary to influence consensus behavior, and such behavior will not be punished. The relationship between the validating node and the committee is shown in Equation 4.1:

$$N = C_1, C_2, C_3, \cdots \tag{4.1}$$

Where the set of validating nodes is defined as N; C_1, C_2, C_3 are different members of committee. The relationship between the committee and each node is shown in Equation 4.2:

$$C = P \cup \{V\} \cap H \cup M \tag{4.2}$$

Where C is the committee, P is the Proposer node, V is the validator, H is the honest node, M is the malicious node.

When the malicious node is a key node in short-range reorganization attacks and in slot i, the definition of malicious node is as shown in Equation 4.3:

$$M(a, R_i, S_i, D_i) \tag{4.3}$$

Where a represents the malicious node address, which is the unique identifier in the network; R_i is the role of the malicious node in slot i, including proposer and validator; S_i represents the attack behavior of the malicious node in the consensus process of slot i. If it is a proposer, it means withholding blocks or issuing two conflicting blocks. If it is a validator, it indicates the way of voting; D_i represents a specific target node to which the malicious node broadcasts.

4.2. Node attack model

When an attacker is able to control some nodes, it can use the vulnerability of the consensus algorithm to launch short-range reorganization attacks. The attack behavior is mainly to coordinate malicious nodes to make appropriate operations at the right time. There are three main attack modes: ex-ante reorganization, fine-grained reorganization, and ex-post reorganization. Before explaining the node attack mode, Table 2 defines the relevant parameters in the consensus process.

Table 2. Definition of relevant parameters in the consensus process.

Parameters	Definition
H_A	Honest node A
M_B	Malicious node B
F	Forking degree
N	Consensus node set
E_n	The n-th epoch
S_n	The n-th slot
P_n	Proposer of n-th slot
B_n	The block proposed by the proposer of n-th slot
$A_{A,n}$	Node A attest the block in n-th slot
$Chain(n)$	The forked chain with the block in n-th slot as the end of the chain, which contains this block and all the blocks traced back to the root
$Chain_A$	Forked chain A
WS_n	Total number of attestations with B_n as the root of the block tree
W_n	Total number of attestations for the blocks contained in $Chain(n)$
WB_n	The number of all attestations for B_n
C_n	The committee of the n-th slot, which contains all nodes assigned to this slot
c_n	The number of nodes in the committee of the n-th slot
t_n	At n-th slot, the moment of broadcasting a block or authentication information
Δw	Percentage of proposer weight increase
Δwa	Percentage of validator voting weight reduction

Ex-ante reorganization attack means that the attacker builds m consecutive blocks in the network to form a forked chain $Chain_A$ in advance, and keeps it from being broadcast to the network. Only the attacker knows the existence of this reserved private chain. Then, the attacker waits for a while and

suddenly broadcasts the private chain to the network. Since the previous m blocks are unknown to the honest node, during this period, the honest node forms a public chain $Chain_B$ that is different from the malicious node, and its length is n blocks. After the malicious node broadcasts the private chain, other nodes make fork selections according to the consensus rules. Since the weight of $Chain_A$ built by the malicious node is greater than the weight of $Chain_B$ built by the honest node, $Chain_B$ is discarded, and the attacker achieves the purpose of ex-ante short-range reorganization. By building blocks on the private chain and voting, the attacker can quickly push the private chain to surpass the public chain, thereby achieving control over the blockchain. This kind of pre-prepared short-range reorganization attacks may lead to security issues such as double spending, which undermines the trust mechanism of the blockchain.

An ex-post reorganization attack means that the attacker attempts to reorganize the public chain generated before the time of the attack in order to achieve a rollback of the public chain. The attacker usually needs to control the majority of nodes in the entire network, and it selects the number of blocks that need to be rolled back. After that, the attacker quickly generates a forked chain $Chain_A$ and votes for it. This can make the weight of the forked chain $Chain_A$ greater than the public forked chain $Chain_B$. Once the weight of $Chain_A$ is greater than $Chain_B$, according to the consensus rules, regardless of whether subsequent nodes are honest, new blocks are appended to $Chain_A$. Therefore, the $Chain_B$ fork chain is discarded, and the ex-post reorganization attack succeed. Ex-post reorganization attacks may also lead to a decrease in consensus security. Comparing ex-ante reorganization and ex-post reorganization, there are the following differences:

(1) The time attributes of the chain are different: the ex-ante reorganization reorganizes chains in the future, while the ex-post reorganization reorganizes chains in the past.

(2) The difficulty of the attack is different: the stake required for ex-ante reorganization is smaller than that of ex-post reorganization, and ex-post reorganization is more complicated.

(3) The fundamental attack methods are different: despite the less difficulty of the ex-ante reorganization attack, its attack methods are more varied, and its means of attack are more complex. The complexity of ex-post reorganization is relatively smaller, but it mainly depends on the stake value it controls.

Whether ex-ante or ex-post reorganization attack, the attacker must control many malicious nodes to launch the attack successfully. A balance attack can increase the success rate and reduce the number of malicious nodes required. This method is implemented by precisely controlling the time the node sends the voting message. Since the message needs to be broadcast on the network before covering all honest nodes, the time the voting message arrives at each honest node may be inconsistent. This inconsistency causes a specific time when the weight of each forked chain observed by half of the honest nodes differs from that of the other half. By taking advantage of this inconsistency, different parts of the honest nodes vote for different forked chains, which is called forking. The forking degree of honest nodes is defined in Equation 4.4:

$$F = H_m / H_h \tag{4.4}$$

Where H_m represents the number of malicious chains supported by the honest node after forking; H_h represents the number of honest chains supported by the honest node. When F equals 0, it means that the attacker has not forked any honest node, and all honest nodes support the chain generated by the

honest node. This situation is equivalent to the attacker not considering the precise control mode, and it is easiest to create a condition with no forking degree. When F equals 1, it means that the attacker divides the honest node set equally, half of which supports the chain generated by the malicious node, and the other half supports the chain generated by the honest node. The attacker precisely controls the time to fork honest nodes as much as possible, which means maximizing F, so that it can reorganize the chain with less controlled nodes, reducing the difficulty of the attack. Compared with the ex-ante reorganization attack, this method requires the forked blocks to be exposed in advance.

4.3. Ex-ante reorganization

This section introduces the specific process of ex-ante reorganization in detail. Assuming that the total number of nodes is 32, the number of malicious nodes controlled by the attacker is 8, and one epoch is divided into 8 slots. First, the attacker's process of reorganizing one block and two blocks is explained, and then the general process of reorganizing k blocks is derived. Through these processes, we can deeply understand how the attacker uses malicious nodes to implement ex-ante reorganization attacks.

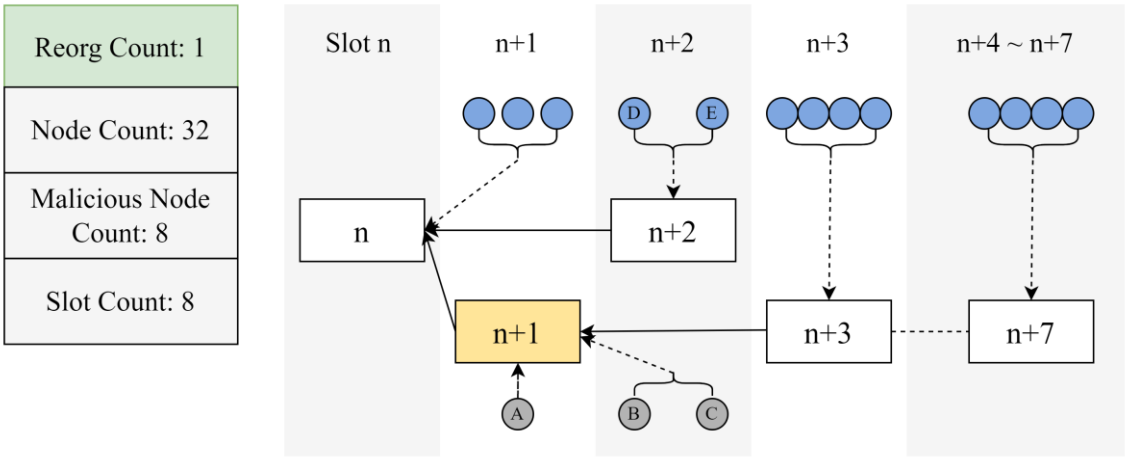

Figure 6. Specific process of one-block ex-ante reorganization.

Figure 6 shows the schematic diagram of one-block ex-ante reorganization. In Figure 6, the square represents the block generated by the slot, the gray circle is the malicious node, and the blue circle is the honest node. The specific steps of the attack include: (1) The first step, the malicious node, as the proposer of the $n+1$ slot S_{n+1}, does not broadcast the generated block immediately. (2) The second step, since the proposer P_{n+2} of slot $n+2$ has not received the block of the $n+1$ slot S_{n+1}, it generates block B_{n+2} and appends it to the block B_n of the $n+2$ slot. (3) The third step, after the block B_{n+2} is broadcasted, the malicious node releases the $n+1$ block B_{n+1}, with the voting message $\{A_{A,n+1}, A_{B,n+1}, A_{C,n+1}\}$ of the malicious node in B_{n+2}. In addition, the honest nodes H_D and H_E vote for B_{n+2}. Finally, the proposer P_{n+3} of slot $n+3$ calculates that WS_{n+1} is 3 and the weight of WS_{n+2} is 2. Therefore, according to the GHOST algorithm, $WS_{n+1} > WS_{n+2}$. It appends the new block B_{n+3} to B_{n+1}, so B_{n+2} is discarded and the consensus is reorganized.

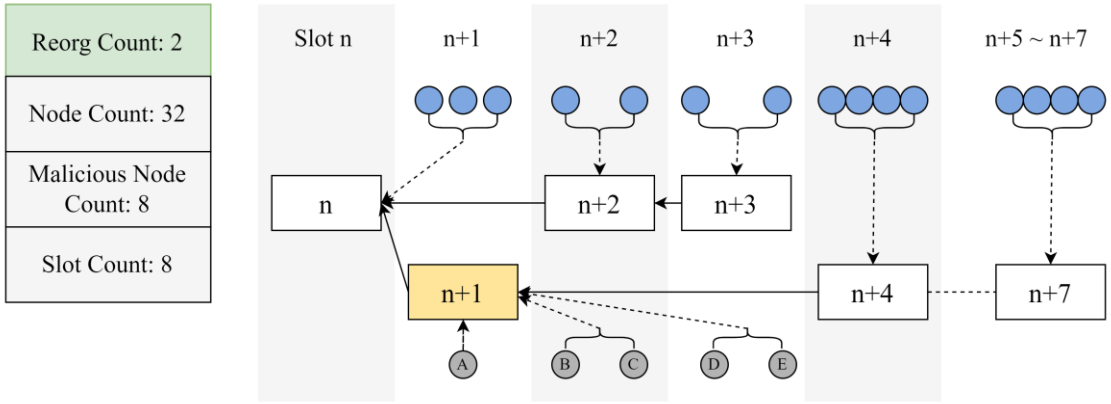

Figure 7. Specific process of two-block ex-ante reorganization.

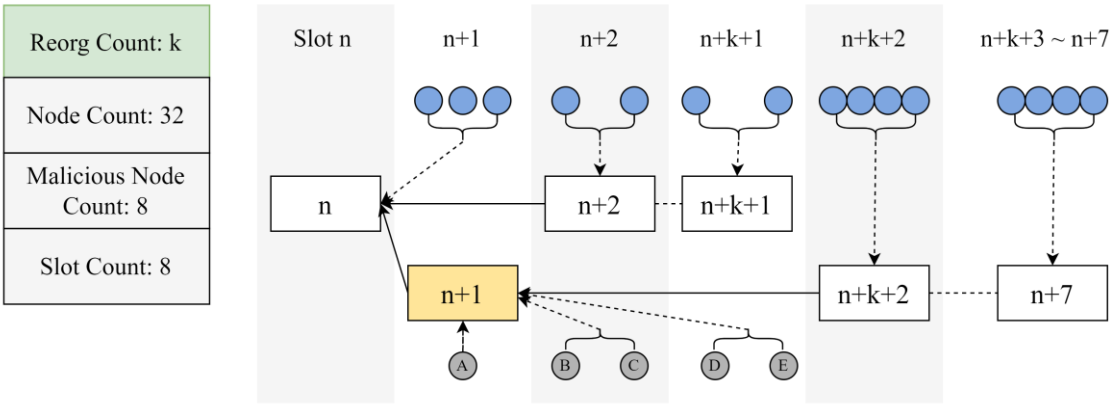

Figure 8. Specific process ex-ante reorganization of k blocks.

Figure 7 shows the process of two-block ex-ante reorganization. It is found that the number of malicious nodes required to reorganize two blocks successfully is 2 more than the case of reorganizing only one block. In this case, the number of malicious nodes required to launch the attack successfully is 5, while the number required to reorganize only one block under the same conditions is 3. The steps of the attack include: (1) The first step, the malicious node becomes P_{n+1} of the $(n+1)$th slot, and does not broadcast immediately after generating the block. (2) The second step, since P_{n+2} does not receive B_{n+2}, it appends block B_{n+2} to B_n. Similarly, B_{n+3} is appended to B_{n+2}. (3) In the third step, after B_{n+3} is broadcast, the malicious node releases B_{n+1} with the malicious node's voting message $\{A_{A,n+1}, A_{B,n+1}, A_{C,n+1}, A_{D,n+1}, A_{E,n+1}\}$. Finally, P_{n+3} calculates that WS_{n+1} is 5 at this time, and the weight of WS_{n+2} is 4. Therefore, according to the GHOST rule, the new block B_{n+4} is appended to B_{n+1}, so B_{n+2} and B_{n+3} are reorganized.

Figure 8 shows the process of k-blocks ex-ante reorganization. The specific steps of the attack include: (1) The first step, the malicious node acts as P_{n+1} of the $n+1$ slot, and does not broadcast the block immediately after generating it. (2) The second step, since P_{n+2} did not receive B_{n+1}, it appends the new block B_{n+2} to B_n. The blocks from B_{n+3} to B_{n+k+1} are appended in sequence as usual. (3) In the third step, after B_{n+k+1} is broadcasted, the malicious node releases the reserved block B_{n+1}, and the malicious nodes from $n+2$ to $n+k+1$ also send out voting messages $A_{A,n+1}, A_{B,n+1}, A_{C,n+1}, A_{D,n+1}, A_{E,n+1}, \cdots$ at this time. Finally, P_{n+k+2} calculates that WS_{n+1} is $2k+1$ at this time, which is greater than the weight

of WS_{n+2} $2k$. Therefore, according to the GHOST algorithm, it appends the new block B_{n+k+2} to B_{n+1}, so the k blocks from $n+2$ to $n+k+1$ are reorganized.

4.4. Fine-grained reorganization

Fine-grained reorganization refers to an attacker who precisely controls the voting behavior of malicious nodes to influence the honest node's judgment of the current forked chain, thereby reducing the difficulty of the attack [18]. The key to this strategy is to control the sending time of the voting message. Using this strategy, the attacker can release the malicious node's voting message at a critical moment, thereby affecting the decision of the honest node during the consensus process, causing the network to reorganize or the node to choose a different forked chain, achieving the purpose of the attack. Through fine-grained reorganization, the attacker can flexibly manipulate the network's consensus process and increase the attack's success rate.

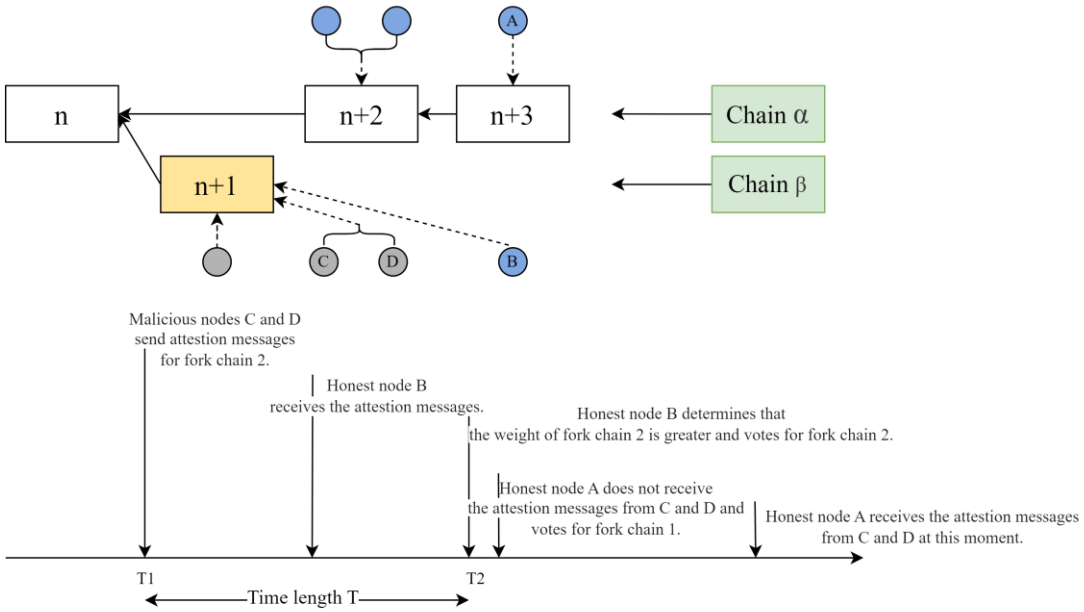

Figure 9. Precisely control the voting message.

Figure 9 shows the precise control of voting message time. The upper part of Figure 9 shows the state of chain bifurcation at $T1$, and the chain is divided into bifurcation chain α and bifurcation chain β. When the slot progresses to $n+2$, bifurcation chain α is currently composed of 2 blocks, namely B_n, B_{n+2}, with a weight of 2, and bifurcation chain β is currently composed of 2 blocks, namely B_n, B_{n+1}, with a weight of 1. When the slot progresses to $n+3$, there are 4 nodes participating in the consensus of the $n+3$ slot, A and B are honest nodes, and C and D are malicious nodes. In the lower part of Figure 11, it is assumed that the attacker can control the voting time of C and D, so that A and B split and vote in different bifurcation chains respectively. First, determine that the malicious node's voting message is sent at the voting time of the $n+3$ slot, $T2-T$, which is $T1$. C and D vote for B_{n+1}. After the broadcast of time T, the honest node B receives the vote of the malicious node. B determines that the weight of the forked chain β is 3, which is greater than the forked chain α, so B votes for the forked

chain β. At this time, A has not received the voting message from the malicious node and still believes that the weight of the forked chain β is less than the forked chain α, so it votes for the forked chain α.

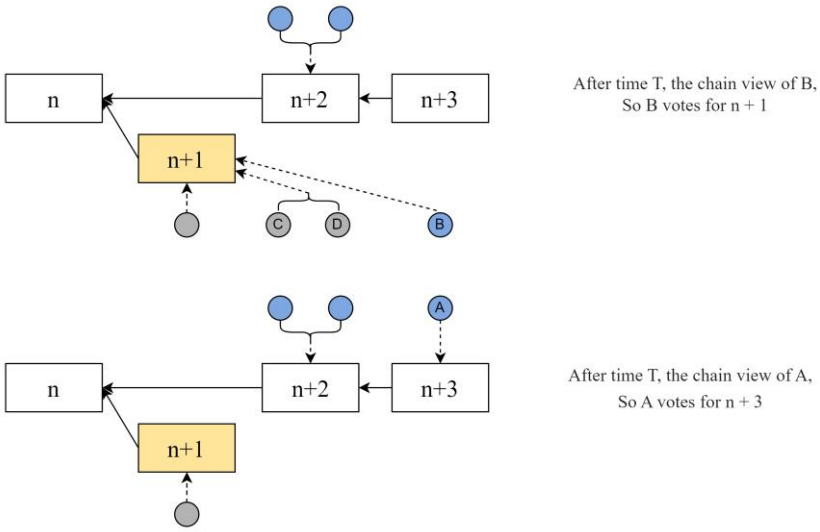

Figure 10. Chain views of precise control of the honest node.

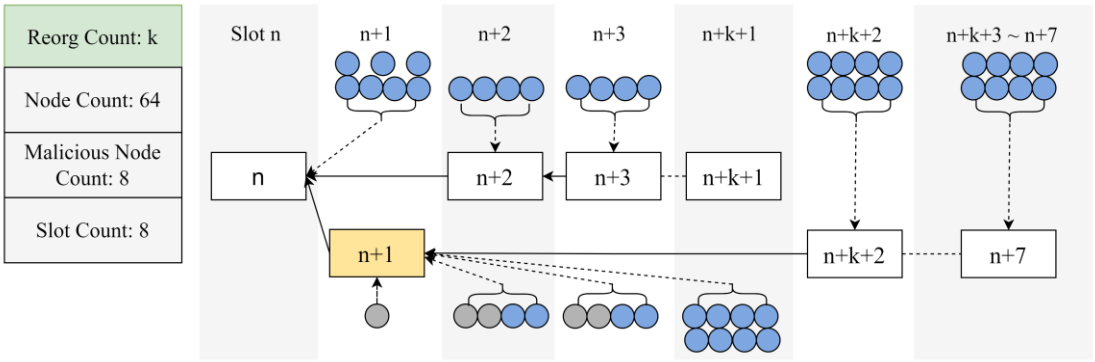

Figure 11. Specific process of fine-grained reorganization.

Figure 10 shows the chain views of honest nodes A and B during the voting process. The upper part of Figure 10 shows the chain view of node B after time T. At this time, it observes that fork chain α has two honest nodes voting for B_{n+2}, and fork chain β has three nodes voting for B_{n+1}, so B decides to vote for fork chain β. The lower part of Figure 10 shows the chain view of node A after time T. At this time, it observes that fork chain α has two honest nodes voting for B_{n+2}, and fork chain β has only one node voting for B_{n+1}, so A decides to vote for fork chain α. In the second half of the time of the $(n+3)$-th slot, after the message is fully broadcast, the chain views of A and B tend to be consistent. At this time, the weights of fork chain α and fork chain β are close to the same. If the attacker wants to increase the length of the reorganized block, it can continue to use this method to attack.

Figure 11 introduces the specific process of fine-grained reorganization in detail. The total number of nodes is 64, the number of malicious nodes controlled by the attacker is 8, one epoch is divided into 8 slots, and the length of the reorganized block is k. The specific steps of the attack include: (1) In the first step, malicious node P_{n+1}, as the proposer of slot S_{n+1}, generates block B_{n+1} but does not broadcast the

block immediately. Implement refined control message strategy and wait for the generation and voting of block S_{n+2}. (2) In the second step, when the voting time of S_{n+2} comes, P_{n+1} broadcasts the block to the network and attaches a vote $A_{p,n+1}$ for $n+1$. After that, the honest nodes H_C and H_D in the committee of S_{n+2} receive the voting message $A_{P,n+1}$ and judge that the weight W_{n+1} of the $n+1$ branch is greater than the weight W_{n+2} of the $n+2$ branch (since the current slot is in S_{n+2}, the block of this slot does not accumulate weight). At this time, the two nodes generate voting messages $A_{C,n+1}$ and $A_{D,n+1}$ and broadcast them. The other four honest nodes H_E, H_F, H_G, and H_H did not receive the block B_{n+1} and voting message of $n+1$ in time, so they generated votes for $n+2$ $A_{E,n+2}, A_{F,n+2}, A_{G,n+2}, and A_{H,n+2}$. (3) In the third step, the two malicious nodes M_A and M_B in slot S_{n+2} wait for the voting opportunity t_{n+2} after the generation of block B_{n+3} of $n+3$. When the opportunity comes, M_A and M_B send votes $A_{A,n+1}$ and $A_{B,n+1}$for $n+1$ to the network. After a period of time, the two nodes H_I and H_J in slot $n+3$ receive $A_{A,n+1}$ and $A_{B,n+1}$, and determine that the branch weight W_{n+1} of $n+1$ is greater than the weight W_{n+3} of $n+3$. At this time, H_I and H_J vote $A_{I,n+1}$ and $A_{J,n+1}$ for $n+1$. The other four honest nodes H_K, H_L, H_M, H_N did not receive B_{n+1} and its voting message in time, so they voted $A_{K,n+3}, A_{L,n+3}, A_{M,n+3}, A_{N,n+3}$ for B_{n+3}. (4) Finally, when it comes to slot S_{n+k}, the attacker finds that there is only one block left, so it directly releases the malicious node vote of $n+k$ in advance. After that, all the honest nodes of S_{n+k+1} calculate W_{n+k} and W_{n+1}, see Equation 4.5 and Equation 4.6.

$$W_{n+k} = WB_n + \sum_{i=2}^{k} WB_{n+i} = 4k+3 \tag{4.5}$$

$$W_{n+1} = WB_n + WB_{n+1} = 4k+4 \tag{4.6}$$

Through calculation, it is found that W_{n+1} is greater than W_{n+k}, so all committee nodes W_{n+k+1} in slot $n+k+1$ vote for B_{n+1}, and the attack is completed. In the best case, only $2k-1$ malicious nodes are needed to complete this attack.

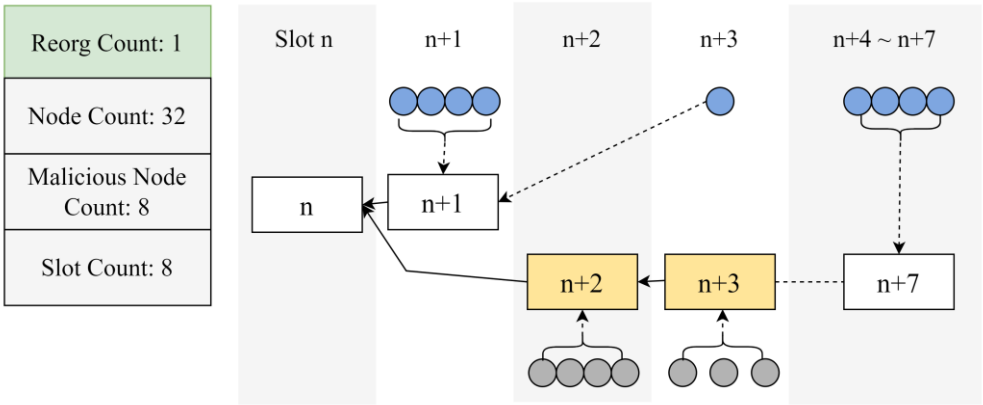

Figure 12. Specific process of one-block ex-post reorganization.

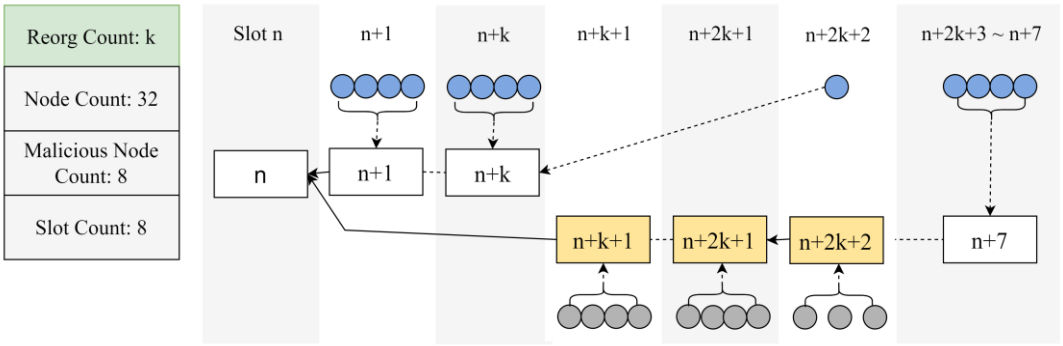

Figure 13. Specific process of k-blocks ex-post reorganization

4.5. Ex-post reorganization

This paper designs an ex-post reorganization attack method and introduces the specific process of ex-post reorganization in detail. The environment parameters of the attack are set to be consistent with ex-ante reorganization. The total number of nodes is 32, the number of malicious nodes controlled by the attacker is 8, and one epoch is divided into 8 slots. First, the process of the attacker reorganizing 1 block is explained, and then the general process of reorganizing k blocks is derived. Through these processes, we can deeply understand how the attacker uses malicious nodes to implement ex-post reorganization attacks. After that, this paper simulates the attack strategy and calculate the success probability of the attack through experimental simulation.

Figure 12 shows the specific process of one-block ex-post reorganization. The square blocks in the figure are blocks generated by the slot, the gray circles are malicious nodes, and the blue circles are honest nodes. The attack steps include: (1) The first step, malicious node as the proposer P_{n+2} of slot S_{n+2}, violates the weight priority principle of the fork selection algorithm, and decides to append to B_n and generate block B_{n+2}.(2) The second step, in slot S_{n+2}, according to the GHOST algorithm, the voting count starts from the block of the previous slot, so in this slot, the honest node only votes for B_{n+1}, and B_{n+2} only has malicious nodes voting. (3) The third step, in slot S_{n+3}, since the weight of W_{n+2} is 4, which is the same as W_{n+1}, the proposer of S_{n+3} must also be a malicious node for the attack to succeed. At this time, the proposer appends the block to B_{n+2}. After the B_{n+3} block is broadcast, assuming that the other honest nodes of S_{n+3} The node votes for B_{n+3}; finally, in slot S_{n+4}, the proposer of this slot calculates the weight W_{n+3} to be 7, which is greater than the weight of W_{n+1}, which is 5. Therefore, according to the GHOST algorithm, the proposer appends the new block to B_{n+3}, so B_{n+1} is discarded and the consensus is reorganized.

Figure 13 shows the specific process of k-blocks ex-post reorganization. The specific steps for a successful attack include: (1) The first step, the malicious node, as the proposer P_{n+k+1} of slot S_{n+k+1}, decides to append the block to B_n and broadcast it. (2) The second step, in slot S_{n+k+1}, according to the GHOST algorithm, the voting count starts from the block of the previous slot, so S_{n+k+1} only has malicious nodes voting. (3) The third step, in slot S_{n+2k+2}, since the weight W_{n+2k+1} calculated by the committee of this slot is consistent with W_{n+k}, P_{n+2k+2} must also be a malicious node, and the proposer appends the block to B_{n+2k+1}. Assume that after the block B_{n+2k+2} is broadcasted, there are two other malicious nodes in slot S_{n+2k+2} that vote for the block generated by this slot. Finally, in slotS_{n+2k+3}, the proposer P_{n+2k+3} of this slot calculates that the weight of W_{n+2k+2} is $2k+3$, which

is greater than the weight of W_{n+k} $2k+1$. Therefore, according to the GHOST algorithm, it appends the new block to B_{n+2k+2}, so blocks $B_{n+1}, \cdots, Bn+k$ are discarded, and the consensus undergoes a k-block reorganization.

5. Experiments

By analyzing short-range reorganization attacks, we can extract the key parameters that affect the attack results. After setting the key parameters, we can simulate the attack method and launch the attack. This paper designs an algorithm for experimental simulation and calculate the probability of success in an epoch. The key parameters are shown in Table 3:

Table 3. Key parameters of the short-range reorganization model.

Parameter	Symbol	Definition
Number of nodes	N	Integer type. The total number of nodes participating in the consensus. The default value is 4096.
Malicious node ratio	m	Floating point type, value range is 0 to 1. The ratio of malicious nodes controlled by the attacker to the total number of nodes. The default value is 0.3.
Slot number	S	Integer type. The number of slots in a single epoch in the consensus algorithm. Currently, the value of Ethereum mainnet is 32, so the default value is 32.
Reorganization model	R	Enumeration type. Includes three modes: ex-ante reorg, fine-grained reorg, and ex-post reorg. The default value is ex-ante reorg.
Degree of honest nodes forking	F	Floating point type, ranging from 0 to 1. The degree of honest node forking when fine-grained reorganization is implemented. If the honest node set is forked into two, the value is 1. The default value is 0
Voting weight enhancement	Δw	Floating point type, value range is 0 to 1. Enhance the proposer weight. If it is 0, it means that this method is not enabled. The default value is 0.
Cross-slot voting weight reduction	Δwa	Floating point type, value range is 0 to 1. Degrade the cross-slot voting weight. If it is 1, it means that this method is not enabled. The default value is 1.
Blockchain reorganization length	B	Integer type. The number of short-range reorganization blocks. The minimum value is 1 and the maximum value is $S-1$. The default value is 1.
Simulations count	c	Integer type. The number of experimental simulations, that is, the number of simulation epochs. The default value is 10,000.
Probability of success	P	Floating point type. Short-range reorganization attacks success probability.

According to the short-range reorganization model, the definition of key parameters is shown in Equation 5.1:

$$P = f(N, m, S, R, F, \Delta w, \Delta wa, B, c) \tag{5.1}$$

Where f is the short-range reorganization attacks model. Other terms that are not listed in the key parameter table but have been mentioned in the Ethereum basic settings include: the number of malicious nodes $M = N \cdot m$, the number of committee nodes $C = N/S$. Regarding the weight value calculated in

the fork selection algorithm, the weight of 1 node is defined as 1, and the weight of the proposer after the power is raised is $W = C \cdot \Delta w$. The function f is computed through different algorithms depending on the value of parameter R. Specifically: when $R = ex\text{-}ante\_reorg$, f is calculated using Algorithm 1; when $R = ex\text{-}post\_reorg$, Algorithm 2 is employed; and when $R = fine\_grained\_reorg$, the computation follows Algorithm 3.

Algorithm 1: Simulating ex-ante short-range reorganization attack

Input: $malicious\_percent$: malicious node ratio, $rogn$:reorganization length, $simulate\_count$: simulation rounds, $node\_count$: total nodes, $slot\_number$: slot number

Output: attack success probability

Initialize parameters:
$malicious\_count \leftarrow malicious\_percent \times node\_count$
$node\_per\_slot \leftarrow node\_count/slot\_number$
$attack\_suc \leftarrow 0$
for $id \leftarrow 1$ **to** $simulate\_count$ **do**
    Shuffle nodes x randomly
    Initialize $malicious\_header[\,], normal[\,], malicious[\,]$
    for $i \leftarrow 1$ **to** $slot\_number$ **do**
        Assign $node\_per\_slot$ nodes to $current\_slot$
        Record leader status
        Count $normal[i]$ and $malicious[i]$
    for $i_2 \leftarrow 1$ to $slot\_number$ do
        if $malicious\_header[i_2] == 1$ **then**
            $item\_malicious \leftarrow 0, item\_normal \leftarrow 0$
            $continue\_0 \leftarrow 0$
            for $i_3 \leftarrow i_2$ **to** $slot\_number$ **do**
                if in initial adversary leader phase **then**
                $item\_malicious \leftarrow item\_malicious + fault[i_3]$
                else
                $item\_malicious \leftarrow item\_malicious + fault[i_3]$
                $item\_normal \leftarrow item\_normal + normal[i_3]$
                if $continue\_0 \geq rogn$ **and**
                $(item\_fault + temp\_fault) > (item\_normal + temp\_normal)$ **then**
                $attack\_suc \leftarrow attack\_suc + 1$
                terminate outer loop
return $attack\_suc/simulate\_count$

Algorithm 2: Simulating ex-post short-range reorganization attack

Input: $malicious\_percent$: malicious node ratio, $rogn$: reorganization length, $simulate\_count$: simulation rounds, $node\_count$: total nodes, $slot\_number$: time slot number

Output: attack success probability

$malicious\_count \leftarrow malicious\_percent \times node\_count$
$node\_per\_slot \leftarrow node\_count/slot\_number$
$attack\_suc \leftarrow 0$
for $id \leftarrow 1$ **to** $simulate\_count$ **do**
    Shuffle nodes x randomly
    Initialize $malicious\_header[\,], normal[\,], malicious[\,]$
    for $i \leftarrow 0$ **to** $node\_count - 1$ **do**
        $slot_i \leftarrow i/node\_count\_per\_slot$
        if $i \bmod node\_count\_per\_slot == 0$ **then**
            $malicious\_header[slot_i] \leftarrow (x[i] < malicious\ count)?\ 1 : 0$
        if $x[i] < malicious\_count$ **then**
            $malicious[slot_i] \leftarrow malicious[slot_i] + 1$
        else
            $normal[slot_i] \leftarrow normal[slot_i] + 1$
    for $i_2 \leftarrow rogn$ **to** $slot\_number - 1$ **do**
        if $malicious\_header[i_2] == 1$ **then**
            for $i_4 \leftarrow (i_2 - rogn)$ **to** $i_2 - 1$ **do**
                $item\_malicious += normal[i_4]$
            $item\_malicious\_count \leftarrow 0$
            for $i_3 \leftarrow i_2$ **to** $slot\_number - 1$ **do**
                $item\_malicious\_count \leftarrow item\_malicious\_count + malicious[i_3]$
                if $item\_malicious\_count > item\_normal\_count$ **then**
                    $attack\_suc \leftarrow attack\_suc + 1$
                    terminate outer loop
return $attack\_suc/simulate\_count$

Next, this paper conducts attack simulation experiment in 3 ways: First, we analyze the impact of the malicious node ratio on success. Attackers can achieve success by controlling more malicious nodes. Then, we analyze whether the increase in the number of nodes affects success. Finally, we analyze the impact of the degree of honest node forking on success when fine-grained reorganization is implemented.

Algorithm 3: Simulating fine-grained short-range reorganization attack

Input: $fork\_degree$: honest node forking degree, $malicious\_percent$: malicious node ratio, $rogn$: reorganization length, $simulate\_count$:simulation rounds, $node\_count$: total nodes, $slot\_number$: time slot number

Output: attack success probability

$malicious\_count \leftarrow malicious\_percent \times node\_count$
$node\_per\_slot \leftarrow node\_count/slot\_number$
$normal\_to\_malicious\_per \leftarrow fork\_degree/(1 + fork\_degree)$
$normal\_to\_malicious\_per \leftarrow 1/(1 + fork\_degree)$
$attack\_suc \leftarrow 0$
for $id \leftarrow 0$ **to** $simulate\_count - 1$ **do**
    Shuffle node x randomly
    Initialize $malicious\_header[\,], normal[\,], malicious[\,]$
    for $i_1 \leftarrow 1$ **to** $node\_count - 1$ **do**
        $slot_i \leftarrow i_1/node\_count\_per\_slot$
        if $i_1 \; mod \; node\_count\_per\_slot == 0$ **then**
            $malicious\_header[slot_i] \leftarrow (x[i_1] < malicious \; count)$
        if $x[i_1] < malicious\_count$ **then**
            $malicious[slot_i] \leftarrow malicious[slot_i] + 1$
        else
            $normal[slot_i] \leftarrow normal[slot_i] + 1$
    for $i_2 \leftarrow 0$ to $slot\_number - 1$ **do**
        if $malicious\_header[i_2] == 1$ **then**
            $item\_malicious\_count \leftarrow malicious[i_2]$
            $temp\_rego \leftarrow 1, start\_0 \leftarrow -1$
            for $i_3 \leftarrow i_2 + 1$ **to** $slot\_number - 1$ **do**
                if $malicious\_header[i_3] == 0$ **and** $start\_0 < 0$ **then**
                    $start\_0 \leftarrow i_3$
                $item\_malicious\_count \leftarrow item\_malicious\_count + malicious[i_3]$
                $normal\_to\_malicious \leftarrow normal\_to\_malicious\_per \times normal[i_3]$
                $normal\_to\_normal \leftarrow normal[i_3] - normal\_to\_malicious$
                if $normal\_to\_normal \geq normal\_to\_malicious + 1$ **then**
                    $needed\_malicious \leftarrow normal\_to\_normal \; - \; normal\_to\_malicious$
                else if $normal\_to\_normal == normal\_to\_malicious$ **then**
                    $needed\_malicious \leftarrow 2$
                else
                    break
                if $item\_malicious\_count \geq needed\_malicious$ **then**
                    $temp\_rego \leftarrow temp\_rego + 1$
                    $item\_malicious\_count \leftarrow item\_malicious\_count - needed\_malicious$
                else
                    b**reak**
            if $start\_0 \geq 0$ **and** $temp\_rego > rogn$ **then**
                $attack\_suc \leftarrow attack\_suc + 1$
                break
return $attack\_suc/simulate\_count$

5.1. Impact of malicious node ratio on success

In order to increase the probability of a short-range reorganization, the obvious way for an attacker is to control more malicious nodes. This section analyzes the impact of malicious node ratio on ex-ante reorganization and ex-post reorganization. The key parameters are malicious node ratio m, reorganization mode R and reorganization block length B. Other parameters are default values. We analyze how changes in these parameters affect the success rate. The number of experiments is calculated based on the length of one year. That is to say, if a reorganization does not occur in a year, the probability of its occurrence can be ignored in this paper. If calculated according to the parameters of the Ethereum mainnet, the duration of each slot is 12 seconds, and the number of epochs in a year is 82125. The key parameters are shown in Equation 5.2:

$$P = f(4096, m, 32, R, 0{,}0{,}1, B, 82125) \tag{5.2}$$

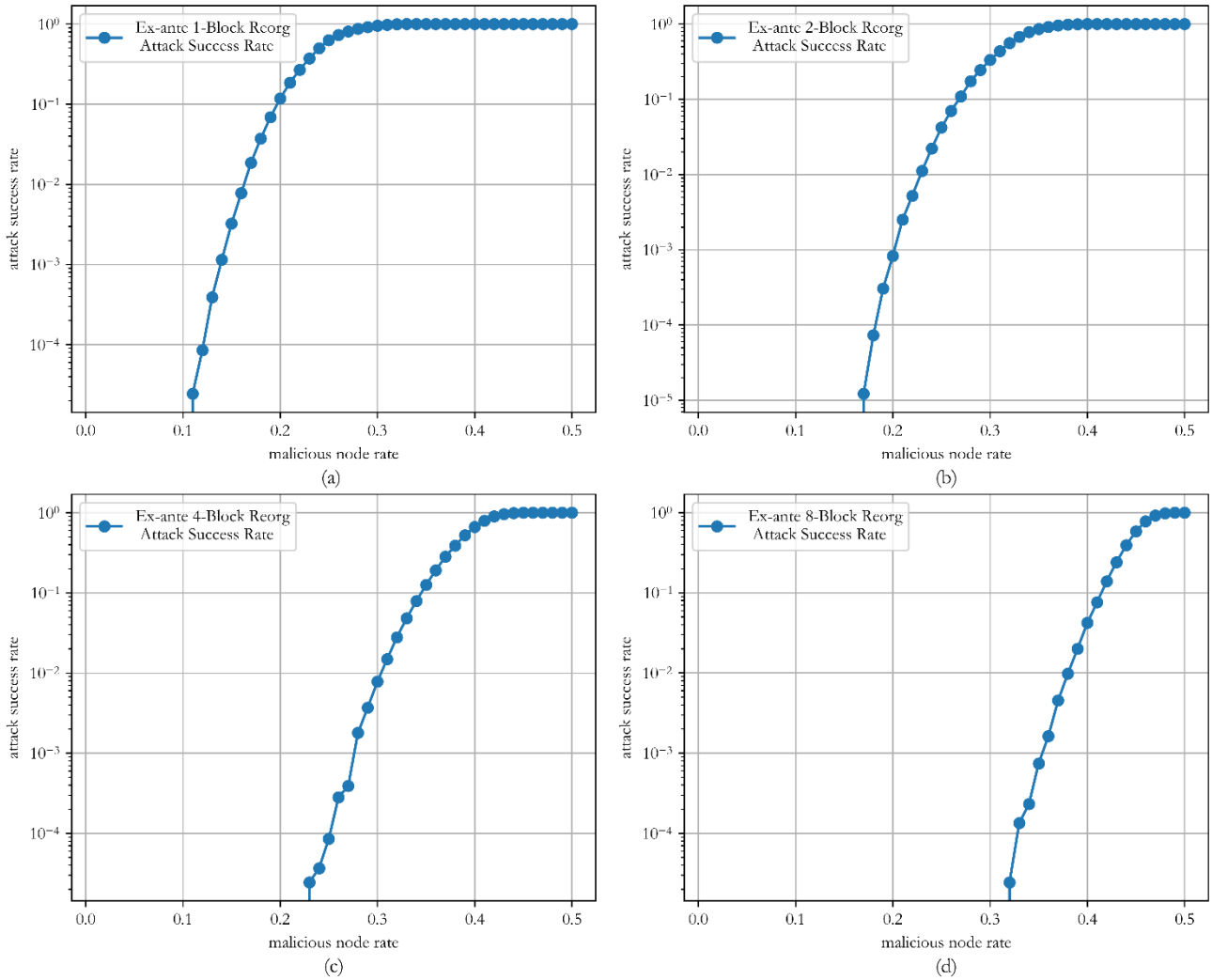

Figure 14. Impact of malicious node ratio on ex-ante reorganization attack success rate.

Figure 14 shows the effect of different malicious node ratios on ex-ante reorganization attack success from 1 block to 8 blocks. From Figure 14(a), we can conclude that: (1) The attack success rate is low when the malicious node ratio is low. When the malicious ratio is less than 0.11, the attack success

rate is nearly 0, which shows that ex-ante short-range reorg attacks fail in most cases. (2) The attack success rate gradually increases as the malicious node ratio increases. At a malicious node ratio of about 0.33, the attack success rate is close to 50%. (3) When the malicious node ratio exceeds 0.4, the attack success rate quickly reaches 100%. This shows that ex-ante short-range reorg attacks almost always succeed at high ratios. From Figure 14(b), only when the malicious node ratio exceeds 0.24 can the attacker launch a 2-block ex-ante short-range reorg attack within 1 year. When the malicious node ratio exceeds 0.35, ex-ante short-range reorg attacks always succeed. From Figure 14(c), only when the malicious node ratio exceeds 0.24 can the attacker launch a 4-block ex-ante short-range reorg attack within 1 year. When the malicious node ratio exceeds 0.40, ex-ante short-range reorg attacks always succeed. From Figure 14(d), the corresponding thresholds are 31% and 50% when reorganizing 8 blocks.

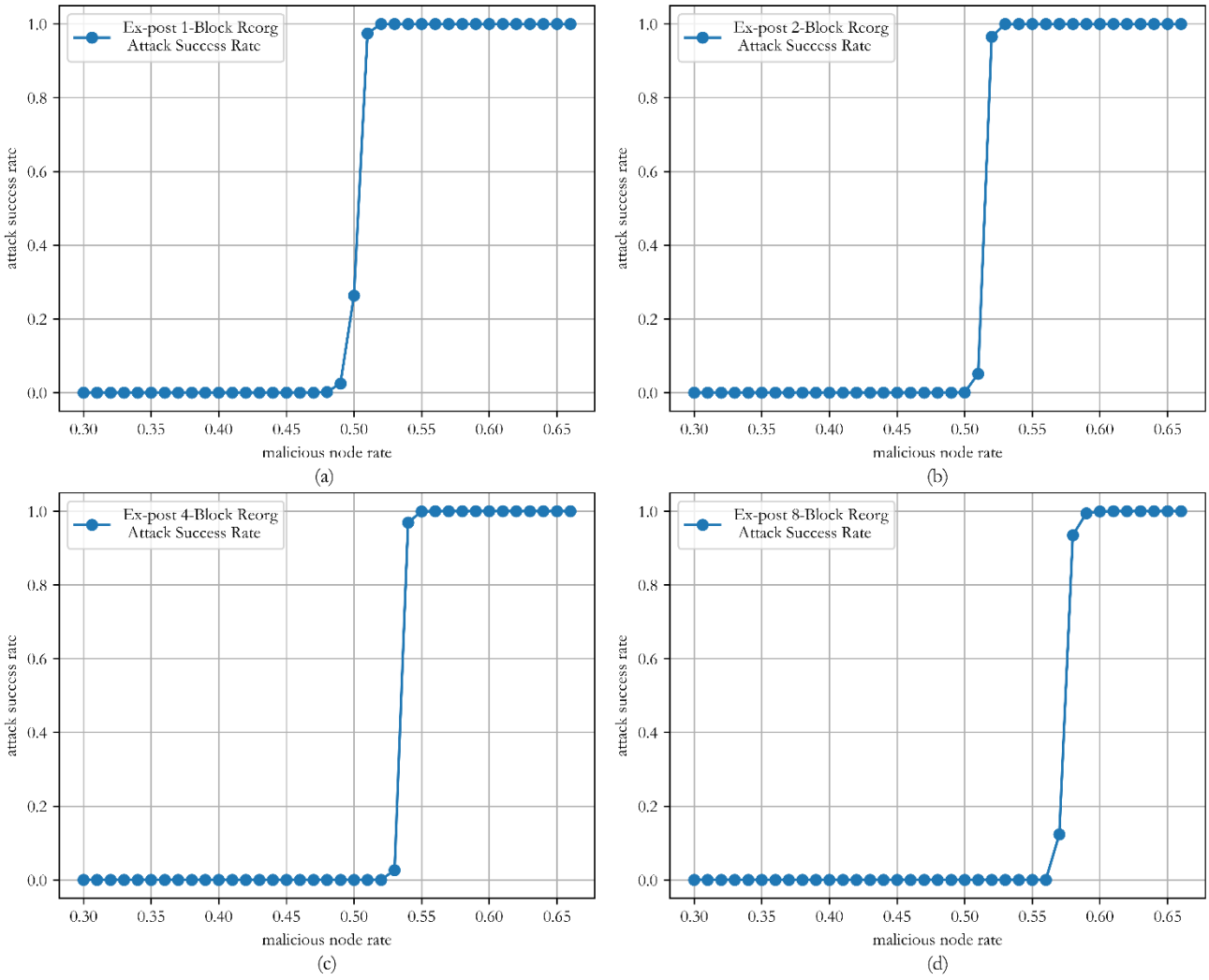

Figure 15. Impact of malicious node ratio on ex-post reorganization results.

Figure 15 shows the impact of different malicious node ratios on ex-post reorganization success from 1 block to 8 blocks. From Figure 15(a), when reorganizing one block, we can conclude that: (1) When the malicious node ratio is lower than 0.47, the attack success rate is 0. That is, the system is relatively safe and has not been attacked significantly. (2) Starting from 0.48, the attack success rate gradually increases but is still low. When it is around 0.5, the attack success rate rises rapidly to about 26%. (3) When the malicious node ratio exceeds 0.5, the attack success rate quickly approaches 100%,

and the system's security drops sharply. From Figure 15(b), only when the malicious node ratio exceeds 0.50 can the attacker launch a 2-block ex-post short-range reorg attack within 1 year. When the malicious node ratio exceeds 0.52, ex-post short-range reorg attacks always succeed. From Figure 15(c), the corresponding thresholds are 0.53 and 0.54 when reorganizing 4 blocks. From Figure 15(d), the corresponding thresholds are 0.57 and 0.58 when reorganizing 8 blocks. Therefore, compared with an ex-ante reorg attack, the ratio of malicious nodes required for an ex-post reorg attack is much larger, and regardless of the number of blocks reorganized, the success rate is very low.

5.2. Impact of node number on success

In actual blockchain networks, the number of nodes participating in the consensus often changes dynamically. As blockchains based on the PoS algorithm become more and more popular, the number of participants is expected to increase significantly. This section analyzes whether the attack success rate remains unchanged as the number of consensus nodes increases while the malicious node ratio remains unchanged. From the perspective of network security, that is, whether the increase in the number of nodes brings additional difficulty to short-range reorganization attacks. The key parameters are the node number N, and the reorganization mode R. The other parameters are the default values. This section analyzes how changes in these parameters affect the success rate. The range of the total number of nodes is set from 960 to 19200. The key parameters are shown in Equation 5.3:

$$P = f(N, m, 32, R, 0{,}0{,}1, B, 82125) \tag{5.3}$$

Figure 16 shows the impact of adjusting the node number on ex-ante and ex-post reorg attacks. Figure 16(a) shows the impact of the increase in the node number on success when the malicious node ratio is 0.3. The result shows that as the node number increases, the attack success rate gradually decreases, stabilizing around 90%. Figure 16(b) shows the impact of the increase in the node number on success when the malicious node ratio is 0.5. The result shows that the attack success rate decreases exponentially as the node number increases, and the trend does not slow down. Therefore, from the experimental results, the increase in the node number increases the difficulty of ex-ante reorganization and ex-post reorganization. The node number has a greater impact on ex-post reorganization in comparison.

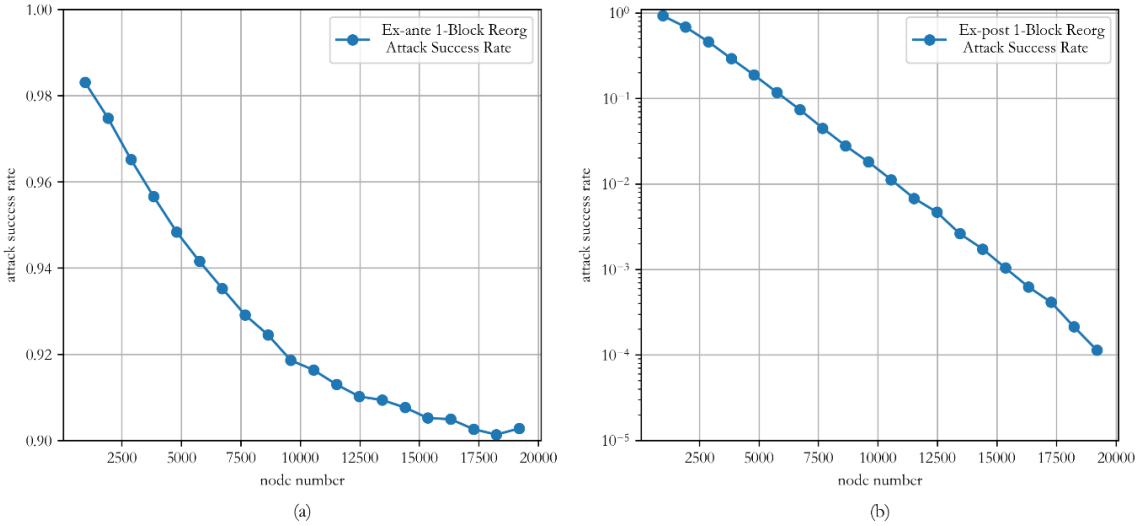

Figure 16. Impact of node number on attack results.

5.3. Impact of honest node forking degree on success

To further reduce the number of nodes required for short-range reorganization attacks to succeed, the honest node can be forked by precisely controlling the sending time, with one part supporting the chain generated by the honest node and the other part supporting the chain generated by the malicious node. The degree of forking depends on the accuracy of the estimation of sending time. Attackers usually test the average sending time by sending messages in advance and then estimate the sending time. However, the network latency is dynamic, so this section analyzes the impact of different node forking degrees on the attack's success rate. As is defined in Equation 4.4, forking degree $F = H_m/H_h$, where H_m represents the number of malicious chains supported by the honest node after forking; H_h represents the number of honest chains supported by the honest node. When F equals 0, it means that the attacker has not forked any honest node, and all honest nodes support the chain generated by the honest node. This situation is equivalent to the attacker not considering the precise control mode, and it is easiest to create a condition with no forking degree. When F equals 1, it means that the attacker divides the honest node set equally, half of which supports the chain generated by the malicious node, and the other half supports the chain generated by the honest node. This situation is the best condition for implementing this attack.

The key parameters of this experiment are the honest node forking degree F and the reorganized block length B. According to the analysis of the impact of malicious node ratio on the attack, the attacker only needs to control more than 10% of the nodes to launch an attack successfully. In contrast, this strategy theoretically only needs less to launch an attack. So, in this section's experiment, the malicious node ratio is set to 10%, and other parameters are the default values. We analyze how changing the forking degree parameter affects the success rate. The key parameters are shown in Equation 5.5:

$$P = f(4096,0.1,32, fine\_grained\_reorg, F, 0,1, B, 82125) \quad (5.4)$$

Figure 17 shows the impact of the honest node forking degree on the success of the reorg attacks from 2 to 16 blocks. As the forking degree increases, the probability of successful attack increases, and the probability stabilizes after reaching a certain level. The minimum forking degrees required for reorganizing 2, 4, 8, and 16 blocks are 0.04, 0.36, 0.5, and 0.58, as shown in Figures 17(a), 17(b), 17(c), and 17(d) respectively. The highest degree reached by the reorganization of different numbers of blocks decreases as the length of the reorganized block increases. The highest success rate of reorganizing 2 blocks can reach 96% in Figure 17(a), while the success rate of reorganizing 16 blocks can only reach 82% in Figure 17(d).

Through this experiment, we can see that the honest node forking strategy can greatly reduce the difficulty of attack. An attacker who controls a small number of nodes can also initiate the reorganization of long blocks. Compared with the strategy without honest node forking, its weakness is that the attacker needs to expose the block in advance, and its malicious characteristics are relatively obvious.

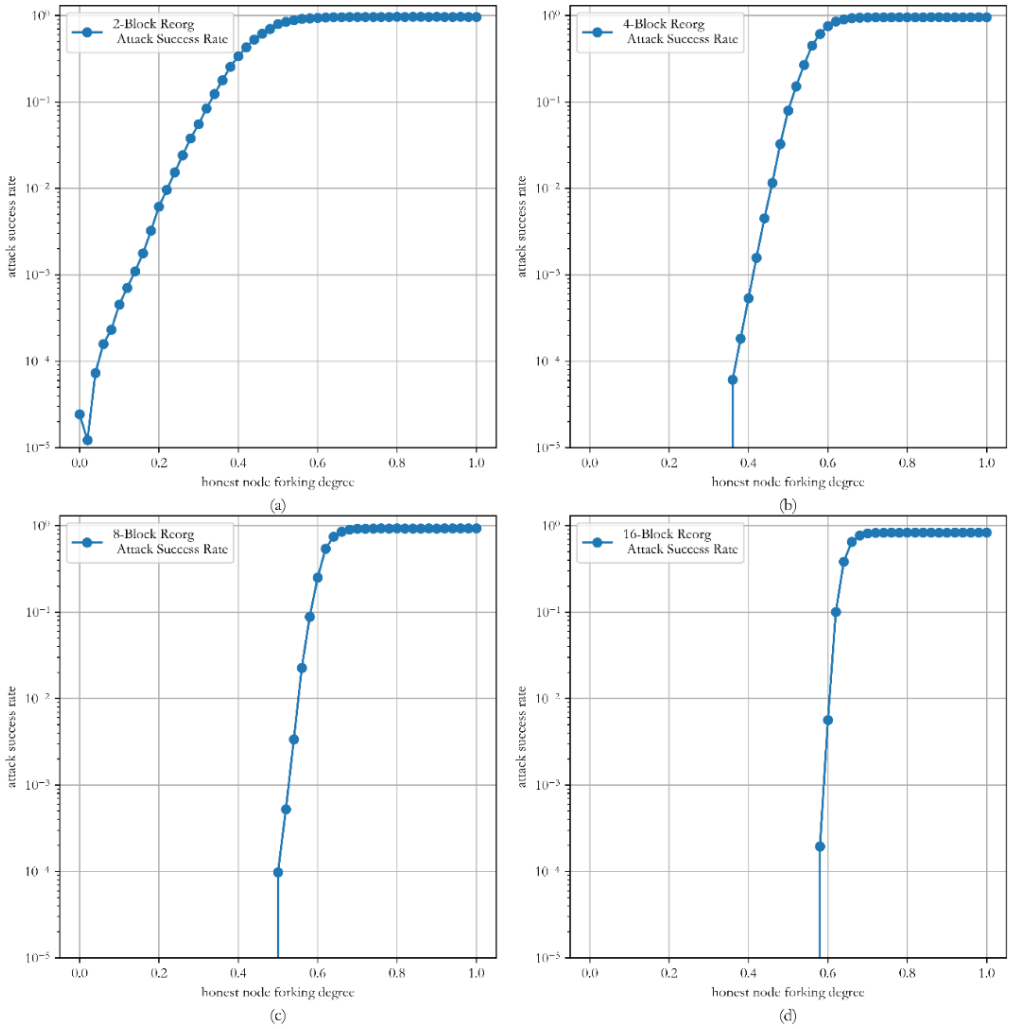

Figure 17. Impact of the honest node forking degree on attack result.

The experiments are conducted on a system running Windows 11 as the operating system, equipped with a 12th Gen Intel® Core™ i7-1260P processor operating at 2.10 GHz and 16.0 GB of RAM. The development environment is configured using Visual Studio Code (VSCode) as the primary code editor, with all implementations being executed through a hybrid programming approach combining C++ and Python.

6. Conclusion

This paper systematically investigates Ethereum's consensus algorithm and known attack methods, developing a comprehensive model for short-range reorganization attacks in Proof-of-Stake (PoS) systems. We categorize attack strategies into three distinct modes: ex-ante reorganization, fine-grained reorganization, and ex-post reorganization. Through systematic experimentation, the study quantitatively analyzes how key factors, including malicious node ratio, honest node forking degree, and reorganization block length, influence attack outcomes.

The experimental findings offer critical insights for enhancing blockchain security against short-range reorganization attacks. The ratio of malicious nodes required for ex-post reorganization is much larger than for ex-ante reorganization. Increasing the node number increases the difficulty of ex-ante and ex-post reorganization. The node number has a greater impact on ex-post reorganization in comparison.

Adopting a fine-grained reorganization strategy can greatly reduce the difficulty of a reorganization attack. Honest node forking strategy can greatly reduce the difficulty of attack.

While the research highlights three specific types of short-range reorganization attacks, it does not address the practical implementation challenges of the described attacks in live Ethereum environments or evaluate how Ethereum's existing safeguards might mitigate their impact. Although real-world Ethereum cases were not analyzed, the quantified non-linear relationships between network scale, honest node forking degree, and attack results provide a mathematical modeling foundation for optimizing consensus parameters and designing defense strategies. These results enable developers to strategically strengthen systems by raising economic and technical barriers to short-range reorganization attacks, ensuring defenses align with evolving threat models.

Acknowledgments

We highly appreciate financial support from the National Natural Science Foundation of China (grant number 72271013), and Beihang University (grant number JKF-20240623).

Conflicts of interests

The authors declare no conflict of interest.

Authors' contribution

Conceptualization, Z.Z.Y., Z.J.H.; methodology, Z.Z.Y., Z.J.H.; software, Z.Z.Y.; validation, Z.J.H., Z.Z.Y., J.R.; formal analysis, Z.Z.Y.; investigation, Z.Z.Y.; resources, Z.Z.Y.; data curation, Z.Z.Y.; writing—original draft preparation, Z.Z.Y.; writing—review and editing, Z.J.H., J.R.; visualization, Z.Z.Y.; supervision Z.J.H.; project administration, Z.J.H.; funding acquisition, Z.J.H. All authors have read and agreed to the published version of the manuscript.

References

[1] Slowmist. 2023 Blockchain Security and Anti-Money Laundering Annual Report[EB/OL]. Available: https://slowmist.medium.com/2023-blockchain-security-and-anti-money-laundering-annual-report-0d556e879fbb (accessed on 1 January 2024).

[2] Kohli V, Chakravarty S, Chamola V, Sangwan KS, Zeadally S. An analysis of energy consumption and carbon footprints of cryptocurrencies and possible solutions. *Digital Commun. Networks* 2023, 9(1):79–89.

[3] Nakamoto, Satoshi. Bitcoin: a peer-to-peer electronic cash system. *Decentralized Bus. Rev.* 2008.

[4] King S, Scott N. PPcoin: peer-to-peer crypto-currency with proof-of-stake. Self-published paper, August 2012, 19(1).

[5] Sayeed S, Marco-Gisbert H. Assessing blockchain consensus and security mechanisms against the 51% attack. *Appl. Sci.* 2019, 9(9):1788.

[6] Li W, Andreina S, Bohli J-M, Karame G. Securing Proof-of-Stake Blockchain Protocols. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Oslo, Norway, September 14–15, 2017, pp. 297–315.

[7] Kwon, J. Tendermint: Consensus without Mining. 2014. Available:

https://www.weusecoins.com/assets/pdf/library/Tendermint%20Consensus%20without%20Mining.pdf (accessed on 22 April 2025)

[8] Buchman E. Tendermint: Byzantine fault tolerance in the age of blockchains. Diss. University of Guelph, 2016.

[9] Gilad Y, Hemo R, Micali S, Vlachos G, Zeldovich N. Algorand: scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, Shanghai, China, October 28–31, 2017, pp. 51–68.

[10] Micali S, Michael R, Salil V. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, New York, USA, October 17–19, 1999, pp. 120–130.

[11] Buterin V, Virgil G. Casper the friendly finality gadget. *arXiv* 2017, arXiv:1710.09437.

[12] Buterin V, Reijsbergen D, Leonardos S, Piliouras G. Incentives in Ethereum's hybrid Casper protocol. *Int. J. Network Manage.* 2020, 30(5):e2098.

[13] Buterin V. Proposal for mitigation against balancing attacks to LMD GHOST. 2020. Available: https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation(accessed on 22 April 2025).

[14] Sompolinsky Y, Zohar A. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security: 19th International Conference, FC 2015*, San Juan, USA, January 26–30, 2015.

[15] Álvarez IA, Gramlich V, Sedlmeir J. Unsealing the secrets of blockchain consensus: A systematic comparison of the formal security of proof-of-work and proof-of-stake. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, Avila, Spain, April 8–12, 2024, pp. 278–287.

[16] Torres CF, Camino R. Frontrunner jones and the raiders of the dark forest: an empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*, Vancouver, Canada, August 11–13, 2021, pp. 1343–1359.

[17] Neuder M, Moroz DJ, Rao R, Parkes DC. Low-cost attacks on Ethereum 2.0 by sub-1/3 stakeholders. *arXiv* 2021, arXiv:2102.02247.

[18] Schwarz-Schilling C, Neu J, Monnot B, Asgaonkar A, Tas EN, *et al.* Three Attacks on Proof-of-Stake Ethereum. In *Financial Cryptography and Data Security*, Grand Anse, Grenada, May 2–6, 2022, pp. 560–576.

[19] Otsuki, Kai, Ryuya Nakamura, and Kazuyuki Shudo. "Impact of saving attacks on blockchain consensus." *IEEE Access* 2021, 9:133011–133022.

[20] Dwork C, Nancy L, Larry S. Consensus in the presence of partial synchrony. *J.ACM* 1988, 35(2): 288–323.

[21] Neu J, Tas EN, Tse D. Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*, San Francisco, USA, May 24–27, 2021, pp. 446–465.

[22] Buterin V, Hernandez D, Kamphefner T, Pham K, Qiao Z, *et al.* Combining GHOST and casper. *arXiv* 2020, arXiv:2003.03052.

[23] Zamfir V. Casper the friendly ghost: a correct by construction blockchain consensus protocol. Whitepaper: https://github.com/ethereum/research/blob/master/papers/caspertfg/caspertfg. pdf, 2017.

[24] Neu, Joachim, Ertem Nusret Tas, and David Tse. "Two more attacks on proof-of-stake GHOST/Ethereum." *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, Los

Angeles, USA, November 7, 2022, pp. 43–52.

[25] Eyal I, Emin GS. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM* 2018, 61(7):95–102.

[26] Rosenfeld M. Analysis of hashrate-based double spending. *arXiv* 2014, arXiv:1402.2009.

[27] Eskandari, S., Moosavi, S., Clark, J. SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In *Financial Cryptography and Data Security*. Gerhard Goos, Juris Hartmanis, Eds. Lecture Notes in Computer Science, Springer, Cham. 2020.

[28] Daian P, Goldfeder S, Kell T, Li Y, Zhao X, *et al.* Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv* 2019, arXiv:1904.05234.