A Study on Prevention and Automatic Recovery of Blockchain Networks Against Persistent Censorship Attacks

JAE-SEOK KIM, JIN-MYEONG SHIN, SEOK-HWAN CHOI, AND YOON-HO CHOI (Member, IEEE)

1School of Computer Science and Engineering, Pusan National University, Busan 46241, Republic of Korea
2Division of Software, Yonsei University, Wonju-si 26493, Republic of Korea

Corresponding author: Yoon-Ho Choi (yhchoi@pusan.ac.kr)

This work was supported in part by the Institute for Information & Communications Technology Planning & Evaluation (IITP) funded by the Korean Government through the Ministry of Science and ICT (MSIT), South Korea, funded by the Convergence Security Core Talent Training Business through Pusan National University under Grant 2022-0-01201; and in part by MSIT under the Information Technology Research Center (ITRC) Support Program supervised by IITP under Grant IITP-2022-00164800.

ABSTRACT Blockchain techniques have been actively used in a wide range of applications owing to their unforgeable characteristics. Currently, many blockchain platforms are applying a pBFT-based Delegate PoS (DPoS) consensus mechanism in their networks, because it ensures a quick consensus process since only a few validators participate in the consensus process. However, the pBFT-based DPoS consensus mechanism has a limitation that a malicious cartel with more than 1/3 of the total stake can launch persistent censorship attacks which disrupt the consensus process. To address this limitation, we propose a new defense method against persistent censorship attacks. First, we introduce a hierarchical consensus architecture that consists of the main-validators and sub-validators. In our consensus architecture, sub-validators can disagree with the results of the consensus process of main-validators known as existing validators. As the number of disagreeing sub-validators increases, the cost of censorship attacks increases. Second, we propose a behavior-based credit-scoring function that can operate on our consensus architecture. Our proposed function changes the role of the validators based on their behavior, thereby disrupting the formation of malicious cartels and removing the persistence of censorship attacks. We show that our proposed defense method makes a blockchain network more tolerant and resistant to censorship attacks based on experimental results using validator information collected from a real blockchain network. We also demonstrate that our defense method can automatically recover the blockchain network from persistent censorship attacks.

INDEX TERMS Blockchain, consensus, censorship, defense, DPoS.

I. INTRODUCTION

With the deployment of Bitcoin [1], a concept of blockchain was introduced. Unlike a conventional network, where traffic and data are controlled by a single node, all traffic and data in a blockchain network are shared and saved by all nodes. Such characteristics render blockchain networks unforgeable. Based on these unforgeable characteristics, blockchain techniques have been actively applied to data-sensitive applications such as patient audit trails in healthcare [2] and monetary transactions in finance [3].

As a representative consensus mechanism, a Proof-of-Stake (PoS) consensus mechanism has been applied to almost blockchain networks because it eliminates the problems of the first consensus mechanism, a Proof-of-Work (PoW) consensus mechanism. In the PoS consensus mechanism, qualified nodes called validators determine which block will be attached to the main-chain through voting rather than mining in the PoW consensus mechanism. Because voting between validators has a much lower computational cost than block
mining, a blockchain network applying the PoS consensus mechanism can process more transactions at the same time than a network applying the PoW consensus mechanism.

In a public blockchain network applying the PoS consensus mechanism, the number of validators is not limited. Therefore, such a network has a limitation that the computational cost for communication increases exponentially as the number of validators increases. To solve this limitation, recent blockchain platforms such as Tendermint [4], EOS [5] and Steem [6] have applied a Delegate PoS (DPoS) consensus mechanism that limits the number of validators in their blockchain networks. In the DPoS consensus mechanism, a node that needs to be selected as a validator should have more than a certain amount of stake. Here, a certain amount of stake is quite large compared to the average stake of the nodes in the blockchain network. Thus, the DPoS consensus mechanism allows small and medium-sized stakeholders to assert influence in the consensus process by delegating their stake to the validators.

Since the DPoS consensus mechanism requires fewer validators than the PoS consensus mechanism, an algorithm which can reliably generate and finalize blocks with only a few validators should be the basis. In particular, a practical Byzantine fault tolerance (pBFT) algorithm [7], which derives reliable consensus results with less communication, has been used as the basis for many DPoS consensus mechanisms. Note that, the pBFT algorithm requires a minimum of \(3f + 1\) nodes to tolerate \(f\) node faults. Thus, because the influence of each node in the DPoS consensus mechanism is proportional to its stake, the consent of nodes with a stake of \(2/3(2f + 1/3f + 1)\) or more is required to finalize block safety in the blockchain network applying the pBFT-based DPoS consensus mechanism.

However, because the number of validators in such a network is too small and the influence of each validator is proportional to the stake, the formation of a malicious cartel between malicious validators with sufficient stake occurs easily. As a representative attack caused by the formation of the malicious cartel, a censorship attack [8], [9] that requires more than \(1/3\) of the stake paralyzes a blockchain network by disrupting the consensus process. For example, in the Cosmos [10] blockchain network, the top five validators out of the 100 validators have more than \(1/3\) of the total stake [11]. Therefore, in the worst-case scenario, they alone can form a malicious cartel and perform censorship attacks. To address it, existing blockchain networks applying the pBFT-based DPoS consensus mechanism have adopted their own defense methods [12], [13], such as voting power-based voting [4] or weighted voting to defend against censorship attacks. However, since such defense methods only increase the cost of the censorship attacks, they can be easily bypassed as a malicious cartel adds the stake. Furthermore, once censorship attacks occur, the blockchain network can be persistently paralyzed. As a result, we concluded that in order to prevent censorship attacks, we needed to not only raise the cost of censorship attacks but also eliminate their persistence.

In this paper, we propose a new defense method against persistent censorship attacks. First, we propose a hierarchical consensus architecture which increases the attack cost of the censorship attacks. In our hierarchical consensus architecture, we introduce the concept of a sub-validator that can disagree with the consensus results of existing validators, called main-validators. Sub-validators make it possible to finalize blocks that are censored by malicious main-validators. As a result, the cost of censorship attacks increases with the stakes of the sub-validators who disagree. Second, we propose a behavior-based credit-scoring function that can operate on our hierarchical consensus architecture. The purpose of our credit-scoring function is to disrupt the formation of malicious cartels and remove the persistence of censorship attacks. In order to achieve this purpose, the credit-scoring function calculates the validators’ scores based on their individual behavior and changes their group membership based on the calculated score. As a result, no validator can remain in a single group indefinitely, and forming a malicious cartel is difficult. Even if a malicious cartel forms via an online validator committee or messenger, the credit-scoring function can still prevent censorship attacks from persisting because it penalizes the main-validators, including the malicious cartel, for failing to maintain the blockchain network normally. Consequently, the credit-scoring function can automatically recover the blockchain network from censorship attacks.

The main contributions of this paper can be summarized as follows:

1) We propose a hierarchical consensus architecture that increases the attack cost of censorship attacks. To finalize a block in our consensus architecture, we introduce a new concept of sub-validator that forms a competitive landscape with the existing validator. Due to the check of the sub-validator, a malicious cartel existing in the existing validator group cannot easily perform censorship attacks.

2) We propose a behavior-based credit-scoring function that changes the membership of each validator group in each block generation process. By disrupting the malicious cartel, the proposed scoring method prevents persistent censorship attacks and enables the blockchain network to automatically recover from censorship attacks.

3) We prove the superiority of our defense method against censorship attacks from experimental results with using a list of validators in real blockchain networks. As a result, we show our proposed method make a blockchain network not only more tolerant and resistant to censorship attack, makes it possible to automatically recover from continuous censorship attacks through the experiments.

The remainder of this paper is organized as follows. First, we describe the background knowledge, including PoS and DPoS consensus mechanisms, pBFT algorithm, and existing countermeasures against censorship attacks in Section II.
In Section III, we describe our proposed defense method, which increases the cost of censorship attacks and eliminates persistent censorship. In Section IV, we present the results of the experiments showing the benefits of our proposed methods. Finally, we conclude this paper in Section V.

II. BACKGROUND

A. PROOF-OF-STAKE (PoS) CONSENSUS MECHANISM

As a first consensus mechanism, the PoW consensus mechanism employs a cryptographic hash function to ensure the integrity of blockchain content. As long as the cryptographic hash function is secure, the PoW consensus mechanism protects the content of the blockchain from the malicious validators. However, the PoW consensus mechanism has many problems such as high latency and large electric consumption, owing to the large number of calculations required to obtain cryptographic solutions.

The concept of proof of stake has been introduced to address such issues. If a stakeholder has a large stake in cryptocurrency, the PoS consensus mechanism assumes that they would act honestly in order to protect their vested interests. Based on this assumption, the PoS consensus mechanism selects a group of validators whose stake is greater than a certain stake. For example, in Ethereum [14], [15], [16], a node with 320ETH or higher can be a validator. Following validator selection, each validator votes to determine which block will be attached to the blockchain. However, the PoS consensus mechanism has a problem in that the cost of communication between all validators is prohibitively expensive. Furthermore, a validator with only the bare minimum of requirements has no discernible effect on the outcome of the consensus process. Thus, many blockchain platforms, such as Cosmos and EOS use a delegate PoS (DPoS) consensus mechanism, which limits the number of validators.

B. DELEGATE PROOF-OF-STAKE CONSENSUS MECHANISM

In the DPoS consensus mechanism, the number of validators is fixed to reduce the cost of communication between the validators. The number of validators, for example, is limited to 100 in the Cosmos blockchain network, which employs the Tendermint consensus algorithm. As another example, in the EOS blockchain network, this limited number of validators is only 21. Thus, a node that wants to be selected as a validator should have a large stake. To prevent a node from being unable to participate in the consensus process due to a lack of stake, the DPoS consensus mechanism allows them to participate in the consensus process by delegating their stake to their representative. When a validator to which nodes delegate their stake rewards for block verification, the delegators receive the rewards together. However, when the validator is penalized for malicious behavior, the delegators are also penalized. A node with a small stake in the DPoS consensus mechanism, like the PoS consensus mechanism, has no significant effect on the outcome of the consensus process. Note that, the goal of the DPoS consensus mechanism is to reduce communication costs between nodes by indirectly participating in the consensus process.

In a blockchain network applying the DPoS consensus mechanism, an administrator should carefully select an algorithm that decides how to finalize blocks reliably with a small number of validators. In addition, they should consider that the blockchain network works in an asynchronous environment, and there can be consensus inconsistency due to unpredictable communication failures. Therefore, most blockchain networks that apply the DPoS consensus mechanism, such as Cosmos and EOS, use the practical Byzantine fault tolerance (pBFT) algorithm.

C. PRACTICAL BYZANTINE FAULT TOLERANCE

The pBFT algorithm is a consensus algorithm which ensures the consistency of the consensus results among validators in asynchronous communication environment.

In Castro et al.’s original pBFT algorithm [7], a client (non-validator) node requests transaction processing to replica (validator) nodes and gets reply from replica nodes. To verify the requested transaction, the replica nodes perform consensus between request from a client and reply to a client. The consensus process consists of three phase: (1) Pre-prepare; (2) Prepare; and (3) Commit.

In the pre-prepare phase, a replica node that receives the request message for transaction processing verifies the format and content of the request message. The replica node then broadcasts the request (pre-prepare) message to all other replica nodes. Every replica node who receives the pre-prepare message begins the prepare phase. In the prepare phase, each replica node verifies the pre-prepare message. If this message is valid, the replica node broadcasts a prepare message containing its own verification result and the content of the request message to all other nodes. Otherwise, it does not do anything. After receiving prepare messages with same content from more than 2/3 of replica nodes, each node begins the commit phase. During the commit phase, each node processes the content of the request message and broadcasts the commit message by packaging the result. When receiving commit messages from more than 2/3 of replica nodes with same content, each node sends a reply message to the client node. After receiving the reply message from more than 1/3 of the replica nodes, the client node determines that the request has been correctly processed, and the consensus process is terminated.

Castro et al. also demonstrated that the network works correctly even when there are 3f + 1 nodes in the network and a maximum of f faulty nodes. Furthermore, they demonstrated that more than 2f + 1 nodes were required to manipulate the content of the request from the client. Thus, because the influence of each node is proportional to its stake in the DPoS consensus mechanism, the consent of nodes with a stake of 2/3(2f + 1/3f + 1) or more is required to finalize block safety in the blockchain network applying the pBFT-based DPoS consensus mechanism.
D. COUNTERMEASURES AGAINST A CENSORSHIP ATTACK
Since only a small number of nodes can be selected as validators that directly participate in the consensus process, the DPoS consensus mechanism has a limitation that malicious cartels can be easily formed [17]. As a result, these malicious cartels can perform group attacks, such as block manipulation and censorship attacks in the DPoS consensus mechanisms. In particular, when compared to block manipulation in the pBFT-based DPoS consensus mechanism, a censorship attack that paralyzes the blockchain network by interfering with the consensus process can be carried out by a small number of malicious cartels. This is because a malicious cartel only needs to own more than 1/3 of the total stake, which does not meet the BFT majority condition.

To prevent the censorship attack, many blockchain platforms and researchers have proposed various solutions. Cosmos and Ethereum, which are representative blockchain platforms that use PoS and DPoS consensus mechanisms, respectively, employ the method of bonding and slashing stakes [18]. Within their blockchain networks, nodes cannot withdraw but bond their stake for a certain period after participating in the consensus process. If a malicious behavior or attack attempt is discovered during the consensus process when they have participated, their stake in the bond is cut(slashed). However, this procedure is only a manual solution against censorship attacks. As mentioned previously, in order to prevent a censorship attack in advance, the cost of the censorship attack must be increased along with the disruption of the formation of the malicious cartel. As an example, Leonardos et al. proposed a weighted voting method using the voting history of validators [19]. In Leonardos et al.’s method, validators are selected proportionally to their stake using a pseudo-random mechanism at each time slot, and each validator has its own voting weight. After determining the voting weight based on the validator’s empirical probability of voting correctly, all weights are assigned to a new block. To maximize the motivation to behave honestly, they also introduced the concept of expected collective welfare and create a penalty rule for invalid voting using the concept. Consequently, the penalty for invalid voting makes the blockchain consensus process robust against censorship attacks. However, Leonardos et al.’s method can only increase the cost of the censorship attack but does not disrupt the formation of the malicious cartel. As a result, if a malicious cartel with sufficient stakes launches a censorship attack, the blockchain network will not be able to stop until manual intervention is taken.

III. PROPOSED METHOD
A. OVERVIEW
Since the existing blockchain consensus mechanism does not change the membership of the validator group until an abnormal or unqualified validator is removed manually, a censorship attack can be persistent when it occurs once. Furthermore, identifying malicious censoring behavior is difficult because the block verification process must be reviewed by humans. Thus, we focused on distinguishing reasonable censorship behavior automatically rather than finding malicious censoring behavior using manpower.

The goal of the proposed defense method is to make the blockchain network more tolerant of censorship attacks by penalizing validators who do not vote rationally, as well as to make the network recover automatically from persistent censorship attacks. To achieve this goal, we propose a new consensus architecture and credit-scoring function for the validator’s behavior.

In our proposed consensus architecture, we introduce the concept of hierarchical consensus by dividing the existing validator group into sub-validators and main-validators. Main-validators perform the same tasks as the validators in the existing blockchain network. If the result of the consensus process between the main-validators satisfies the majority condition of BFT(2/3 or more of the total stake), a block is normally attached to the blockchain. If not, sub-validators can disagree with the result of the consensus process from the main-validators and attach normal-but-censored blocks to the blockchain. Consequently, each validator group forms a competitive landscape in the proposed hierarchical consensus architecture.

We also propose a behavior-based credit-scoring function. When our credit-scoring function operates, the credit score of each validator changes according to its behavior at every consensus process. If the credit score of the sub-validator is higher than the credit score of the main-validator with the lowest credit score, they are promoted to the main-validator, and the existing main-validator is demoted to the sub-validator. Finally, the validators cannot remain in a group for an extended period of time, so the formation of a malicious cartel is disrupted. Even if a malicious cartel forms outside the blockchain network and launches a censorship attack, the attack cannot be launched indefinitely because the roles of malicious validators shift with each consensus process.

B. HIERARCHICAL CONSENSUS ARCHITECTURE
In this section, we describe the hierarchical consensus architecture, which forms a competitive landscape among validators. In addition, we theoretically prove that the proposed consensus architecture incurs a higher cost of censorship attacks than the existing consensus architecture.

1) COMPETITIVE CONSENSUS PROCESS
In the existing BFT-based PoS consensus mechanism, the censorship condition can be expressed as:

\[
\frac{S_a}{S} < \frac{2}{3},
\]

where \(S\) is the total stake of validators and \(S_a\) is the stake of the validators who agree to attach the proposed block to the main-chain. As can be seen from Equation 1, in the existing blockchain network, the consensus architecture has a simple structure in which only validators vote for and against each other. Therefore, until the malicious validator is
FIGURE 1. The structural difference between (a) the existing network’s consensus architecture and (b) the proposed hierarchical consensus architecture.

manually ejected from the validator group, there is no change in the members of the validator group, as shown in Figure 1. Because a validator group is made up of fixed validators in such a network, a malicious cartel among the malicious validators is easily formed. As mentioned in Section I, a malicious cartel can be formed using only five validators in the Cosmos network. Such a malicious cartel can cause persistent censorship attacks unless the members of the validator group change.

To address this limitation, it is necessary to add a new validator group that can observe and block the malicious behavior of existing validators. Therefore, we divided the validators into two groups, as shown in Figure 1b, that is, main-validators and sub-validators. The main-validators verify and vote on the proposed block, similar to the validators in the existing blockchain network. If the outcome of their vote meets the BFT majority condition, the consensus process proceeds normally. If not, sub-validators judge about the unsatisfied condition by voting. Here, since main-validators have more influence than sub-validators during the consensus process, they can obtain a major reward compared to sub-validators. As a result, they should actively participate in the consensus process to continue to belong to the main-validator group, and a competitive landscape is formed between the two validator groups. In order to realize our hierarchical consensus architecture in the environment of the existing network, we define three properties:

1) When the voting results of sub-validators affect the consensus results, the sub-validators’ voting results should follow the BFT majority condition.

2) The voting results of the sub-validators only determine the behavior of main-validators that voted for censorship.

3) The sub-validators vote when main-validators fail to create a new block.

Since the sub-validator network is an asynchronous distributed network, it can suffer from Byzantine fault problem. Property 1 prevents the inconsistency of the consensus between the sub-validators in such an environment. Thus, consensus function $C(B)$ can be defined as follows:

$$C(B) = \frac{S_m}{S_m} + \left( \frac{1}{3} - \frac{S_s}{S_v} \right),$$

where $B$ is a new block that is the target of consensus, $S_m$ is the total stake of the main-validators, $S_m$ is the total stake of the main-validators that accept block $B$, $S_s$ is the total stake of the sub-validators, and $S_s^r$ is the total stake of the sub-validators that reject block $B$. In Equation 2, the term $\frac{1}{3}$ indicates BFT majority condition of the sub-validators. This implies that sub-validators can participate in the consensus process alongside main-validators without encountering the Byzantine fault problem. However, because sub-validators have the same voting power as main-validators in such an equation, they can cause side effects such as censorship attacks on their own.

Property 2 is designed to prevent censorship attacks by a malicious cartel of only sub-validators. We assume that the decision criterion for block finalization in the blockchain network to which Equation 2 is applied is 2/3. This assumption implies that the proposed consensus architecture follows the
principle of the bft-based DPoS consensus mechanism. Under this assumption, if all the main-validators agree to block finalization and a malicious cartel satisfying the majority condition of bft is formed in the sub-validator group, the result of the consensus function \( C(B) \) is lower than the assumed decision criteria. In other words, the block may not be finalized by the malicious cartel of the sub-validator group.

To avoid such a situation, it is necessary to limit the influence of sub-validators during the consensus process. Weighted voting between two validator groups is one of the simplest methods. However, in the BFT consensus scenario, since the distribution of stakes between main-validators and sub-validators varies significantly, it is impossible to find a general weight value for every stake distribution.

Instead, we designed weights to prevent malicious censorship behavior of the main-validators. If the censorship behavior of the main-validators is justified, the bft majority condition for the block finalization of the sub-validators will not be satisfied. However, if the censorship behavior of the main-validators is malicious, the bft majority condition of the sub-validators will be satisfied (more than 2/3), which prevents the censorship behavior of the main-validators. As a result, the influence of the sub-validators varies with the degree of rejection of the main-validators, and the Equation 2 is modified as follows:

\[
C(B) = \frac{S_m}{S_m} + \left( \frac{1}{3} - \frac{S_s}{S_s} \right) \frac{S_m}{S_m},
\]

where \( S_m \) is the stake of the main-validators rejecting block B. In Equation 3, the influence of the sub-validators depends on the degree of rejection of the main-validators on block B. In addition, for the sub-validators to have the same influence as the main-validators, all main-validators should reject block finalization.

Although Property 2 reduces the influence of sub-validators, it is possible that the block is not finalized due to a malicious cartel in the sub-validator group. For example, even though the main-validators with a 70% stake agree to block finalization, censorship occurs if the malicious cartel of the sub-validator group rejects it by 47% or more. To prevent this situation, Equation 3 should operate only when the result of the consensus process between the main-validators does not satisfy the BFT majority condition. Note that, the sub-validators only judge the failure of the consensus process between the main-validators and do not agree with the main-validator. Property 3 is designed to limit the role of the sub-validator, and the modified consensus function \( C(B) \) to which Property 3 is applied can finally be expressed as follows:

\[
C(B) = \begin{cases} 
\frac{S_m}{S_m}, & \text{if } \frac{S_m}{S_m} > \frac{2}{3} \\
\frac{S_m}{S_m} + \left( \frac{1}{3} - \frac{S_s}{S_s} \right) \frac{S_m}{S_m}, & \text{otherwise} 
\end{cases}
\]  

In Equation 4, the consensus function returns the consensus result of main-validators when the BFT majority condition of the main-validators is satisfied. Otherwise, the consensus function returns the consensus results of both the main-validators and sub-validators. As a result, regardless of the malicious cartel size of the sub-validator group, a censorship attack cannot be performed if the voting result of the main-validators is 2/3 or more, which is the BFT majority condition and decision criterion.

2) THEORETICAL ATTACK COST ANALYSIS

In this subsection, we show that the consensus process in the proposed hierarchical architecture is more secure than the existing consensus process by analyzing the censorship attack cost. To demonstrate a concrete mathematical proof of security, we define the attack cost on the proposed hierarchical consensus network as follows:

Definition 1 (Censorship Attack Cost on Proposed Architecture): Let \( S^m, S^s \) be the total stake of the main-validators and sub-validators, with \( S = S_a + S_r \). The cost \( C \) required to censor a block is

\[
C > \frac{S_m S^s}{2S^s + 3S_r^s} + S_r^s.
\]

Proof: According to Equation 4, the condition for block censorship is

\[
S_m^r > \frac{S_m}{3} (\frac{3S^s}{2S^s + 3S_r^s}).
\]

If the left-hand side of this equation is summarized as the stake \( S_m^r \) of the main-validators who reject block creation, it can be expressed as

\[
S_m^r > \frac{S_m}{3} (\frac{3S^s}{2S^s + 3S_r^s}),
\]

with \( S_m^r + S_m = S_m \). Then, we add \( S_s^r \) to each side of the above equation, since the malicious validators can have a stake in both the main-validator group and the sub-validator group. Consequently, the final cost required to censor the block is

\[
C > \frac{S_m S^s}{2S^s + 3S_r^s} + S_r^s,
\]

where the attack cost \( C \) is the same as \( S_m^r + S_s^r \).

Given the attack cost required by our consensus scheme in Equation 5, we can compare the cost of a censorship attack between the existing consensus process and the proposed consensus process.

Theorem 1 (Comparison of Attack Cost): Let \( C' \) be the cost of a censorship attack required by the existing network. If \( S_m^r < 1/3, S_s^r < 1/3, \) and \( S_m^r > 3S_s^r \), then

\[
\min(C) > \min(C')
\]

Proof: Assuming that the number of validators in both networks is the same, \( C' \) which is the sum of \( S_m^r + S_s^r \) that can be expressed as

\[
C' = S_m^r + S_s^r > \frac{1}{3} (S_m^r + S_s^r)
\]

Let \( \text{Diff}_{C, C'} \) be the difference between the minimum attack costs of the two networks, given by

\[
\text{Diff}_{C, C'} = \min(C) - \min(C') = \frac{S_m S^s}{2S^s + 3S_r^s} + S_r^s - \frac{1}{3} (S_m^r + S_s^r)
\]
The consensus process in both our consensus architecture and the existing consensus architecture works only when the majority condition of the BFT between the main-validators is not satisfied. Therefore, we did not consider the difference in the attack cost when \(3S^a_s < S^a\) and replaced \(S^a\) with \(aS^a(0 < a < \frac{1}{3})\). Consequently, the final \(\text{Diff}_{C,C'}\) is arranged as follows.

\[
\begin{align*}
\text{Diff}_{C,C'} &= \frac{S^m S^s}{(2 + 3a)S^s} + aS^s - \frac{1}{3}(S^m + S^s) \\
&= \frac{1}{3}\left[\frac{3S^m S^s}{(2 + 3a)S^s} + 3aS^s - S^m - S^s\right] \\
&= \frac{1}{3}\left[\frac{3S^m S^s}{(2 + 3a)S^s} - (2 + 3a)S^m S^s\right] - (1 - 3a)S^s \\
&= \frac{1}{3}\left[\frac{(1 - 3a)S^m}{(2 + 3a)S^s} - (1 - 3a)S^s\right] \\
&= \frac{1}{3}\left[\frac{1 - 3a}{2 + 3a}\right]\left[\frac{S^m}{S^s} - (2 + 3a)\right]S^s \\
\propto S^m - (2 + 3a)S^s
\end{align*}
\]

Thus, we conclude that \(S^m > 3S^s\) in order for \(\text{Diff}_{C,C'} = \min(C) - \min(C') > 0\).

To demonstrate the applicability of the proposed consensus process used in our hierarchical consensus architecture, we used the distribution of validator stakes in the Cosmos blockchain network, which has 100 validators. Then, we assigned the upper half of the validators the role of the main-validator and the rest of the validators the role of the sub-validator in changing the consensus structure of this network into our hierarchical consensus structure. Figure 2 shows the distribution of the role of each validator and its stake. The ratio of the sum of the stakes of the main-validators to the sum of the stakes of the sub-validators is 7.2:1, which is greater than the 3:1 required by Theorem 1 to satisfy Equation 6. Therefore, the proposed consensus process, along with our hierarchical consensus architecture, can operate out of the box on the Cosmos blockchain network immediately.

\section{C. Behavior-Based Credit Scoring Function}

In this section, we propose a credit-scoring function that dynamically changes the membership of a validator group for every consensus process. First, we explain how the proposed credit-scoring function works and what each subfunction does. In addition, we show ways to apply our credit-scoring function to a practical network by adding a new phase to the pBFT algorithm.

1) \textit{Credit Scoring Function}

According to our hierarchical consensus architecture, it becomes possible to assign different rewards to each validator group. Therefore, sub-validators who receive relatively low rewards will participate more actively in the consensus process to become the main-validators. However, if all the main-validators behave correctly, not even one main-validator will be penalized and no sub-validator will become the main-validator. Consequently, the main-validators can form a large malicious cartel, which can maliciously manipulate blockchain networks.

To prevent such a situation, we must ensure that the validators do not stay in one group for a long time. Therefore, we designed a credit-scoring function that calculated the credit score according to the behavior of the validators and assigned groups according to the credit score of the validators in each consensus process. After the consensus process is completed and all of the validators’ credit score is calculated, a validator whose credit score is higher than half of all validators can become the main-validator. Here, our credit-scoring function can be expressed into:

\[
C^h_k = C^{h-1}_k + b_k R(r_k, s_k) + (1 + H(a_k, b_k))((b_k - 1)P(r_k, s_k) - r_k D_t(k)),
\]

where \(C^h_k\) represents the credit score of \(k\) node at \(h\) block height, which is affected by the credit score calculated in the previous block. This credit-scoring function requires five parameters: (1) \(b\), which represents the node’s behavior and is initialized to 0. If the node normally votes at (Prepare) or (Commit) phase, it is set to 1; (2) \(r\), which represents the role of the node. If the node is the main-validator, it is set to 1; otherwise, it is 0; (3) \(s\), which represents the node’s stake ratio. If the node is a main-validator\((r = 1)\), it is set as the amount of stake held in \(S^m\) discussed previously; (4) \(t\), which represents the number of times the node participated in consensus as a main-validator. When \(r\) is changed to 0 (sub-validator), it is initialized to 0, and when \(r\) is changed back to 1 (main-validator), the increment trigger of \(t\) is executed; and (5) \(a\), which represents the active time the node has been active as a consensus participant. It is reset to zero only when a consensus participant exits or is expelled. When the block finalization is complete, such parameters are calculated and saved to the score table, which is described in detail later.

In addition, the credit-scoring function requires four subfunctions: (1) \(R(r, s)\), which is a reward function only for honest nodes (\(b = 1\)). The reward increases in proportion...
Algorithm 1 Credit Scoring Function

Initialize:
\[ \text{Sig} \leftarrow \text{Signatures collected during pBFT} \]
\[ T \leftarrow \text{Total number of participants} \]
\[ h \leftarrow \text{Current block height} \]

1: Procedure **OnSuccess:**
2: \[ C^h = \{ T \} \]
3: \[ C^{h-1} = \text{scores}^{h-1} \]
4: \[ \text{for } k=1 \text{ to } T \text{ do} \]
5: \[ s, r, t, a, \text{addr} = \text{getRows}(k) \]
6: \[ b = \text{boolean}(\text{addr in Sig}) \]
7: \[ C^h_k \leftarrow C^{h-1}_k + \text{Increment}(b, s, t, r, a) \]
8: \[ \text{end for} \]
9: \[ \text{Finalization()} \]
10: \[ \text{updateCreditScoreTable}(C^h) \]

12: Procedure **OnFail:**
13: \[ C^{h-1} = \text{scores}^{h-1} \]
14: \[ \text{for } k=1 \text{ to } T/2 \text{ do} \]
15: \[ s, r, t, a, \_ = \text{getRows}(k) \]
16: \[ b = 0 \]
17: \[ C^{h-1}_k \leftarrow C^{h-1}_k - \text{Penalty}(b, s, a) \]
18: \[ \text{end for} \]
19: \[ \text{updateCreditScoreTable}(C^{h-1}) \]

Algorithm 2 Update Credit Score Table

Initialize:
\[ C^h \leftarrow \text{Score of all participants} \]
\[ T \leftarrow \text{Total number of participants} \]
\[ \text{Rows} \leftarrow \text{Rows of credit score table} \]

1: Procedure **updateCreditScoreTable:**
2: \[ \text{for } k=1 \text{ to } T \text{ do} \]
3: \[ \text{Rows}^h_k \text{-score} = C^h_k \]
4: \[ \text{end for} \]
5: \[ \text{Sort(Rows)} \]
6: \[ \text{for } k=1 \text{ to } T/2 \text{ do} \]
7: \[ sr, r, t, a, \text{addr} = \text{get(Rows}^\text{param}_k) \]
8: \[ \text{Rows}^h_k \text{-param} = \{sr, 1, t + 1, a + 1, \text{addr}\} \]
9: \[ \text{end for} \]
10: \[ \text{for } k=T/2 \text{ to } T \text{ do} \]
11: \[ sr, r, t, a, \text{addr} = \text{get(Rows}^\text{param}_k) \]
12: \[ \text{Rows}^h_k \text{-param} = \{sr, 0, 0, a + 1, \text{addr}\} \]
13: \[ \text{end for} \]
14: \[ \text{Rows}^{h+1} = \{\text{Rows}\} \]
15: \[ \text{WaitForNextBlock()} \]

where \( Err \) is the error penalty, which grows in proportion to stake(s) and historical information.

The overall process of the proposed credit-scoring function is depicted in Algorithm 1. If the result of our proposed consensus function \( C(B) \) in Equation 4 is greater than decision criterion \( 2/3 \), the OnSuccess procedure operates (line 1). Otherwise, the OnFail procedure will operate (line 12). When the OnSuccess procedure operates, each validator initializes a new list to store the credit scores of other validators and loads the list of credit scores saved from the previous consensus (lines 2 to 3). Then, each validator obtains the address, stake ratio, role, validating time, and active time from the credit score table for all the validators (lines 4 to 5). Subsequently, each validator checks whether the address exists in the signed votes collected during the voting process for each called node (line 6) and calculates the current credit score using Equation 7 (line 7). Finally, the verified block is finalized, and the credit score table is updated using the current credit score (lines 9 to 10). On the other hand, when the OnFail procedure operates, the block height \( h-1 \) is not updated (lines 13 and 19), and the censorship penalty function works for all main-validators according to Equation 8 (lines 14 to 18).

The operational process for updating the credit score table is presented in Algorithm 2. First, the credit scores calculated using Algorithm 1 are assigned to the validators (lines 2 to 4). Then, the Rows representing the validators’ information are sorted in ascending order based on the credit score (line 5). Afterward, when the consensus participants are \( T \) nodes, the role of the main-validator of the next block is assigned to the top \( T/2 \) (line 6), and the role of the sub-validator of the next block is assigned to the remaining \( T/2 \) (line 10). Then, in the case of the main-validator, the number of...

to the stake(s) and can be paid differently depending on the role(r). (2) \( H(a, b) \), which is a history function that refers to the historical information of the node’s behavior. The output of this function increases when the node performs dishonest behaviors such as duplicate signing or is absent during the consensus. As a result, the dishonest nodes receive a larger penalty \( P, D \) than the normal nodes. (3) \( P(r, s) \), which is a penalty function that works for malicious nodes \( b = 0 \). As with the reward function, it is affected by the stake and role of the node; and (4) \( D(t) \), which is a decentralization function only applied to the main-validators \( r = 1 \). This function is used to prevent the formation of malicious cartels and gives main-validators a penalty similar to the number of the consensus times \( t \).

The proposed credit-scoring function only works when the pBFT-based consensus process is normally completed. In other words, when the pBFT-based consensus process is not finished due to a censorship attack, the credit-scoring function will not work. To address this, the blockchain network should be able to recover from attacks by automatically changing the validator group. As a result, we concluded that when the consensus process is delayed for an extended period of time, all main-validators bear joint responsibility for the censorship attack. The error penalty applied to the main-validators for incomplete consensus is as follows:

\[ C^h_k = C^{h-1}_k - r_k(1 + H(a_k, b_k))Err(s_k), \]
Algorithm 3 Overall Process of Update Signature Phase

Initialize:

\[ \text{Sig} \leftarrow \text{Signatures collected during pBFT} \]

1: \( \text{matchPoint} = 0 \)
2: for each \( s \in \text{Sig} \) do
3: \( \text{Addr}_{\text{auth}}.\text{add}(s.\text{Addr}) \)
4: \( \text{Sort(Addr}_{\text{auth}}) \)
5: End for
6: Async for each time interval do
7: if \( \text{matchPoint} < 2/3 \) then
8: \( \text{Addr}_{\text{others}} = \text{Addr}_{\text{auth}}.\text{remove(Addr}_{\text{own}}) \)
9: \( \text{msg.header} = \text{Comp.request} \)
10: \( \text{msg.content} = H(S(\text{Addr}_{\text{others}})) \)
11: Broadcast(msg)
12: else
13: Break
14: end if
15: end for
16: Async while \( \text{matchPoint} < 2/3 \) do
17: if get(msg.header) then
18: Case \( \text{Comp.request} \):
19: GetRequestComp()
20: Case \( \text{Comp.reply} \):
21: GetReplyComp()
22: Case \( \text{Evid.request} \):
23: GetRequestEvid()
24: Case \( \text{Evid.reply} \):
25: GetReplyEvid()
26: end if
27: end while
28: return OnSuccess()

2) PRACTICAL APPLICATION SOLUTION

The members of each validator group can be dynamically changed in each block creation by applying the credit-scoring function to all validators equally. In practice, however, the nodes may not obtain the same score table containing the validators’ roles at the same block height. This is because some nodes with network latency cannot forward their votes to all other nodes, owing to an asynchronous network. In the worst case, this can lead to chain branching of the blockchain. To solve this problem, we added the update signature phase after the commit phase to make the network weakly synchronous, as shown in Figure 3.

The update signature phase uses two processes to synchronize the signatures of all nodes during pBFT phases: Compare and Evidence. Here, each process operates based on request messages and reply messages. Algorithm 3 describes the overall operation of the update signature phase.

First, each node initializes \( \text{matchPoint} \) to zero (line 1). Here, \( \text{matchPoint} \) increases when a node finds another node with the same information as itself. Then, the node stores the addresses in the signatures received during the three phases of pBFT as a one-dimensional sorted list (lines 2 to 5). When a sorted list is created, the node executes two asynchronous operations. In the first operation, the node broadcasts a message containing the address list at regular intervals until \( \text{matchPoint} \) is more than 2/3 (lines 6 to 15). At this time, the node removes its address from the list before propagation. This can prevent malicious nodes from adding their addresses to the list while also allowing honest nodes to be authenticated by other nodes. The node then sends the list as a message. The header of the message is \( \text{Comp.request} \) to request the comparison of equality, and the whole message is \( \{\text{Comp.request}, D(S(\text{Addr}_{\text{others}}))\}_{\text{Auth}} \), where \( D \) is a hash function, \( S \) is serialization, and \( \text{Auth} \) is the address of sender \( i \) for verification. In the second operation, the node processes the received message until \( \text{matchPoint} \) is 2/3 (lines 16 to 27). The processing method according to each message is described in detail in Algorithms 4 to 6. Finally, the proposed phase ends when the match point reaches 2/3, and the onSuccess procedure in Algorithm 1 is called (line 28).

Algorithm 4 describes the operation for processing the \( \text{Comp.request} \) message mentioned in lines 18 to 19 of Algorithm 3. The node first removes the sender’s address from its address list (line 2). If the hash of the list from which the sender’s address is removed and the value contained in the message is the same, \( \text{matchPoint} \) is increased by the sender’s stake ratio (lines 3 to 4). Otherwise, the node removes its address from the list it owns and unicasts it as a message to the sender (lines 5 to 10). The header of the message is \( \text{Comp.reply} \), and the entire message is \( \{\text{Comp.reply}, D(S(\text{Addr}_{\text{others}})), S(\text{Addr}_{\text{others}})\}_{\text{Auth}} \).

Algorithm 5 describes the operation for processing the \( \text{Comp.reply} \) message mentioned in lines 20 to 21 of Algorithm 3. First, the node initializes two empty lists (line 2), then it lists the serialized addresses in the message and modifies them for comparison (lines 3 to 4). The node then finds the address that it does not authenticate in the

![Figure 3](image-url)
pBFT phase and adds it to the list (lines 5 to 9). If the list contains more than one address, it is sent as a message (lines 10 to 15). At this time, the header of the message is \( Evid_{\text{request}} \) to request evidence, and the entire message is \( \langle Evid_{\text{request}}, D(S(Addr_{\text{unknown}})), S(Addr_{\text{unknown}}) \rangle_{a,b} \). However, if the sender is unaware of any consensus participants, the node lists their addresses (lines 16 to 20). Subsequently, if there is more than one address in this list (line 21), the node adds the signature corresponding to the address to the new list (lines 22 to 27). Finally, this list is sent to the sender in reply form, and the entire message is \( \langle Evid_{\text{reply}}, D(S(Sig_{\text{eid}})), S(Sig_{\text{eid}}) \rangle_{a,b} \).

Algorithm 6 describes the process of the two procedures in lines 22 to 25 of Algorithm 3. The first procedure occurs when a node receives \( Evid_{\text{request}} \) message. At this point, the node adds signatures to the messages corresponding to the addresses on the list and sends them to the \( Evid_{\text{reply}} \) message, as described in lines 21 to 32 of Algorithm 5 (lines 1 to 12). The second procedure occurs when a node receives a \( Evid_{\text{reply}} \) message. Initially, the node lists the serialized signatures contained in the message (line 15) and adds them to its signature list (lines 16 to 18). Finally, the node initializes \( \text{matchPoint} \) because the list of signatures it holds has changed (line 19).

D. OPERATIONAL EXAMPLE

This section describes an example of the overall operation in which censorship occurs in both networks to help understand the difference between the network with the existing bft-based consensus mechanism and the network that uses the proposed methods.

Before the description, we assumed the basic sub-functions of the credit-scoring function, which is as follows:

\[
R(r, s) = P(r, s_k) = 0.2s \\
D(t) = 0.03t \\
Err(a, b, r, s) = r(1 + H(a, b))(0.5s)
\]

where each function has been configured to have a linear relationship with the parameters. We then set the initial credit score to the validator stake ratio. Also, in order to prevent continuous non-participation in consensus, we set the standard score for the expelling of the consensus process to -0.05. Finally, we made the validator composition and stake in both networks the same and set N1 and N3 as a malicious cartel with more than 33% stake, as shown in Figure 4.

Figure 4a shows that a censorship attack occurs by a malicious cartel(N1,N3) in the existing network. When the local time is 100, the stake of the malicious cartel occupies 40% of the total; therefore the bft-majority condition is not satisfied.
does not participate in the consensus, and Rej-main-validator, Val which consists of the same malicious cartel as before. Where and no blocks will be finalized until manual management the future. As a result, the existing network will be paralyzed, same validator group members, this censorship is repeated in because of the existing simple consensus structure and the according to the existing consensus process. Furthermore, Finally, in step 3(local time : 102), the network is stabili- tion 9. As a result, N5 with the highest score was promoted of the malicious cartel, was demoted to the sub-validator. of time and is proportional to the stake of the validator when the consensus process does not end for a certain period error function (Err) is applied equally to only main-validators set according to the network’s purpose, as detailed below. The experimental results show that the blockchain network that uses our proposed method is not only resistant and tolerant to censorship but also recov- ers automatically in the occurrence of persistent censorship attacks.

A. EXPERIMENTAL SETUP

In the overall experiments, we used the Cosmos Validator List [11], extracted from a network utilizing tendermint where 100 validators go through a consensus process. We then define the basic subfunctions of the credit-scoring function, which are described below.

\[ R(r, s) = (1 - r) s \]
\[ P(r, s) = s \]
\[ D(t) = \alpha t \]
\[ \text{Err}(a, b, r, s) = \beta r(1 + H(a, b)) s \]
\[ H^0(a, b) = 0 \]
\[ H(a, b) = \frac{(a - 1)H(a - 1, b) + 1 - b}{a} \quad (10) \]

The reward function (R) is proportional to the stake and is applied only to sub-validators to prevent the divergence of credit scores generated from the main-validators with many stakes. However, we define the penalty function (P) to be equally proportional to the stake of all validators. The decentralization function (D) is proportional to the time spent in the main-validator group and the hyperparameter \( \alpha \), which can be set according to the network’s purpose, as detailed below. The error function (Err) is applied equally to only main-validators when the consensus process does not end for a certain period of time and is proportional to the stake of the validator and the censorship parameter \( \beta \) set manually. The history function (H) is defined as the number of non participation in consensus among the number of blocks generated after the node becomes a validator.

B. OPTIMIZATION OF CREDIT SCORING FUNCTION

To avoid the formation of a malicious cartel in a normally functioning network, the proposed credit-scoring

Algorithm 6 Reply Evidence

Initialize:

\[ \text{Sig} \leftarrow \text{Signatures collected during pBFT} \]
\[ \text{Length}_{addr}, \text{Length}_{sig} \leftarrow \text{Length of address, signature} \]

1: Procedure GetRequestEvidence:
2: \[ \text{Addr}_{get} = \text{list}(\text{split}(\text{msg.content}_2, \text{Length}_{addr})) \]
3: \[ \text{Sig}_{evid} = \{ \} \]
4: for each \( s \in \text{Sig} \) do
5: \[ \text{if } s.\text{Addr} \in \text{Addr}_{get} \text{ then} \]
6: \[ \text{Sig}_{evid}.\text{add}(s) \]
7: end if
8: end for
9: \[ \text{msg.header} = \text{Evid}_\text{reply} \]
10: \[ \text{msg.content}_1 = H(S(\text{Sig}_{evid})) \]
11: \[ \text{msg.content}_2 = S(\text{Sig}_{evid}) \]
12: \[ \text{Unicast}(\text{msg}) \]
13:

14: Procedure GetReplyEvidence:
15: \[ \text{Sig}_{get} = \text{list}(\text{split}(\text{msg.content}_2, \text{Length}_{sig})) \]
16: for each \( s \in \text{Sig}_{get} \) do
17: \[ \text{Sig}.\text{add}(s) \]
18: end for
19: \[ \text{matchPoint} = 0 \]
function should be optimized based on the consensus process participants. In particular, we tried to optimize a decentralization function, a sub-function of the credit-scoring function, by focusing on finding an appropriate hyperparameter \( \alpha \). Note that the purpose of the decentralization function is to give the main-validator a penalty to change roles, thereby reducing the bond between them. \( \alpha \) must be set up carefully, because several side effects may exist according to this value. Specifically, if it is set too high, the importance of the decentralized function increases, rendering the credit score unaffected by other functions. However, if \( \alpha \) is set too small, it may be impossible to prevent the formation of malicious cartels. Therefore, the network administrator should set \( \alpha \) by considering various factors such as the stake of the node and the effects of other functions. We focused on \( \alpha \) to provide a fair chance of block verification.
According to the basic proof of stake (PoS) rule, a node is given the right to generate blocks and obtain rewards equal to the difference in the amount of stake held. When this rule is applied to DPoS validators, the reward and influence on the block validation result are proportional to the difference in stake held. In our hierarchical architecture-based DPoS consensus mechanism, it can be defined that the difference in the number of block validations as the main-validators between two validators should be equal to the difference in stake. Therefore, we considered $\alpha$ to distribute the roles of validators fairly according to their stakes.

Figure 5 shows the ratio of the block validation times of a node with the maximum stake ($n_1$) and another node with the minimum stake ($n_{100}$) according to $\alpha$ when 10000 blocks are generated. If the alpha is too low ($\alpha < 0.00225$), $n_{100}$ has never been able to verify a block as the main-validator; therefore, the ratio of block validation times is infinite. On the other hand, if the alpha is too high, the proportion of the decentralized function increases, showing that the $n_{100}$ node frequently acts as a main-validator regardless of the amount of stake held. In the cosmos validator list used in this experiment, the two nodes ($n_1 = 13594581$ and $n_{100} = 144806$) show a stake ratio of approximately 93:1. As mentioned before, we decided to consider $\alpha$, which makes the stake proportional to the number of block verifications. Finally, we found $\alpha$ that best reflects a ratio close to 93:1 is 0.002352.

C. EXPERIMENTAL ANALYSIS

1) TOLERANCE FOR UNEXPECTED CENSORSHIP

The occurrence of censorship is caused not only by malicious cartels but also by various reasons such as network delays, errors, and offline block proposers. As a result, we manually caused nonresponse errors on some arbitrary nodes to demonstrate the proposed methods’ tolerance to censorship that occurs for unknown reasons.

Figure 6 shows the difference in the censorship rate between the existing network and the network that uses the proposed method, as the number of error-generating nodes increases when $\alpha, \beta$ are set to 0.002352, 1. In each network, the number of error nodes was increased from 10 to 60 for 10000 normal block generations. This is because the number of nodes at which an attack starts is 10 and the number of nodes where an attack occurs unconditionally is 60. Additionally, we randomly selected the error nodes for each block generation. In Figure 6, while the existing network shows a sharply increasing censorship rate depending on the error node, it can be seen that the network can be made highly tolerant of censorship by applying the proposed method. This is because censorship occurs in the existing network if validators do not satisfy the bft majority condition; however, censorship is frequently prevented by sub-validators in the proposed network.

2) CENSORSHIP ATTACK RESISTANCE

The blockchain network where the proposed method is used is also highly resistant to continuous censorship attacks via the formation of a malicious cartel, as well as censorship caused by node errors. To demonstrate this, we present the success rate of censorship attacks as a function of the stake in carrying out the attack, as determined by the formation of a malicious cartel. We increased the number of malicious validators for the top 50 validators by the stake in this experiment to demonstrate a dramatic attack effect caused by the formation of a malicious cartel with a large stake.

Figure 7 shows the empirical attack success rate, which is the number of censorships occurring in the total number of block proposals (=10,000) according to the distribution of the malicious validators’ stake in the blockchain networks. Each blockchain network consisted of an existing network and networks with $\beta$ values of 1, 5, and 10. In the existing network, the network could no longer generate blocks from 19 malicious validators whose stakes exceeded 33% of the
3) AUTOMATIC RECOVERY AGAINST CENSORSHIP ATTACKS

Our proposed method enables the blockchain network to automatically recover from a persistent censorship attack, because the proposed method changes the members of the validator group by giving a solidarity penalty to all main-validators when the network does not complete the consensus process for a certain period of time. Thus, we demonstrate the resilience to persistent censorship attacks over time in a blockchain network which applies the proposed method. In this experiment, we increased the number of malicious validators for the top 50 validators in the same manner as in section IV-C2.

Figure 8 shows the number of block finalizations over time when a censorship attack occurs in the existing network and in the network that applies our proposed method. In the experiment, we assumed that the normal block generation process took 2 seconds from creation to finalization and that the consensus process took 10 seconds. We ignored the banishment from the consensus participants and set the first 100 seconds as the time for validators outside the blockchain network to form a malicious cartel. In both networks, when there is no attack on the network, 500 blocks can be proposed and finalized for 1000 seconds. However, from 100 seconds onwards, if a malicious cartel with a stake of more than 33% in the existing network performs an attack, it will be seen that the blockchain network is paralyzed continuously until manual action is taken. In contrast, the network that applies the proposed method imposes a penalty for the censorship attack occurring in the current block and allows the next block to be normally generated by changing the role of the validator. In the case of an attack by a malicious cartel of 35 validators with a 40% stake, approximately 200 blocks of 500 block proposals were created. In particular, even when all 50 of the network’s first main-validators formed a malicious cartel that held a 63% stake, approximately 100 blocks were generated out of 500 block proposals. Finally, we conclude that the proposed method allows for forcing block generation even in the presence of continuous network censorship.

V. CONCLUSION

In this paper, we described the risk of the formation of a malicious cartel in a blockchain network by applying a DPoS consensus mechanism. In particular, it was also introduced that when the DPoS consensus mechanism operates based on the pBFT algorithm, persistent censorship attacks can be caused by a malicious cartel that holds more than 1/3 of the total stake. As the cause of such attacks, we pointed out the easy formation of a malicious cartel from a fixed validator group and a simple consensus structure.

Therefore, we proposed a new defense method against persistent censorship attacks. We first introduced the concept of main-validators and sub-validators, and engaged them in the consensus process, thus creating a consensus hierarchical architecture. Through this, we increased the cost of censorship attacks in the blockchain network which applies the hierarchical architecture. Subsequently, we propose a behavior-based credit-scoring function that enables the members of each group to be replaced according to the behavior of the validator at every block generation. Thus, our behavior-based scoring method not only prevents the formation of a malicious cartel but also allows the network to automatically recover in the event of continuous censorship attacks by changing the validator group.
We conducted experiments using an actual validator list to demonstrate the practicality of the proposed method. Through experiments, we showed that the blockchain network which applies our proposed method is tolerant and resistant to censorship. In addition, unlike the existing network, which is paralyzed when a continuous censorship attack occurs, the network which applies the proposed method continues to create and finalize blocks by automatically changing the validator group membership.

REFERENCES

[1] S. Nakamoto. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[2] D. Tith, J.-S. Lee, H. Suzuki, W. M. A. B. Wijesundara, N. Taira, T. Obi, and N. Ohyama, “Application of blockchain to maintaining patient records in electronic health record for enhanced privacy, scalability, and availability,” Healthcare Informat. Res., vol. 26, no. 1, p. 3, 2020.
[3] ACS Solutions. (2017). Blockchain: For Secure Financial Transactions. [Online]. Available: https://acsicorp.com/blogs/blockchain-for-secure-financial-transactions/
[4] E. Buchman, J. Kwon, and Z. Milosevic, The latest gossip on BFT consensus, 2018, arXiv:1807.04938.
[5] D. Larimer. (2018). Eos.io Technical White Paper v2. [Online]. Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md
[6] J. Steem. (2018). Steem: An Incentivized, Blockchain-Based, Public Content Platform. [Online]. Available: https://steem.com/steem-whitepaper.pdf
[7] C. Miguel and L. Barbara, “Practical byzantine fault tolerance,” in Proc. 3rd Symp. Operating Syst. Design Implement (USENIX), vol. 99, 1999, p. 173.
[8] H. Hasanova, U.-J. Baek, M.-G. Shin, K. Cho, and M.-S. Kim, “A survey on blockchain cybersecurity vulnerabilities and possible countermeasures,” Int. J. Netw. Manage., vol. 29, no. 2, p. e2060, Mar. 2019.
[9] G. Zhang, F. Pan, M. Dang’ana, Y. Mao, S. Moteppali, S. Zhang, and H.-A. Jacobsen, “Reaching consensus in the byzantine empire: A comprehensive review of BFT consensus algorithms,” 2022, arXiv:2204.03181.
[10] J. Kwon and E. Buchman. Cosmos: A Network of Distributed Ledgers, Accessed: Mar. 5, 2021. [Online]. Available: https://v1.cosmos.network/resources/whitepaper
[11] Cosmos Validator List. Accessed: Apr. 18, 2021. [Online]. Available: https://cosmoscan.net/cosmos validators-stats
[12] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, “The honey badger of BFT protocols,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2016, pp. 31–42.
[13] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, “Prism: Deconstructing the blockchain to approach physical limits,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Nov. 2019, pp. 585–602.
[14] V. Buterin and V. Griffith, “Casper the friendly finality gadget,” 2017, arXiv:1710.09437.
[15] V. Buterin, D. Hernandez, T. Kamphofner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, “Combining GHOST and casper,” 2020, arXiv:2003.03052.
[16] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” Ethereum Project Yellow Paper, vol. 151, pp. 1–32, Apr. 2014.
[17] (2019). Cartels and Whales in DPoS Voting. [Online]. Available: https://medium.com/@CyberRepublic/cartels-and-whales-in-dpos-voting-d18e48ca534a
[18] V. Buterin, Serenity Design Rationale. Accessed: Jun. 4, 2021. [Online], Available: https://notes.ethereum.org/@vbuterin/serenity_design_rationale
[19] S. Leonardos, D. Rejsbergen, and G. Piloureas, “Weighted voting on the blockchain: Improving consensus in proof of stake protocols,” Int. J. Netw. Manage., vol. 30, no. 5, p. e2093, Sep. 2020.

JAE-SEOK KIM was born in Pusan, South Korea, in 1996. He received the B.E. degree from Pusan National University, Busan, South Korea, in 2014, where he is currently pursuing the Ph.D. degree in computer science and engineering. His research interests include security for artificial intelligence and blockchain consensus vulnerability.

JOON-HYUK OH received the B.E. and Ph.D. degrees from Pusan National University, Busan, Republic of Korea, in 2016 and 2022, respectively. He is currently an Associate Professor with the Division of Software, Yonsei University, Wonju-si, Republic of Korea. His research interests include artificial intelligence for security, security for artificial intelligence, adversarial examples, backdoor attacks on deep learning models, malware detection, and intrusion detection systems.

SEOK-HWAN CHOI received the B.E. and Ph.D. degrees from Pusan National University, Busan, Republic of Korea, in 2016 and 2022, respectively. He is currently an Associate Professor with the Division of Software, Yonsei University, Wonju-si, Republic of Korea. His research interests include artificial intelligence for security, security for artificial intelligence, adversarial examples, backdoor attacks on deep learning models, malware detection, and intrusion detection systems.

YOUNG-HO CHO (Member, IEEE) received the M.S. and Ph.D. degrees from the School of Electrical and Computer Engineering, Seoul National University, Seoul, South Korea, in 2004 and 2008, respectively. He was a Postdoctoral Scholar at Seoul National University, in 2008, and Pennsylvania State University, University Park, PA, USA, in 2009. From 2010 to 2012, he was a Senior Engineer with Samsung Electronics. From 2012 to 2014, he was an Assistant Professor with the Department of Convergence Security, Kyonggi University, Suwon, South Korea. He is currently a Professor with the School of Computer Science and Engineering, Pusan National University, Busan, South Korea. His research interests include privacy-preserving deep learning, adversarial examples, anomaly detection, deep packet inspection for high speed intrusion prevention, and the IoT security for realizing secure computer systems and networks. He has served as a TPC member and an editor for various international conferences and journals.

* * *