Received: 25 October 2019 | Revised: 25 March 2020 | Accepted: 6 May 2020

DOI: 10.4218/etrij.2019-0489

ORIGINAL ARTICLE

ETRI Journal WILEY

Algorithm based on Byzantine agreement among decentralized agents (BADA)

Jintae Oh | **Joonyoung Park** | **Youngchang Kim** | **Kiyoung Kim**

Artificial Intelligence Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea

Correspondence
Jintae Oh, Artificial Intelligence Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea.
Email: showme@etri.re.kr

Funding information
This research was supported by the Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government [No. 2018-0-0020, Development of High Confidence Information Trading Platform Based on blockchain (PON algorithm)].

Abstract

Distributed consensus requires the consent of more than half of the congress to produce irreversible results, and the performance of the consensus algorithm deteriorates with the increase in the number of nodes. This problem can be addressed by delegating the agreement to a few selected nodes. Since the selected nodes must comply with the Byzantine node ratio criteria required by the algorithm, the result selected by any decentralized node cannot be trusted. However, some trusted nodes monopolize the consensus node selection process, thereby breaking decentralization and causing a trilemma. Therefore, a consensus node selection algorithm is required that can construct a congress that can withstand Byzantine faults with the decentralized method. In this paper, an algorithm based on the Byzantine agreement among decentralized agents to facilitate agreement between decentralization nodes is proposed. It selects a group of random consensus nodes per block by applying the proposed proof of nonce algorithm. By controlling the percentage of Byzantine included in the selected nodes, it solves the trilemma when an arbitrary node selects the consensus nodes.

KEYWORDS

decentralization, distributed consensus algorithm, $O(N)$, proof of nonce

1 | INTRODUCTION

Recently, blockchain has been attracting attention as a technology that can realize a trust infrastructure in the 4th industrial revolution era and it is considered a field with high growth potential. However, because of the blockchain trilemma, the manner in which blockchain can be developed further is questionable. This trilemma implies that a trade-off needs to be found between three key aspects of blockchain: decentralization, scalability, and security. A blockchain system can address at most two of these trilemma points. Permissioned blockchains cannot resolve the trilemma, because they do not fully meet the decentralization criteria. Public blockchains guarantee security and decentralization; however, in their case, the scalability problem also needs to be resolved. This is called the scalability trilemma. The Bitcoin community has sought to solve this problem by implementing an additional protocol layer (called the Lightning Network) that can increase the number of transactions per second without degrading security or decentralization [1]. Developers of the Ethereum platform have attempted to mitigate the limitations inherent in the consensus mechanism by sharding (blockchain fragmentation) or using second-layer solutions, such as plasma [2].

Despite these efforts, public blockchains remain limited in terms of providing real-time services because of architectural limitations regarding the support of immediate finality [3]. In the case of Bitcoin, even a transaction stored

in a block cannot be completed within 1 hour because of the six-block confirmation required. Finality occurs as soon as a block is created, and the transactions contained therein become immutable. Real-time services cannot be implemented without finality. Recently, consensus algorithms have been developed to ensure finality and to optimize consensus costs by applying provable decentralization technologies [4–6]. However, these methods have disadvantages, such as weak decentralization, because of their inability to construct a new congress for each block, and high consensus complexity, because the minimum number of nodes required for Byzantine tolerance is significantly large. Therefore, an algorithm is required that can construct a congress that can withstand Byzantine faults with the decentralized method.

In this paper, we propose a new distributed consensus algorithm called Byzantine agreement among decentralized agents (BADA) that allows both decentralization and finality to be ensured, while it maintains the same level of security in public blockchains. The proposed algorithm can provide a decentralized consensus by fairly selecting the consensus congress and consensus quorum from among participating nodes and achieves block-level finality by exchanging $O(N)$ messages. The proposed algorithm can operate when the number of nodes participating in the agreement is larger than four, as compared to other algorithms that require at least several hundred nodes.

A proof of nonce (PoN) algorithm is proposed for decentralized consensus node selection. It describes the generation of a random value by PoN and the selection of a node, and it explains the method used to define a consensus congress and consensus quorum to tolerate Byzantine faults.

In addition, the proposed distributed consensus algorithm allows non-fixed nodes to exchange $O(N)$ messages through the following consensus steps: delegate request, prepare, and commit. At the delegate request step, a preparation block is created by selecting only the transaction agreed by $f + 1$ nodes or more among the responding $2f + 1$ consensus quorum among a $3f + 1$ consensus congress. The created preparation block is delivered to the $2f + 1$ consensus quorum at the prepare step, and finally, the consensus is completed using multisignatures from the consensus quorum at the commit step.

Section 2 introduces existing published studies on overcoming the problems and limitations of decentralization technologies and provides an analysis of the methods presented to solve these problems from the perspectives of structure and operation. Section 3 describes PoN applied to establish decentralization and to determine the consensus congress and consensus quorum, and Section 4 describes the consensus algorithm proposed in the present paper, the complexity of which is $O(N)$. Section 5 presents the results of the protocol simulation and proof of concept (PoC) implementation, and Section 6 outlines the main conclusions.

2 | RELATED WORK

Consensus algorithms can be categorized into competitive and non-competitive methods. A consensus algorithm can be chosen according to the purpose of the application field. Thus far, no consensus algorithm has been successfully commercialized because of technical limitations, such as the requirement that it simultaneously provide decentralization, scalability, and security.

The consensus algorithm constitutes a technology that is aimed to provide an agreed conclusion among unreliable nodes and basically is required to solve the Byzantine problem. However, the Byzantine problem is complex because of the traitor's ability to lie. Existing Byzantine fault tolerant (BFT) studies have shown that the Byzantine generals' problem can be solved with a node size of $2f + 1$ for f Byzantine nodes, assuming that the signatures can be used to detect changes in the message content of an honest general [7].

A BFT algorithm relies on two assumptions: detection of the original data corruption is possible through the signature of the data and the data should be delivered within a certain time (synchronous network). However, the actual blockchain technology is implemented as a peer-to-peer (P2P) network, which cannot guarantee that the data will be delivered within a certain time frame. To address this issue, the practical BFT (PBFT) algorithm is enhanced using a method to operate under the realistic partial synchronous network environment [8].

To solve the problem of delayed finalization, which is considered the most important limitation of the competitive consensus algorithm, various studies have been conducted to provide block-level finality by applying the PBFT concept to the competitive consensus method [9]. Recent research work has been focused on improving the PBFT algorithm to reduce the consensus cost in terms of the message exchange complexity from $O(N^2)$ to $O(N)$ [10,11] and the consensus congress size from $3f + 1$ to $2f + 1$ [12,13].

MinBFT is a well-known consensus algorithm that allows a reduction in the consensus congress size from $3f + 1$ to $2f + 1$. It can tolerate f Byzantine nodes, requiring only $2f + 1$ nodes, and it includes a unique identifier generated by the trusted execution environment (TEE) in a message so that the process of message exchange is verifiable. Consequently, occurrences of equivocation are avoided, and therefore, f Byzantine nodes can be tolerated when using only $2f + 1$ nodes. To generate a unique identifier, the following conditions must hold: the same identifier should not be assigned to two different messages, a message identifier must be monotonically incremented so that it cannot take a value smaller than that of the previous message identifier, and the message identifier must be updated sequentially.

Moreover, the FastBFT algorithm was introduced to reduce the consensus of the message exchange complexity

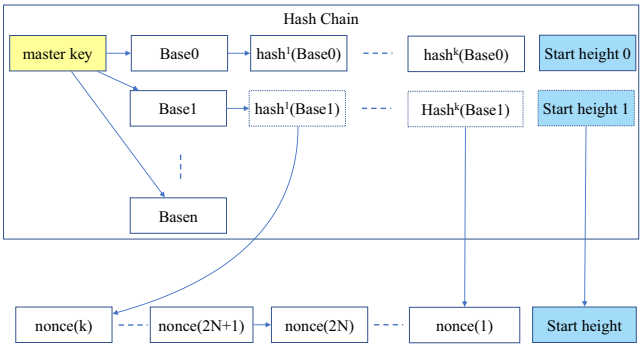

FIGURE 1 Proposed nonce chain

from $O(N^2)$ to $O(N)$. FastBFT increases the processing power through aggregation of messages to enable their delivery through the primary node, rather than allowing all the nodes to send and receive messages between each other directly.

However, both the above algorithms utilize TEE as the underlying hardware specification to prevent the equivocation of Byzantine nodes. Then, they reduce the total number of nodes required for a consensus and therefore the number of messages exchanged during a consensus.

Recently, two algorithms called Algorand and Zilliqa were proposed to provide block-level finality based on the decentralization concept. The decentralization concepts suggested by both algorithms are mathematically proven, implying that only a portion of the nodes are required to participate in a consensus process.

Algorand is a consensus algorithm that uses a verifiable random function (VRF) to address the issue of the number of nodes participating in a consensus. In Algorand, it is assumed that more than 80% of participating nodes are honest nodes. Under the Byzantine proportion of 20%, it creates a committee of 2000 nodes in Phase 1 and then sets up a committee of 2000 nodes for 10 out of 11 steps in Phase 2. In the last step in Phase 2, it makes a committee of 10 000 nodes to ensure a safety violation probability of $5e^{-9}$ [4]. The message complexity of Algorand is $O(cn)$, where c is the number of nodes that compose a committee using VRF in a 20% Byzantine agreement environment [14,15]. In the conducted tests, which included 50 000 nodes, the consensus time was not negligible as, on average, the completion of this process required 22 seconds. Therefore, various opinions exist about the applicability of this approach to the various applications that require real-time service.

The Zilliqa algorithm arbitrarily selects 800 nodes from a 25% Byzantine environment to proceed with the consensus [16]. A number of consensus algorithms, including Zilliqa, run the prepare phase of a consensus by the agreement of $2f + 1$ nodes. As the composition of a committee is based on proof of work (PoW), a new committee cannot be constructed per block, and therefore, this algorithm fails to provide decentralization. It also requires that two rounds of multisignatures be completed to agree on a block.

3 | DECENTRALIZATION BY PROOF OF NONCE

Many decentralized consensus algorithms used in blockchain are limited with respect to providing full decentralization. When a consensus node is randomly selected for decentralization, the following three conditions must be satisfied. First, each node must be able to perform eligibility verification only once per block height. Second, the eligibility of a node to participate in the agreement should not be predictable in advance. Finally, other nodes must be able to confirm that they will qualify for the fair agreement. In this paper, a new method for the decentralization of consensus nodes and a non-competitive agreement algorithm that has a $O(N)$ message exchange complexity is proposed.

3.1 | Participation in the pool with a nonce chain

In this study, given a nonce and the previous blockhead value, the general approach is to calculate an arbitrary value, compare it with the threshold value, randomly select a node from the participating pool, and then create a congress. A "congress" is defined as a group of nodes selected to participate in the consensus and a "quorum" as the minimum number of nodes required for voting within a congress. Nodes can participate in the pool by posting their nonce chain information. However, the participation entitlement of a node must be limited by providing an incentive mechanism to prevent a malicious node from registering multiple nonce chains.

For the process of selecting arbitrary nodes in the pool, we propose a verification method that uses a nonce chain and demonstrates its use. Then, we discuss the creation and use of the nonce chain.

3.1.1 | Creation and preparation of a nonce chain

To enable nodes to participate in an agreement, a hash chain generated as shown in Figure 1 is required. Then, we generate a seed value (base) to obtain one hash chain and hash the value continuously k times to generate a hash chain having a length of k [17]. As shown in Figure 1, the base value provided by the key chain generated by the master key and random secret value is used as a seed value. The seed value used by the node is a hash value with the master key of the node and the random value generated secretly by the node as

the input value. When one hash chain is consumed, to create the subsequent hash chain a base value is obtained by incrementing the random secret value by 1 and hashing with the master key. Then, the base value is hashed continuously to generate a hash chain. The generated data have the features of a hash chain, but in this paper, we refer to it as a nonce chain, because this information is used as a nonce to calculate the random value. Nodes are ready to use the nonce chain to participate in the pool when its last value (*nonce*(1)) and the *start_height* of the nonce chain have been revealed.

3.2 | Checking the eligibility of nodes

In the process of finding a consensus, if the current block height is h, the node can calculate a nonce chain index N based on (1) and can determine a nonce by indexing the nonce chain.

$$N = h - start\_height. \tag{1}$$

A node performs the eligibility check for a member of a congress according to (2). The header $h - 1$ is the header hash of the last block, and *nonce*(2N) is obtained from the node's own nonce chain. These two values are hashed, and the obtained result is compared with a threshold referred to as "difficulty." If the comparison outcome is true, then the node qualifies as a candidate for the congress. The value of difficulty is determined based on the selection probability, p, of the node.

$$H(header\, h-1 || nonce\,(2N)) < difficulty. \tag{2}$$

A node cannot predict the unexposed nonce of other nodes. Moreover, a node knows its own nonce value in advance, but it cannot predict unpublished blockhead values. Therefore, it is impossible to predict who will be a member of a congress. A node that has passed the eligibility check publishes its own *nonce*(2N) and block height h. We add a digital signature to verify the integrity of the data to be published.

3.2.1 | Verification of nodes claiming eligibility

To verify the eligibility claim of a node, we check that h published by the node has the same value as the last block *height* + 1 published to the network. If this result is true, the value obtained by hashing the node's *nonce*(2N) by $(2N - 1)$ times is compared with the value of *nonce*(1) issued from that node. If the result is true, then the condition of (2) is checked. If the outcome is true, then the node's eligibility claim is true; therefore, it can be included in the subsequent block congress. However, if the result of the test is false, then it is considered that the node made a false claim, and consequently, it is excluded from the subsequent block congress and can be added to the blacklist. In general, digital signatures are used to ensure data integrity. If the number of congress nodes increases to 1000 or more, the integrity check causes a delay of a few seconds. If an attacker creates a large number of signed eligibility checks and uses them to attack a server, the process can be delayed, and the entire service can fail. In this paper, we propose a simple verification method to check the node's eligibility by computing two hashes and performing a comparison operation. Hash operations can be performed considerably faster than the asymmetric cryptographic operations used in digital signatures. The load on the verification server can be reduced to that of checking the electronic signature of only a node that failed the eligibility check.

3.3 | Message integrity check

The proposed consensus algorithm employs multisignatures to reduce the number of consensus messages. The message integrity can be checked by using two nonces sequentially from the nonce chain. We assume that Node i is selected as a congress candidate as a result of the previous eligibility check. The node publishes *nonce*(2N) and h to prove its qualification. At the same time, Node i generates $msg = nonce(2N)||pk_i||Q_i$ including pk_i and Q_i and then calculates and publishes the message digest value of *nonce*($2N + 1$) and message(*msg*) data, as

$$msg||H(msg||nonce(2N+1)). \tag{3}$$

Having *nonce*(2N), it is difficult to predict *nonce*($2N + 1$); therefore, the node publishes *nonce*($2N + 1$) in the agreement process to enable the integrity check of *msg*. In (3), || refers to an operation that consists of connecting two arguments by a concatenate operation to create a joint dataset and $H(\,)$ is a hash function to calculate a message digest.

As shown in Figure 2, the proposed distributed consensus algorithm is composed of the following stages: request, delegate request, prepare, and commit. The client broadcasts requests to the participating nodes, and then, the nodes determine whether the request is abnormal and save normal requests in their own proposed transaction pool. Node i selected as a congress member in the delegate request stage transmits a message, including the requests stored in the proposed transaction pool, the value of its own *nonce*($2N + 1$), and the digital signature of these messages, to Server 0. At this time, Server 0 verifies the delegate request of Node i based on the process of Algorithm 1.

Algorithm 1 Pseudocode for request verification.

```
procedure CHECK_NODE_REQUEST((nonce (2N), nonce (2N + 1), digest, msg))
  if H(nonce(2N + 1)) = nonce(2N) then
    if H(msg, nonce(2N + 1)) = digest then
      node_request = good
    end if
    node_request = bad
  end if
  Return node_request
end procedure
```

An attacker can compromise the transferred *msg* but cannot create a message digest without knowing *nonce*$(2N + 1)$. Only when the verification result of Algorithm 4 fails, does Server 0 check the digital signature to reduce the verification load of Server 0. It should be noted that multisignatures are vulnerable to rogue keys [5,6,18,19]. In general, when registering a public key to prevent a rogue key attack, it is necessary to acquire the knowledge of a user who can verify the secret key [20]. In this study, we sought to implement protection against rogue key attacks using a nonce chain. As the attacker cannot predict the nonce arbitrarily, the validity of the nodes, pk_i and Q_i, can be confirmed by applying the nonce inspection process. When Server i publishes the value of *nonce*$(2N + 1)$ in the delegate request stage, $H($*nonce*$(2N + 1))$ must be the same as the value of *nonce*$(2N)$ that was published by Node i in the process of selecting a congress.

3.4 | Role of congress members

In the process of selecting congress members, the role of each node must be assigned, as shown in Figure 2. Servers 0–3 are the congress members of the current block. Server 0 consists of one node, and Servers 1–3 consist of $3f(f \geq 1)$ nodes. The congress members are changed for each block. We perform the following operations to grant the role of Server 0 to any node in the congress: a part of the $H($*header* $h-1 \| $*nonce*$(2N))$ data transmitted by each node is cut (for example, the most significant of 32 bits) to create a coupon. At this time, the node with the minimum coupon value performs the role of Server 0. If there are two or more nodes with the same minimum coupon value, the node with the smallest *nonce*$(2N)$ value is selected as Server 0. If a node uses the normal *nonce*$(2N)$ and *nonce*$(2N + 1)$, *nonce*(1) of the node can be updated with *nonce*$(2N + 1)$, and at the same time, *start_height* is updated with the current block height. This approach can serve to reduce hash computation delays in the future. Moreover, if a problem arises involving Server 0 in the consensus and the agreement becomes impossible, the node having the smallest coupon other than Server 0 can assume the role of the new Server 0.

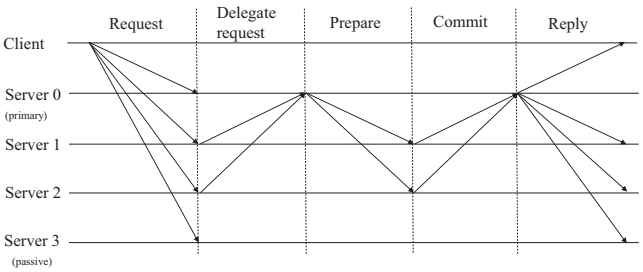

FIGURE 2 Proposed Byzantine agreement among decentralized agents (BADA) consensus algorithm

The next section explains the determination of a congress and quorum for a consensus.

3.5 | Size of congress and quorum

In the case of the PBFT algorithm, considering the number of consents equal to $f + 1$ at the prepare stage, the agreement is completed if the consent of $2f + 1$ is obtained at the commit stage. Thereby, the PBFT algorithm requires a $3f + 1$ size congress and a $2f + 1$ size quorum to be able to tolerate f Byzantine nodes. This condition is also applied to the random selection of consensus nodes. The reason for requiring $f + 1$ consents at the prepare stage is that the considered Byzantine environment constitutes at most 33%, and therefore, at least one normal consent is included in $f + 1$; consequently, the prepare stage cannot be completed only by Byzantine nodes. Therefore, to design a new Byzantine resistance consensus algorithm, a means of detecting data corruption and keeping the percentage of Byzantine nodes below 33% should be provided in the communication process.

In this study, we selected a congress through a threshold comparison, as shown in (2). The nodes are selected in a Bernoulli process, where the success probability is p. In addition, the selection of a congress from n nodes results in a binomial distribution in terms of the problem of finding the probability distribution of congress member x. From the probability distribution, when a PBFT agreement is made by any number of nodes, x, selected in a particular moment, (4) returns the probability value for the Byzantine node to win the agreement. To ensure that the agreement operates appropriately, the selection probability, p, for a node, where the probability value in (4) is less than or equal to a certain value, must be determined.

$$\sum_{x=1}^{n} \sum_{k=\left\lfloor \frac{x-1}{3} \right\rfloor + 1}^{x} \binom{b}{k} p^k (1-p)^{(b-k)}. \tag{4}$$

However, even if p is determined, it is not guaranteed that all nodes will respond within a fixed time T through the asynchronous network. It is not possible to ensure that z is the same as the number of selected nodes x, where z is the number of nodes responding. Therefore, it is difficult to guarantee that the proportion of Byzantine nodes is less than 33%.

Nodes that participate in the pool have a nonce chain generated from each master key. As this nonce chain is a random value that is not related to the values of other nodes, the selection of nodes is an independent event. Assuming that the consensus algorithm can tolerate Byzantine faulty nodes at the maximum b_p rate, the maximum number of Byzantine nodes contained in all nodes is $b = nb_p$. As this is an independent event, the cumulative probability that f or fewer Byzantine nodes are selected can be calculated as

$$\sum_{k=0}^{f} \binom{b}{k} p^k (1-p)^{(b-k)}. \tag{5}$$

Therefore, when the probability that the Byzantine nodes exceed f is less than or equal to $P_{\text{max\_bzt}}$, a condition can be set, such as

$$1 - \sum_{k=0}^{f} \binom{b}{k} p^k (1-p)^{b-k} \leq P_{\text{max\_bzt}}. \tag{6}$$

Assuming that the number of nodes participating in the pool is n and that the proportion of Byzantine nodes is b_p, (6) shows that the probability of the largest number of Byzantine nodes to be selected with a given probability p exceeding f is smaller than $P_{\text{max\_bzt}}$. While forming a congress $3f + 1$ in size under the same conditions of n and p, the maximum probability that Byzantine wins is less than $P_{\text{max\_bzt}}$, and the PBFT agreement is possible. At this time, if the number of responding nodes is $3f$ or less, it is not possible to update a new congress composed of $3f + 1$. To change the congress according to every height, the cumulative probability of being selected must be minimized to be below $3f$. We define (7) with the cumulative probability that nodes of $3f$ or less are selected less than $P_{\text{min\_node}}$.

$$\sum_{x=0}^{3f} \binom{n}{x} p^x (1-p)^{n-x} \leq P_{\text{min\_node}}. \tag{7}$$

Assuming that the number of participating pools is n and that the Byzantine fraction b_p that the algorithm is required to tolerate is given, one can calculate the probability, p, to satisfy (6) and (7) at the same time. On the above basis, we propose a new scheme to determine the size of the quorum and the congress, as follows.

3.5.1 | Decision scheme

When the number of participating pools is n and the tolerated Byzantine ratio is b_p, we calculate the probability, p, to satisfy (6) and (7) at the same time and set the congress size fixed at $3f + 1$ and the quorum size at $2f + 1$. At this time, if the number of congress candidates obtained across the network is less than $3f + 1$, the congress is not updated, whereas if it is $3f + 1$ or more, an arbitrary (first-come-first-served) $3f + 1$ congress is formed. In the study, we set $P_{\text{max\_bzt}}$ to $1.1e^{-16}$ and $P_{\text{min\_node}}$ to $1.0e^{-6}$. By using the SciPy package in Python, the maximum cumulative probability of binomial distribution is obtained, being approximately $1 - 1.1e^{-16}$. Assuming that the period of the consensus is 10 seconds, if more than f Byzantine nodes are selected under a given $P_{\text{max\_bzt}}$ condition, they can occur once in 26 billion years. Moreover, the probability of selecting $3f$ or fewer congress nodes is $P_{\text{min\_node}}$.

Figure 3 shows the results of simulating an example of the proposed scheme assuming that the number of n is 10 000 and the Byzantine proportion is 20%. The Y-axis refers to the number of nodes corresponding to each condition and the X-axis to the average number of randomly selected nodes. Let us suppose that probability p is set to 0.1419; then, the probability that the number of Byzantine nodes exceeds 418 is approximately $1.1e^{-16}$ according to (6). Further, $3f$ in (7) becomes 1254, and the probability that the number of selected nodes is $3f$ or less is approximately $9.6e^{-7}$. Considering this condition, a PBFT agreement within random selected nodes is possible, in which the probability that Byzantine wins is $1.1e^{-16}$ or less.

Figure 4 represents the probability p, congress size $3f + 1$, and maximum number of Byzantine nodes f, which simultaneously satisfy (6) and (7), when the proportion of Byzantine nodes is maintained at 20% and the number of participation pools is changed up to 50 000.

Previous methods were limited in that a minimum number of nodes were required to satisfy the design conditions. Unlike in the previous methods, in the scheme proposed in the present paper, the design conditions are satisfied according to the change in the number of nodes, as, given the possibility to calculate the probability p, the operation is possible even when the number of nodes participating in the agreement is small.

4 | BYZANTINE AGREEMENT AMONG DECENTRALIZED AGENTS (BADA) ALGORITHM

In this section, we discuss multisignature-based consensus algorithms applied to reduce the number of messages required to reach an agreement to $O(N)$, under the assumption that the scheme proposed in the previous section is used for a congress and quorum.

4.1 | Message complexity

In general, $O(N)$ is an expression of message complexity, which refers to the number of messages exchanged between nodes in a distributed network environment and is calculated mainly through an analysis that considers the worst-case scenarios. In other words, an evaluation can be obtained by calculating the number of messages sent and received between nodes in the process of reaching an agreement using the algorithms of the variance agreement. The number of messages in this process depends on the number of nodes that participate directly in the agreement. If f is 1 in PBFT, there are four nodes and the number of messages sent and received during the PBFT commit phase can be calculated as $4 \times 3 = 12$. If this is done in a network of N nodes, the number of messages that occur can be calculated as $N(N-1)$. If this is expressed using big-O notation, which is a type of complexity notation, the message complexity can be expressed as $O(N^2)$. The message complexity that occurs in the commit phase of MinBFT is $O(N^2)$. FastBFT has a message complexity of $O(N)$, which occurs at the committee stage as a result of gathering messages through sharing.

In the proposed BADA algorithm, the largest message exchange is $2f$ in the commit stage of Figure 2, where $2f < N$ is shown in Figure 4. The message complexity of BADA is $O(N)$.

4.2 | Multisignature algorithms

Multisignature is a scheme to produce a single signature that aggregates signatures from a set of users who signed the same message [21]. A pairing-based multisignature has the advantage that it can be signed without the need to establish a quorum in advance during the process of signature integration. It requires secret sharing preprocessing; however, it can aggregate node signatures simultaneously following the order in which the quorum has not been predetermined [22]. Such a structure has the advantage that it can aggregate any $2f + 1$ signatures delivered previously without an agreement being preset across the quorum nodes in an asynchronous network such as a P2P network. However, the problem arises that a fast cryptographic acceleration algorithm to complete the entire agreement process in a few seconds with a key length of 256 bits or more [23] has yet to be introduced. In addition, a pairing cryptograph can be defined only on a supersingular elliptic curve. When a digital signature algorithm, such as the elliptic curve digital signature algorithm (ECDSA) in blockchain, is used, it is suggested that two elliptic curve and encryption methods be employed. However, it is not compulsory to use pairing cryptography for multisignature. The Schnorr multisignature is a possible example.

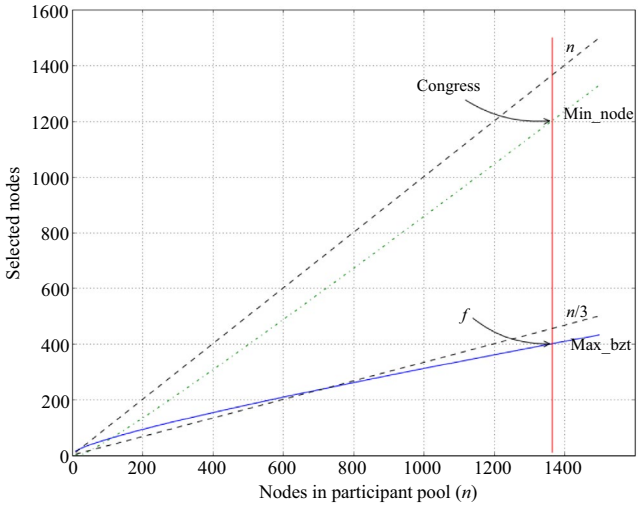

FIGURE 3 Determination of the congress and the quorum using maximum Byzantine distribution

The Schnorr multisignature must, however, fix and set up the quorum in advance, although the signature integration process is simple. In the case of the EC-Schnorr multisignature, the deficiency exists that the process of merging the public key and random value of each signer has to be completed in advance before the signatures are merged [18]. The Schnorr multisignature algorithm is vulnerable to rogue key attacks and requires a means of protecting signatures from fake keys. However, the process of combining signatures is simple and has the advantage that the entire consensus algorithm is completed in a few seconds using keys of 256 bits or more. The Schnorr signature can be used as a substitute for the ECDSA digital signature.

Table 1 presents the simulation of the time required to process EC-Schnorr multisignatures, when the value of probability p is selected as 0.1419, under the assumption that the number of participating pools is 10 000 and the Byzantine proportion is 20%. The secp256k1 elliptic curve is used for the Schnorr signature. It is obtained as a result of simulation using Python on a computer equipped with a 2.3 GHz Xeon process. The time consumed by each node to generate a public key pair and a Q_i pair is approximately 101 ms on average. Server 0 requires approximately 1.164 seconds to complete the remaining process, except for key generation, whereas nodes other than Server 0 require approximately 9 ms; therefore, the time consumed by Server 0 must be minimized.

4.3 | Operation of the proposed consensus algorithm

The proposed consensus algorithm employs the EC-Schnorr multisignature in the process of finding the agreement. The consensus algorithm presented in Figure 2 is described step by step in the following.

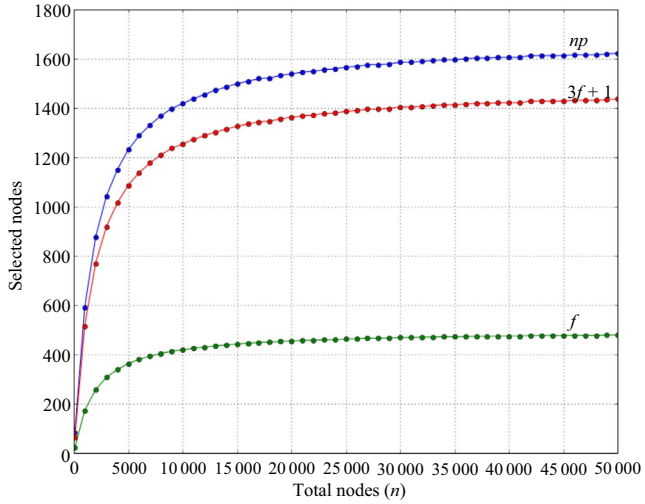

FIGURE 4 Scheme satisfaction condition according to the number of nodes

TABLE 1 Simulation results of the delay time of each part of the EC-Schnorr multisignature with 837 nodes

	Time (ms)	
action	**Server 0**	**Server 1–3**
Private, public key	50.523	50.523
k, Q_i	50.576	50.576
$r = H(Pk\|\|Q\|\|Msg) \bmod p$	436.567	N/A
$s_i = k_i - rsk_i \bmod p$	N/A	8.820
Invalid Q_i check	620.660	N/A
$S = \sum_{i=1}^{n} s_i$	3.392	N/A
Sig verify	103.872	N/A

4.3.1 | Request

In the proposed algorithm, the client is not able to specify Server 0, as the congress is not fixed but is reconstructed for each block. The client broadcasts a request to all nodes of the participation pool. Therefore, other nodes not selected in the congress continue to receive the request. In general, the PBFT algorithm assumes Server 3 is in a passive state if it does not respond within a given time.

4.3.2 | Delegate request

Participating nodes in the pool determine whether there is an abnormality in the received request and place a normal transaction in their own proposed transaction pool. A new consensus can be initiated when the previous block Server 0 sends a reply block. The new agreement is initiated with a delegate request. At this stage, the node selected as a congress member in the new block generates a delegate request message with the proposed transaction stored in the transaction pool and sends it to Server 0. The delegate request message of the node contains the proposed transaction, $nonce(2N + 1)$, and $Q_i(h + 1)$. The proposed transaction is a set of transactions that were stored in the proposed transaction pool of a node. $nonce(2N + 1)$ and $Q_i(h + 1)$ can be used in the consensus of the subsequent block height if the congress is not updated appropriately. In addition, the digital signature of the sending server is attached to the message. The signature can be used in the commit phase to check for errors in the message.

The operation of Server 0 receiving the delegate request message is represented in Algorithm 2. Server 0 receives the delegate request from a node and repeats the operation until the quorum size $2f + 1$, including itself, is satisfied in line 3, that is, the "While" statement of Algorithm 2. In the block, where the agreement is completed, pk_i and Q_i of the nodes selected as the congress in the subsequent block are recorded. Therefore, line 5 searches the index of the congress node that has sent the delegate request message and combines the values of the multisignature keys in line 6. The MT_LIST function of line 14 is a processor that provides a list of transactions from the delegate requests of a server and creates a map in which each element of the list is requested by each server. When the "While" statement is completed, r of line 11 is generated, and if the value of line 12 is returned, a prepare message including this value is generated and transmitted to the member of the congress.

4.3.3 | Prepare

Using the line 14 function of Algorithm 2, $f + 1$ or more nodes simultaneously request results corresponding to t_list and t_map. At this time, Msg of line 10 is the head value of the complete block including the pregenerated t_list and t_map. At the prepare stage, Server 0 generates a prepare message including Q, Pk, r, and Msg returned at line 12, and transmits it to the congress. At this time, the prepare message is amended with the digital signature of Server 0 so that the attacker cannot change the content in the message delivery path. The prepare block must be a complete block including transactions as in a general block chain. It is a complete block, because we need to use multisignatures signed by all the agreed quorums on the prepare block provided by Server 0 so that bits in future blocks cannot be modified or added. In addition, to identify each of the $2f + 1$ nodes, we create and send a delegate server bitmap. The delegate server bitmap can also be calculated based on the proposed transaction bitmap (t_list, t_map) if the delegate request is restricted to include at least one

transaction. Therefore, Server 0 adds its own digital signature to a message containing information (prepared transaction, proposed transaction bitmap, delegate server bitmap, Q, Pk, and r) to create a prepare block and transmit it to Servers 0–2, which perform multiple signing on that block. The quorum node confirms that the prepare message will be reflected without errors including transactions that are requested from at least $f + 1$ nodes of the quorum. This prevents Server 0 from creating a message with false transactions.

Algorithm 2 Pseudocode for delegate request

```
1:  procedure DELEGATE_REQUEST( )
2:    node_num = 0, t_list = 0, t_map = 0, Pk = 0, Q = 0
3:    while node_num < quorum do
4:      Input nodes delegate request
5:      find node index i
6:      Pk += pk_i, Q += Q_i
7:      t_list, t_map = MT_LIST(request, t_list, t_map)
8:      node_num += 1
9:    end while
10:   Msg = t_list, t_map, …
11:   r = H(Pk ||Q|| Msg)mod p
12:   Return Q, Pk, r, Msg
13: end procedure
14: procedure MT_LIST(request, t_list, t_map)
15:   for each item in request do
16:     if item is in the t_list then
17:       push item in t_list
18:       t_map[len(t_list)] = 1
19:     else
20:       t_map[t_list[item]] +=1
21:     end if
22:   end for
23:   Return t_list, t_map
24: end procedure
```

4.3.4 | Commit

Algorithm 3 is used to confirm the process of preparing a message. Servers 0–2, which receive the prepare message, verify the digital signature of Server 0 by line 2 of the algorithm to check whether there is an abnormality in the message. If the verification result of the digital signature is true, the remaining operation of the algorithm is performed. However, if it is false, then, Server 0 immediately returns false and notifies other nodes of this result and activates an empty agreement.

Algorithm 3 Pseudocode for confirming a prepare message

```
1:  procedure CHECK_PREPARE_MSG(Q, Pk, r, Msg,
     Sign)
2:    if Sign is valid then
3:      Pk = 0, Q = 0
4:      node_list = CHECK_BITMAP(Msg)
5:      if node_list > 0 then
6:        for i in node_list do
7:          Pk += pk_i, Q += Q_i
8:        end for
9:        if r = H(Pk ||Q|| Msg)mod p then
10:         Return True
11:       end if
12:     end if
13:   end if
14:   Return False
15: end procedure
16: procedure CHECK_BITMAP(Msg)
17:   node_list = 0
18:   if server count in Msg < quorum then
19:     return 0
20:   end if
21:   for item, index in Msg do
22:     if bit count of item = 0 then
23:       Return 0
24:     else
25:       push index in node_list
26:     end if
27:   end for
28:   Return node_list
29: end procedure
```

Upon receiving the prepare message, the server executes lines 3 to 12 of Algorithm 3 to verify the validity of r generated by Server 0. The required quorum is checked at line 18, and as a result, it returns 0 if false, and exits the check_bitmap( ) function. In addition, line 22 checks whether the transaction requested by at least one or more servers is included in the Proposed Transaction, and returns 0 if the result is false. Line 25 identifies the indices of the transactions that have obtained consent from $f + 1$ or more nodes, adds them to node_list, and returns the final result.

If the value of node_list is 0 in line 5, the line returns false and terminates the function. When node_list is not empty, pk_i and Q_i are extracted for index i stored in the node_list, and Pk and Q are combined in line 7. When the loop of line 5 ends and the result of executing line 9 is true, the node partial signature $s_i = k_i - rsk_i$ is created using the secret key

(sk_i) and secret random value (k_i) of a node, and the message containing the result is transmitted. If the result of line 9 is false, Server 0 receives invalid Pk, Q, r, or Msg. Therefore, other nodes are notified that they should proceed with the empty agreement. Server 0 does not immediately verify partial signatures individually. It takes time to verify hundreds of partial signatures, which affects the overall agreement time. Thereafter, Server 0 merges each partial signature into a tree and first verifies the validity of the multisignature. If the verification result is false, the fake signature is searched in the tree and the digital signature of the server that has sent the fake signature can be additionally checked.

The anomaly of the partially merged signature can be checked, as shown in Algorithm 4, by the characteristics of the EC-Schnorr multisignature. To accomplish this, it is required that Algorithm 4 up to line 6 be executed for two values (Pk_{par} and S_{par}) to obtain Q_{par}, to repeat the process of executing Algorithm 4 up to line 6, which matches these results again, and then to construct a tree. At this time, the vertices of the tree become S, Q, and Pk. Here, S is a part of the value of the Schnorr multisignature result, Q is a value obtained by combining the parts of Q_i, and Pk is the public key of the combined multisignatures.

Algorithm 4 Pseudocode for verifying multisignature

1: **procedure** PARTIAL_VARIFY(pk_i, pk_j, Q_i, Q_j, s_i, s_j)
2: $Pk_{\text{par}} = pk_i + pk_j$
3: $Q_{\text{par}} = Q_i + Q_j$
4: $r = H(P\ k_{\text{par}} \| Q_{\text{par}} \| Msg) \bmod p$
5: $S_{\text{par}} = s_i + s_j$
6: $Q_{\text{var}} = S_{\text{par}}G + rP\ k_{\text{par}}$
7: **if** $Q_{\text{par}} = Q_{\text{var}}$ **then**
8: Return True
9: **else**
10: Return False
11: **end if**
12: **end procedure**

If Server 0 scans entire partial signatures sequentially, the identification of fake digital signatures requires, on average, a number of verifications equal to $(2f + 1)/2$. However, it is possible to discover a server that sends a fake signature by applying the partial signature inspection through a tree and one digital signature inspection.

As soon as Server 0 has received s_i from all quorum servers, the subsequent steps are to create S and execute verification; thereafter, if the obtained result is true, Server 0 generates a message that includes the prepared transaction, Pk, and (r, S). It completes the agreement by forwarding the message to all nodes.

4.4 | Consensus error handling procedure

The following process can be performed to address errors that occur in the agreement process. The delegate request, prepare, and commit processes set a timer at the start of each step; erroneous handling can occur if the following procedure is not started before the timer expires. The processing method applied when an error occurs at each step is now described. An agreement on a block that does not contain a transaction constitutes an empty agreement. An empty agreement does not include an effective transaction but has the effect of changing the congress. In general, the PBFT algorithm has a fixed consensus node, and, if the agreement fails, it applies a view change to resolve this. However, as the nodes of the consensus are not fixed, it is less computationally expensive to reconfigure the consensus node rather than apply a view change if the agreement fails. An empty block can be created as a means of reducing the complexity of the protocol while reducing the risk, instead of allowing an unreliable node to return a consensus upon the failure of the agreement.

4.4.1 | Delegate request stage error handling

In the current block consensus, Server 0 starts the timer of the delegate request stage at which the reply block corresponding to the previous block height is received and the current block agreement is initiated. If normal delegate requests corresponding to the quorum have not been received before the termination of Server 0's delegate request timer, an error message is generated indicating that fewer requests than the quorum have been received. At this time, the quorum can be reconfigured to execute the empty agreement, except for the node that does not respond in the congress. Moreover, after receiving the reply block of the previous agreement, each of Servers 1–3 starts its own delegate request timer, while forwarding its own delegate request. The server with the smallest coupon among the remaining ones, except for the coupon possessed by Server 0, acquires a new Server 0 qualification and tries an empty agreement, if Server 0 does not transmit the prepare block before the delegate request timer expires. In this case, the block may contain a log of errors.

4.4.2 | Prepare stage error handling

Server 0 starts the prepare timer while forwarding the first prepare block. If Server 0 does not receive s_i from all the servers in the quorum before the prepare timer expires, the algorithm implies that the servers that do not send s_i to the blacklist should be added and the quorum should be reformed

to exclude the blacklist corresponding to the congress, and the operation should proceed with the empty agreement.

4.4.3 | Commit stage error handling

Even when Server 0 does not have a valid integrated signature, the algorithm determines another server that sends fake signatures and adds it to the blacklist. Then, the blacklist corresponding to the congress is excluded, and the empty agreement can proceed. Moreover, each server corresponding to the quorum starts a commit timer when it receives a prepare packet of Server 0. If Server 0 has not started replying before the commit timer expires, it is replaced with a new Server 0 and an empty agreement can proceed.

5 | IMPLEMENTATION AND RESULTS

This section describes the experimental environment and explains the analysis of the experimental results for evaluating the proposed BADA algorithm.

5.1 | Experimental environment

The BADA algorithm was implemented in Python and then executed on docker containers running on the Linux operating system. In the BADA algorithm, the hash function required for the nonce chain and block used the SHA256 hash function. The signature was used to check the integrity of the message, and the multisignature scheme for consensus used EC-Schnorr signatures with the secp256k1 elliptic curve.

To analyze the performance characteristics of the BADA algorithm in large-scale nodes, consensus nodes were constructed using from 200 to 1400 docker containers in two experimental environments. The first experimental environment consisted of four Dell PowerEdge R730 servers equipped with two Intel Xeon Gold 6152 CPUs, 512 GB memory, 10 G Ethernet interfaces per server, and CentOS 7.5 as the operating system. The second experimental environment consisted of 64 Dell PowerEdge R530 servers with two Intel Xeon E5-2623 CPUs, 64 GB memory, 10 G Ethernet interfaces per server, and CentOS 7.5 as the operating system.

In this experiment, we assumed that 20% of the total nodes were Byzantine nodes. The BADA algorithm randomly selects a $3f + 1$ congress for each block height to proceed with the consensus. For this purpose, the probability of congress selection, p, is calculated and used so that an average of $3f + 1$ nodes out of the total number of nodes are selected as the congress, as shown in Figure 4. For example, if the total number of nodes is 1000, using $p = 0.59$ results in the number of non-Byzantine nodes in the congress node being less than $1.1e^{-16}$, and the probability that the number of congress nodes is less than or equal to 514 ($=3f + 1 = 3 \times 171 + 1$) can meet the $1.0e^{-6}$ condition. In addition, if 100 nodes participate in the consensus, using $p = 0.83$ results in the probability of exceeding 20 Byzantine nodes being zero, and the probability of selecting fewer than 61 congress nodes being less than or equal to $1.0e^{-6}$.

5.2 | Decentralization simulation

Decentralization means that authority is not concentrated on certain nodes participating in the consensus and that all the nodes participate fairly. To achieve this, the BADA algorithm adopts the PoN algorithm using the nonce chain. To evaluate the proposed algorithm in terms of decentralization, we validated the eligibility of a node to participate in a consensus using its own nonce value corresponding to the height of the block to be agreed upon by all nodes participating in the consensus and form a quorum with the nodes that responded quickly. To verify that the proposed PoN algorithm is decentralized, 1000 nodes were configured and the quorum selection probability, p, was set to 0.59. Then, 1000 blocks were created to measure the frequency of the nodes participating in the quorum for each block. Figure 5 shows the frequency of quorum participation by node and the distribution of these values.

In Figure 5, the horizontal axis represents the frequency with which a node was selected as a part of a quorum for a consensus considering 1000 blocks and the vertical axis represents the number of nodes participating in these cases. The analysis showed that the number of nodes that participated in a quorum was 573.54, on average, and the form of the distribution was normal, 29.17. The maximum number of blocks participating as a quorum is 631, and the minimum number of blocks is 230. As shown in the figure, 23 nodes participated in 593 blocks for a quorum, and all the nodes participated in between 466 and 631 blocks, with the exception of one node that participated in 230 blocks, indicating that all the nodes participated in the quorum fairly.

5.3 | Block interval analysis

Conventional PBFT-like consensus algorithms have limitations in terms of scalability. The number of nodes participating in the consensus increases, and consequently, the number of nodes participating in the agreement requires a message complexity of $O(N^2)$. The BADA algorithm is intended to provide $O(N)$ message complexity to overcome these limits on scalability, thereby providing shorter consensus times. Therefore, in this subsection, we discuss the measurement of the block creation time of the BADA algorithm and describe the obtained results to assess its applicability.

In the two experimental environments described in Section 5.1, from 200 to 1400 docker-based nodes were run and the test was conducted with the probability of node selection, p, set to 1 so that all nodes could be configured in the consensus. Figure 6 shows the results of the time measurement with respect to block creation according to the number of nodes participating in the consensus in the two test environments. The first test environment consisted of 4 Dell PowerEdge R730 servers, and the second consisted of 64 Dell PowerEdge R530 servers. As shown in Figure 6, the block creation time was measured as approximately 4.4 seconds for four servers and 5.61 seconds for 64 servers, when the number of the consents was 200. However, as the number of the consensus increased, the block generation time in the 4-server environment also increased, exceeding that in the environment with 64 servers. This is because the four servers have 44 CPU cores per server, leading to a better performance than in the second environment with 64 servers having eight CPU cores per server. At the same time, the number of consensus nodes per server should be adjusted to form the same number of consensus nodes involving fewer servers. If the number of consents is 1200, or 300 consensus nodes per server, 7 or 8 consensus nodes per core operate, and in the case of 64 servers, only 18 or 19 consensus nodes per server operate with approximately 2 or 3 consensus nodes per core. If the number of the congress was 1400, then the test environment with four servers could not be configured because of a lack of system resources. In turn, in the environment with 64 servers the block creation time was measured as approximately 12.7 seconds. As shown in Figure 4, even if the total number of nodes is increased to more than 50 000, a consensus can be reached by forming a congress of approximately 1400 nodes providing an acceptable block creation time of less than 13 seconds. This is because the message complexity required for reaching consensus in the BADA algorithm is $O(N)$, and the EC-Schnorr multisignature verification algorithm described in Section 4 can be used to reduce the time required for the corresponding operation. In Table 2, we show a comparison of the proposed BADA algorithm and the methods proposed in previous papers.

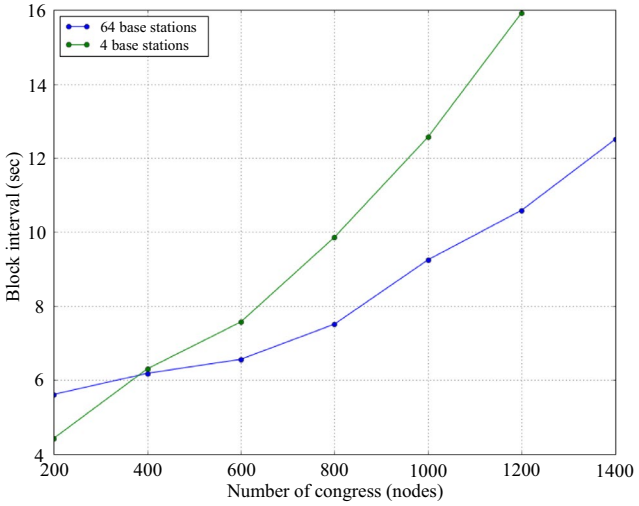

FIGURE 6 Consensus time with the increasing number of nodes

From the above analysis results, we conclude that the proposed BADA algorithm can be deemed to overcome the limitations of existing consensus algorithms in terms of ensuring decentralization and scalability across tens of thousands of nodes.

In MinBFT or FastBFT, there is no random node selection function; thus, all nodes must participate in the consensus. Therefore, in this table, we assume that 50 000 nodes participate in the consensus and show the results of the comparison of BADA and algorithms that can arbitrarily select congress nodes. However, a direct comparison was difficult, as Zilliqa and Algorand already have commercial platforms, whereas the proposed algorithm does not, as it is only a consensus

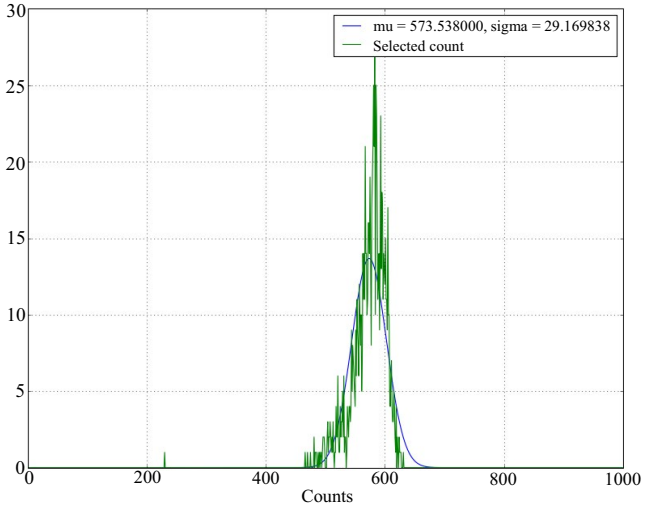

FIGURE 5 Statistics on acquisition of consensus

TABLE 2 Comparison of the proposed algorithm and previous methods

	BADA	Algorand	Zilliqa
Message complexity	$O(c)$, where $c < n$	$O(cn)$, where $c < n$	$O(n)$
Congress size	$3f + 1$	$3f + 1$	$3f + 1$
Congress size with safety violation prob.			
$1e^{-6}$	670	N/A	800
$5e^{-9}$	862	2000	N/A
$2e^{-16}$	1433	N/A	N/A
Congress change period	Per block	Per block	Replace only some nodes
Special requirements	No	No	PoW support required

algorithm. Therefore, the numbers of congresses selected under the same safety violation probability condition were compared. In the case of Zilliqa, this is the minimum number of nodes for a $1e^{-6}$ safety violation, and there is no assumption that 50 000 nodes participated in the agreement.

6 | CONCLUSIONS

In this paper, we proposed a new distributed consensus algorithm, BADA, in which non-fixed nodes can find agreement through $O(N)$ message exchanges. The nodes in the BADA algorithm can be decentralized by applying PoN. With regard to PoN, we presented a method to calculate the parameters used to define the number of a congress and a quorum by the hash-based method to satisfy the Byzantine tolerance. In addition, the distributed consensus algorithm can provide extensibility across several arbitrarily selected nodes considering the same argument, and hence, it can be applied to a group of nodes from at least five to several tens of thousands or more nodes. Through the experiments, we confirmed that the proposed algorithm operated normally in the experimental environment driven by 1400 dockers distributed across 64 servers.

ACKNOWLEDGMENT

We sincerely thank Prof. Seungwon Shin for his detailed and valuable comments on the earlier version of the draft.

ORCID

Jintae Oh https://orcid.org/0000-0002-4372-0943

REFERENCES

1. J. Poon and T. Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2015, available at https://lightning.network/lightning-network-paper.pdf.
2. J. Poon and V. Buterin, *Plasma: Scalable autonomous smart contracts*, White paper, 2017, available at https://plasma.io/plasma.pdf.
3. S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2009, available at https://bitcoin.org/bitcoin.pdf.
4. Y. Gilad et al., *Algorand: Scaling byzantine agreements for cryptocurrencies*, in Proc. Symp. Oper. Syst. Principles (Shanghai China), 2017, pp. 51–68.
5. L. Harn, *Group-oriented (t, n) threshold digital signature scheme and digital multisignature*, IEE Proc. Comput. Digital Techn. **140** (1994), 307–314.
6. K. Ohta and T. Okamoto, *Multi-signature schemes secure against active insider attacks*, IEICE Trans. Fund. Electron. Commun. Comput. Sci. **E82-A** (1999), 21–31.
7. L. Lamport, R. Shostak, and M. Pease, *The byzantine generals problem*, ACM Trans. Program. Lang. Syst. **4** (1982), 382–401.
8. M. Castro and B. Liskov, *Practical byzantine fault tolerance*, USENIX OSDI **99** (1999), 173–186.
9. J. Kwon, *Tendermint: Consensus without mining*, 2014, available at http://tendermint.com/docs/tendermint{_}v04.pdf.
10. J. Liu et al., *Scalable byzantine consensus via hardware-assisted secret sharing*, IEEE Trans. Comput. **68** (2018), 139–151.
11. G. S. Veronese et al., *Efficient byzantine fault-tolerance*, IEEE Trans. Comput. **62** (2013), 16–30.
12. M. Yin et al., *Hotstuff: Bft consensus in the lens of blockchain*, arXiv preprint, 2018, arXiv:1803.05069.
13. Y. Yang, *Linbft: Linear-communication byzantine fault tolerance for public blockchains*, arXiv preprint, 2018, arXiv:1807.01829.
14. P. Schindler, A. Judmayer, and E. R. Weippl, *Hydrand: Efficient continuous distributed randomness*, in Proc. IEEE Symp. Security Privacy (San Francisco, CA, USA), May 2020, pp. 73–89.
15. S. Bano et al., *Consensus in the age of blockchains*, arXiv preprint, 2017, arXiv:1711.03936.
16. Zilliqa team, *The Zilliqa technical whitepaper*, 2017, available at http://zilliqa.com.
17. L. Lamport, *Password authentication with insecure communication*, Commun. ACM **24** (1981), 770–772.
18. G. Maxwell et al., *Simple schnorr multi-signatures with applications to bitcoin*, Designs, Codes Cryptography **87** (2019), 2139–2164.
19. C. Li, T. Hwang, and N. Lee, *Threshold-multi- signature schemes where suspected forgery implies traceability of adver- sarial shareholders*, in Proc. Adv. Cryptol.-EUROCRYPT (Perugia, Italy), May 1994, pp. 194–204.
20. T. Ristenpart and S. Yilek, *The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks*, in Proc. Adv. Cryptol.-EUROCRYPT (Barcelona, Spain), May 2007, pp. 228–245.
21. D. Boneh et al., *Aggregate and verifiably encrypted signatures from bilinear maps*, in Proc. Adv. Cryptol.-EUROCRYPT (Warsaw, Poland), May 2003, pp. 416–432.
22. A. Boldyreva, *Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme*, in Proc. Public Key Cryptography-PKC (Miami, FL, USA), Jan. 2003, pp. 31–46.
23. D. Boneh, B. Lynn, and H. Shacham, *Short signatures from the Weil pairing*, J. Cryptol. **17** (2004), 297–319.

AUTHOR BIOGRAPHIES

Jintae Oh received his B.S. and M.S. degrees in Electronics Engineering from Kyungpook National University, Daegu, Rep. of Korea, in 1990 and 1992, respectively, and his Ph.D. degree in Computer Engineering from Chungnam National University, Deajeon, Rep. of Korea, in 2011. He was with the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea from 1992 to 1998, where he was a senior researcher. He was with three start-ups in St. Louis, MO, U.S.A. from 1998 to 2002. Since 2003, he has been with the Electronics and Telecommunications Research Institute, where he is now a principal researcher. His main research interests are net work security and blockchain.

Joonyoung Park received his B.S. and M.S. degrees in Computer Science and Engineering from Korea University, Rep. of Korea in 2014 and 2016, respectively. Since 2016, he has been with the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea, where he is now a researcher. His main research interests are distributed systems and blockchain.

Youngchang Kim received his M.S. and Ph.D. degrees in Computer Engineering from Chonbuk National University, Jeonju, Rep. of Korea, in 2003, and 2009, respectively. Since 2009, he has been working as a senior researcher at the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His main research interests are distributed computing and blockchain.

Kiyoung Kim received her M.S. degree in Computer Science and Statistics from Chonnam National University in 1993, and her Ph.D. degree in Computer Science from Chungbuk National University in Korea in 2002. Since 1988, she has been with the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea, where she is now a principal researcher. Her main research interests are smartphone security and blockchain.