IGD-ScoreChain: A Lightweight and Scalable Blockchain Based on Node Sharding for the Internet of Things

Elnaz Mehraein[a], Zahra Ahmadian[a,*] and Reza Nourmohammadi[b]

[a]*Department of Electrical Engineering, Shahid Beheshti University, Tehran, Iran*
[b] *Department of Software and IT, École de technologie supérieure, Montreal, Canada,*

ARTICLE INFO

Keywords:
Internet of Things
Blockchain
Sharding
Consensus algorithm
Score
Scalability

Abstract

Due to the significant development of the intelligence industry worldwide, various initiatives have increasingly recognized the value of the Internet of Things (IoT). IoT systems, however, are often hindered by fundamental challenges, such as the need for a central server to manage them. Decentralizing these systems can be achieved through the use of blockchains. Recently, there has been an increase in the popularity of blockchain in various fields, such as banking, IoT, and the intelligence industry, and human societies have taken notice of it. One of the main problems is with the scalability of such systems as the network size grows.

This paper examines how to overcome this challenge in blockchain-based IoT systems. We introduce a sharding-based blockchain that is lightweight and scalable. In the proposed method, the nodes are assigned to a number of shards based on their history of activity. As part of this study, the Improved Byzantine Fault Tolerance with Graceful performance Degradation (IGDBFT) consensus algorithm is introduced within the proposed scheme for intra-shard consensus. A solution to storing blocks and cross-shard transactions has been developed using a global chain containing parent blocks in the cloud layer. Finally, we analyze the security and efficiency of our scheme and compare our sharding-based protocol with previous protocols.

1. Introduction

The Internet of Things is an innovative architecture that connects various smart devices such as sensors, vehicles, smart home devices, and physical objects, allowing them to gather and exchange data via the internet. IoT devices may contain confidential and private information and pose many security threats aimed at exploiting weaknesses in the centralized infrastructure of these systems, which prevents their scalability [1].

Over time, IoT edge devices become empowered to mitigate issues related to centralized system vulnerabilities. A decentralized approach can be used as a possible solution to prevent existing faults and attacks from improving IoT safety. Centralized methods for securing privacy and data usage require robust servers controlled by third parties [2].

Blockchain is one of the technologies available to enhance the security and distribution of IoT. This technology helps share data securely through the use of cryptographic algorithms and a decentralized network. [3].

Switching from a centralized IoT to a blockchain-based IoT (BIoT) increases fault tolerance and eliminates unique failure points. The decentralized network architecture allows IoT devices to be autonomous. Blockchain participants can verify the authenticity of the data sent and the identity of the sending participant. However, current blockchains have limitations such as network overload and reduced scalability when increasing the number of IoT nodes and exchanging large amounts of data simultaneously, which lead to increased network traffic and additional processing costs [4]. Blockchain uses sophisticated computational and resource-consuming algorithms that reduce the throughput of IoT and cause latency inconsistent with IoT goals of responding to the user. Designing a lightweight-scalable blockchain is the perfect solution to these challenges. Adding a lightweight blockchain can change the system topology in designing a blockchain-based IoT.

1.1. Contribution

In this paper, we present a novel sharding-based method that considers dynamic node behavior to improve the scalability of blockchains at the IoT system's edge. So, by considering nodes' activity histories, our method outperforms currently used sharding techniques and enables nodes to join or leave shards without sacrificing performance. Moreover, we propose an improved DBFT algorithm with Graceful Performance Degradation (IGDBFT), which acts as an intra-shard consensus for the blockchain-based IoTs. Indeed, our method offers a solution that is specifically adapted to the dynamic nature of IoT nodes, making it a more effective and efficient method for managing this context than existing sharding-based approaches.

1.2. Background summary

The related literature can be investigated in two directions: the works focusing on the usage of blockchain in IoT systems, and the works toward the scalability of blockchain.

*Corresponding author: Zahra Ahmadian
elmehraein@gmail.com,e.mehraien@mail.sbu.ac.ir (E. Mehraein);
z_ahmadian@mail.sbu.ac.ir (Z. Ahmadian);
reza.nourmohammadi.1@ens.etsmtl.ca (R. Nourmohammadi)

1.2.1. Blockchain-based IoTs

Blockchain is a digital transaction ledger that offers trust and reliability in distributed, third-party-free contexts. Security, scalability, storage capacity, and anonymity are the most important issues with this design. Blockchain-based IoT is proposed for usage in a variety of intricate application areas, including crowdsourcing [5], smart cities [6], and healthcare [7]. Reyna et al. [8] have analyzed the challenges and opportunities of blockchain and IoT integration. Devetsikiotin and Christidis [9] believed that the operational power of the IoT might be considerably increased by adopting blockchain. They clarified how the combination of blockchain and IoT could make it easier to share resources and create an enabled service market. The capacity of blockchain to produce, store, and transmit digital assets in a decentralized and anti-tamper distribution method was presented by Samaniego and Deters [10], which has significant practical utility for the Internet of Things. Danzi et al. [11, 12] argued that although blockchain is suitable for IoT interaction, it is impossible to maintain a local version of that for power-limiting devices. So they designed lightweight protocols for collecting blockchain data. Huh used blockchain to relax the restrictions and resolve the problems with IoT client synchronization [13]. Pan et al. [14] focused on the vulnerability of IoT devices against malicious hackers, and designed "EdgeChain" to prevent possible network abuses using authorized blockchain and smart contracts to overcome security vulnerabilities. A public blockchain architecture for data authentication in the Internet of Things was created by Pinto et al. [15]. In order to minimize user privacy leaks for IoT devices in the blockchain network flexibly and securely, Cha et al. [16] investigated the architecture of a blockchain-connected gateway. Zhang and Wen [17] used blockchain and smart contracts to realize digital asset transactions and payment data in the IoT.

1.2.2. Blockchain Scalability

Scalability refers to maintaining throughput and controlling transaction confirmation delays when the number of transactions increases. With the dynamic of nodes and transactions in the IoT, the scalability of the blockchain is reduced.

This issue has been important enough to be considered in many studies of blockchain. Eyal et al. [18] designed a new blockchain protocol in Bitcoin-NG for scalability. Bitcoin-NG is a highly robust, fault-tolerant blockchain protocol with the same trust model as Bitcoin. Luu et al. at Elastico [19] uniformly divide or parallelize the mining network into smaller shards, each of which processes a set of transactions. While sharding is common in non-Byzantine environments, Elastico is the first proposal for a secure sharding protocol with Byzantine adversaries. OmniLedger [20] optimizes parallel, intra-shard transaction processing on the ledger via collectively signed state blocks and low-latency "trust-but-verify" validation for low-value transactions. Dorri et al. [21] believed that blockchain could address IoT security and privacy issues, and they proposed a small lightweight and scalable blockchain. A similar classification scheme is provided in [22]. RapidChain [23] is the first public sharding-based blockchain protocol which is Byzantine fault tolerant up to $1/3$ participants. This scheme uses an optimal intra-shard consensus algorithm to achieve a higher throughput through block pipelining, which is a novel gossiping protocol for large blocks, and a secure reconfiguration mechanism to ensure robustness. ZyconChain [24], a scalable blockchain system for broad uses, was suggested by Sohrabi et al. This paper introduces three types of blocks forming three separate chains: parent, side, and state chain in the blockchain system. Different consensuses are used to create blocks because each algorithm has unique characteristics that make it appropriate for a particular form of block. A novel scalable public blockchain called Groupchain [2] is presented by Lei et al. In order to increase the efficacy of all transactions, Groupchain uses the leader group to create blocks. It also adds incentives and payments to the incentive system to track how well the leader group members are performing.

2. System overview

IGD-ScoreChain is a blockchain-based IoT solution proposed for the IoT's edge that combines cloud-based storage and blockchain technology. The blockchain nodes are located at the edge of a partially synchronous peer-to-peer network within the fog layer, as shown in Fig. 1. Various forms of information such as sensor data, user data, device status, and smart contract data that are saved, processed, and sent between IoT devices, can be included in the data that is handled in IGD-ScoreChain. Our approach aims to manage various kinds of data of IoT networks in a flexible and scalable way while overcoming the challenges of the centralized nature of IoT systems. Actually, IGD-ScoreChain provides a secure and reliable platform for implementing transactions, improving the scalability of blockchains at the edge of IoT, and providing a more efficient and flexible solution for managing data in an IoT environment.

Transactions, in our design, are assumed to be account-based [25]. In account-based transactions, the transfer takes place between two accounts and has a sender and a recipient. The transactions connected to an account must be completed one after the other since the outcome of a transaction depends on the input state, which gives designs employing dynamic nodes more flexibility.

In fact, our design aims to consider all nodes performance in aspect of computational capacity in a fair manner that does not based on randomness issues. The system includes several main features:

- **Sharding Based on Node's Score**: This paper introduces the concept of "node's score" based on which the sharding of nodes is performed. The scores are calculated according to the node's activity history in transaction processing.
- **Mini-Block**: The shards process transactions in parallel and generate mini-blocks.

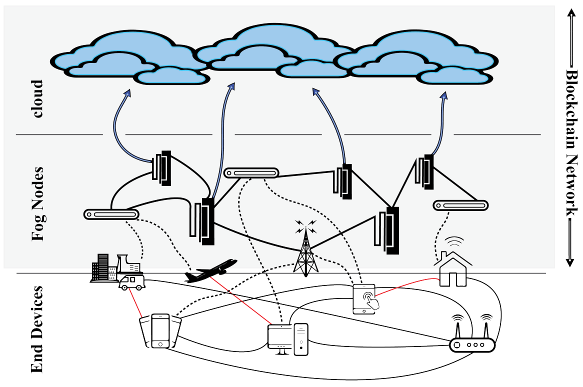


Figure 1: Overview of Blockchain Architecture in IoT.

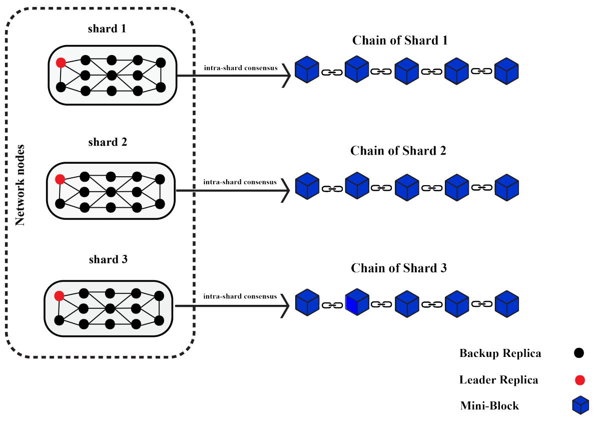


Figure 2: The simple layout of a Sharding-Based Blockchain.

- **Parent Shard**: By solving the PoW puzzle in random and uncertain rounds by fog nodes, we construct a parent shard to manage shards and store mini-blocks in the cloud layer.
- **IGDBFT Consensus**: The IGDBFT consensus based on DBFT [26] is used to reach an agreement between the nodes of the shards to process transactions and generate a mini-block.
- **Parent Block**: Mini-blocks will be stored in the cloud layer as a parent block by using the collective threshold signature.

3. Sharding in Blockchain

Sharding is one of the solutions to improve scalability. There are three types of sharding in the blockchain: 1) network nodes sharding, 2) transactions sharding 3) state sharding [27]. We focus on node sharding and its relevant challenges in order to reduce overload and increase scalability. A simple scheme of sharding is shown in Fig. 2. This technology in blockchain refers to dividing existing nodes into subnets called shards that can process transactions in parallel. Each shard has its separate chain. In our model, each shard chain consists of mini-blocks which are generated by shard nodes by processing a certain number of transactions. In a simple sharding protocol, there is a leader among each shard's members responsible for managing the shard. With an intra-shard consensus, the nodes process the transactions and confirm them.

3.1. Sharding Challenges

In the sharding method, we face some challenges that, if not appropriately addressed, will lead to severe problems, including various security attacks and throughput degradation.

1. **Nodes Assignment to shards**: One of the main challenges in sharding is to define a way for assigning nodes to shards in each round. Usually, in blockchain, it takes a certain amount of time for a specific process, such as block generation or reconfiguring nodes, to occur in the network, which is called a "round." Conventional methods for assigning nodes to different shards are random methods of selecting nodes through a random protocol like MBFT [4], Zyconchain [24], or solving puzzles in PoW like Groupchain [2]. The remarkable thing about these methods is that they ignore the performance differences between the nodes. Sometimes, the difference in nodes' throughput causes a difference in the speed of block generation or causes inconsistencies and delays in processing transactions, leading to reduced throughput or vulnerability to attacks. As a result, the activity of honest nodes in processing transactions remains ineffective. In time-sensitive IoT systems, the critical factors are the proper speed of transaction processing and reasonable throughput.
2. **Shard's Reconfiguration** : The ability to shards reconfiguration is the next challenge of a sharding-based blockchain. With a fixed sharding structure, the malicious node can penetrate the shard's structure and control the affairs of a shard based on this fixed background [27]. If a shard member remains constant in the several rounds, the proportion of members controlled by the attacker may exceed the predetermined safety threshold, such as 1/3 in PBFT [28], so old members must be replaced by new members in uncertain and random rounds. The ratio of honest nodes in the shard should be ensured according to the safety threshold. The safety threshold is the ratio of honest nodes to malicious nodes in a shard, which is defined by the consensus to determine the level of fault tolerance in a shard.
3. **Intra-Shard Consensus**: The selection of an appropriate intra-shard consensus algorithm, which in addition to a reasonable response time has less communication complexity, is another issue in blockchain-based IoT. The choice of intra-Shard consensus algorithm should be such that the minimum security guarantee for the system is 50% to avoid 51% attack.

4. Score-Based Sharding Method

In this section, we explain how the proposed nodes sharding method works in the blockchain. We clarify the score-based algorithm for node assignments to shards and then propose the intra-shard consensus protocol. Note that there is a parent shard in the system that is responsible for managing shards and storing the mini-blocks in the cloud. So, we first explain how the parent shard is constructed.

4.1. Parent shard construction

Algorithm 1 shows the process of constructing and then updating a main shard in the system named the parent shard. In this algorithm, a PoW puzzle is initially solved to identify the powerful nodes to be members of the parent shard. We assume that the attacker is a slowly adaptive adversary [27]. Then, the parent shard is reconfigured every few random rounds so that any new node that can solve the puzzle is added to the shard, and the oldest member is removed to reduce the overload.

Algorithm 1 Constructing and updating the Parent Shard

```
    Input: {Fog Nodes $N$, parent shard size $k$}
    Output: {Parent shard $P$}
$A$ = SHA256 ($Tx_{1root}$, timestamp)
$B$ = SHA256 ($Tx_{2root}$, timestamp)
 1: $i \leftarrow 1$;
 2: $|P| \leftarrow \emptyset$;
 3: $T \leftarrow N$;
 4: while $|P| \leq k$ do
 5:     if solve-crypto-puzzle($A, T$) then
 6:         $a_i \leftarrow$ solver of the puzzle;
 7:         $P \leftarrow P \cup \{a_i\}$;
 8:         $T \leftarrow T - \{a_i\}$;
 9:         $i++$;
10:     end if
11: end while
12: for each round do
13:     if solve-crypto-puzzle($B, T$) then
14:         $a_i \leftarrow$ solver of the puzzle;
15:         $P \leftarrow P \cup \{a_i\}$;
16:         $P \leftarrow P - \{a_{i-k}\}$;
17:         $T \leftarrow T - \{a_i\}$;
18:         $T \leftarrow T \cup \{a_{i-k}\}$;
19:         $i++$;
20:     end if
21: end for
```

4.2. Scoring nodes based on their activity history

By dividing the nodes into different shards based on their activity history, we aim to have a homogeneous distribution of nodes over the shards, such that the average throughput of all shards remains almost the same. This method takes the diversity of the nodes' performance into account, which would be proper for variable structure of IoT networks. We can use a non-random method based on the score of nodes to assign them to shards. Node's score refers to their history of them in the previous rounds. The history of each node can include the duration of transaction confirmation and the speed of mini-block generation by that node. In this method, the nodes are evaluated during each transaction processing and get the scores based on their performance, and the shard leader sends these scores to the parent shard for updating the configuration. The score of nodes is updated after a random number of rounds for each reconfiguration.

Let $T^i_{j,Tx} = t^i_{j(confirm)} - t^i_{(send)}$, where $t^i_{(send)}$ is when the client sends the request to the members of shard i, and $t^i_{j(confirm)}$ is when node j in shard i confirms the transaction Tx. After each transaction, we score the nodes based on the processing time and the result of their confirmation. $(score^i_j)_{Tx}$ is the score of node j in shard i after confirming transaction T_x, which is defined as follows.

$$(score^i_j)_{Tx} = \left(\frac{\overline{T^i_{Tx}}}{T^i_{j,Tx}}\right) \cdot \epsilon \tag{1}$$

where ϵ is a coefficient indicating the consistency of the result of the node transaction verification with the consensus algorithm agreed upon by the nodes. This parameter will be introduced more in the following. $\overline{T^i_{Tx}}$ is the average time over all nodes in shard i to confirm T_x and is defined as follows.

$$\overline{T^i_{Tx}} = \frac{\sum_{j=1}^{n} T^i_{j,Tx}}{n} \tag{2}$$

where n is the number of nodes in each shard. Finally, $(score^i_j)_{confirm}$ is the total score of node j in shard i after confirming t transactions, defined as follows.

$$(score^i_j)_{confirm} = \sum_{Tx=1}^{Tx=t} (score^i_j)_{Tx} \tag{3}$$

It just remains to characterize ϵ in 1. The parameter ϵ reflects the correctness or incorrectness of the transaction confirmation of the node, or its passivity. Note that some nodes in the blockchain, though being non-malicious, may confirm some transactions in some rounds but do not contribute to transaction confirmation in the other ones. These nodes are called passive nodes. For passive nodes, or for nodes whose transaction confirmation is not in line with the consensus, ϵ is set to 0.

Despite the malicious or passive nodes, an honest-active node confirms the transactions, correctly. We consider two situations for honest-active nodes. The first situation is when the node delivers the correct transaction confirmation to the client, on time. ϵ is set to its maximum value, i.e. $\epsilon = 1$, in this case. The second situation is when the node confirms the transaction correctly but delivers it to the client with some delay due to unexpected events in the IoT, such as a sudden increase in traffic. The node confirmation score will be reduced depending on the amount of delay as follows. let $d = t^i_{j(receive)} - t^i_{j(confirm)}$ be the delay of the delivering a transaction confirmation to the client, and $t^i_{j(receive)}$ is the time that the client receives the response from the node. The node confirmation score is defined as $\epsilon = \epsilon_d$ according to 4.

$$\epsilon_d = 1 - \frac{d}{Max(D)} \tag{4}$$

where $Max(D)$ is the maximum acceptable delay to deliver the transaction confirmation. For the nodes with $d > Max(D)$, ϵ will be set to zero.

All in all, the coefficient ϵ is defined as stated by 5.

$$\epsilon = \begin{cases} 0 & \text{not confirming or inconsistent confirming} \\ 1 & \text{correct confirmation and on time} \\ \epsilon_d & \text{correct confirmation with } 0 < d < Max(D) \\ 0 & \text{correct confirmation with } d > Max(D) \end{cases} \tag{5}$$

The final score of node j in shard i is obtained when it successfully generates the mini-block. This score is calculated by 6.

$$(score^i_j)_{mini-block} = (score^i_j)_{confirm} \cdot \frac{T^*}{t_{mini-block}} \tag{6}$$

where $t_{mini-block}$ is the time it takes for a node to generate a mini-block. T^* is the number of confirmed transactions to generate a mini-block for the node.This required number can be different in various rounds because the traffic patterns in IoT systems are variable. Thus, the maximum mini-block volume, which refers to the number of approved transactions to finalize the mini-block, may not be considered the same for all transactions processing rounds. However, it has a fixed value in each round. If any mini-block accepts more transactions than its capacity, it will be rejected by the network. The pseudo-code for computing the node score is summarized in Algorithm 2.

Algorithm 2 Score of node j in shard i

```
Input: { Set of shard transactions $\mathcal{T}$
       { Confirmation duration $T^i_{j,Tx}, Tx \in \mathcal{T}$
       { Average confirmation duration $\overline{T^i_{Tx}}, Tx \in \mathcal{T}$
       { Maximum number of $T_x$ to generate mini-block $T^*$
       { Duration to generate a mini-block $t_{mini-block}$
Output: $\{(score^i_j)_{mini-block}\}$
1:  for each $Tx \in \mathcal{T}$ do
2:      if $Tx_{confirm} = Tx_{Consensus}$ and $0 \leq d < Max(D)$ then
3:          $\epsilon_d \leftarrow 1 - \frac{d}{Max(D)}$
4:      else
5:          $\epsilon \leftarrow 0$
6:      end if
7:      $(score^i_j)_{Tx} \leftarrow (\frac{\overline{T^i_{Tx}}}{T^i_{j,Tx}}) \cdot \epsilon$
8:  end for
9:  $score_{confirm} \leftarrow \sum_{Tx=1}^{Tx=t} (score^i_j)_{Tx}$
10: $socre^i_{j(mini-block)} \leftarrow score_{confirm} \cdot \frac{T^*}{t_{mini-block}}$
11: return $socre^i_{j(mini-block)}$
```

4.3. Intra-shard consensus

Blockchain-based systems require a suitable consensus algorithm. A consensus with low energy consumption and communication complexity is required due to the limited computing power of IoT devices. An appropriate option for sharding-based systems is Byzantine Fault Tolerance (BFT) based algorithms. The main feature of such algorithms is the certainty of block generation. As far as block generation certainty is concerned, the main feature of these algorithms is their liveness and safety. Liveness refers to the order in which transactions are processed, and safety refers to the certainty that a block will not be destroyed. BFT protocols ideally perform in fault-free systems but their performance degrades in normal faulty scenarios. In the fault-free scenario, the replicas are honest, and there are no errors during transaction processing. However, in the normal case, some replicas may be faulty. Replicas are a set of backup nodes in a shard. As the algorithms evolve, they are designed to ensure no significant performance degradation in normal operating conditions.

One of the most common BFT protocols is Practical Byzantine Fault Tolerance (PBFT) [28], which reduced the complexity of Byzantine fault tolerance from exponential to polynomial [26], for the first time. This protocol is based on the leader that $n \leq 3f + 1$ is the number of shard nodes. f is the number of byzantine or malicious nodes. The system has a safety threshold when n follows the above relation. In PBFT, for the prepare and commit phase, replicas require two rounds of all -to -all communication to agree on executing a request that increases communication complexity between the nodes. This requires high consumption of computational resources and time. PBFT can perform best in fault-free cases. However, in normal cases, it encounters a considerable reduction in performance, which can lead to a delayed or incorrect response to the client. In blockchain-based IoT, we are looking for a consensus that, in addition to responding at the desired speed, does not have a "Performance Degradation" in normal cases. The system can trade-off between fault tolerance and performance degradation for graceful performance degradation.

DBFT [26] is a Byzantine Fault Tolerance protocol with graceful performance degradation. This algorithm assumes that the underlying cryptographic primitives, such as hash, data encryption, and signature are computationally secure. One of the main differences between DBFT and other BFT protocols is the existence of a double response to requests depending on the existing conditions, designed to achieve lower latency and higher throughput. A replica responds to the client twice. The first response is the speculative execution of the request, and the second response is the commitment to execute that request correctly by the replicas. If the client receives a number of $3f+1$ consistent responses from the replicas in the speculative execution, the request is completed and is considered the first response. If the client receives the $2f + 1$ or $3f$ first responses, the request must be completed with the $2f + 1$ second responses equal to one message delay, so the overhead caused by the double response mechanism is low. DBFT is based on three parts the activity history of replicas. The speculative processing history of requests that do not reach the quorum requires a second response to complete the request. Commitment history includes a certificate of commitment for the request. The third part contains the history of collecting additional information, the so-called garbage, which is checked in the checkpoint protocol. To achieve a consensus with low communication complexity between nodes, we designed a new consensus algorithm based on DBFT. Actually, DBFT has high communication complexity in commitment stage, which cannot be suitable for time sensitive and limited resources of IoT systems. So, in IGDBFT, we try to reduce this complexity regarding to IoT network limitations.

5. Improvement Byzantine Fault Tolerance with Graceful Performance Degradation (IGDBFT)

In designing a new consensus algorithm based on DBFT [26], we seek to reduce the communication complexity between nodes for the commitment phase in the agreement protocol. In normal cases, Algorithms 3, 4, and 5 show the performance of the client, leader, and replicas in the agreement protocol.

The other improvement on DBFT is related to one of its main problems which is the rotation of the leader per request to avoid the unpredictable destructive behavior of a leader. In the following, we discuss the problems arising from the random rotation and propose a non-random method for selecting the leader.

5.1. View change protocol and Leader selection

Replicas will initiate a view change protocol in the event of malicious behavior from the leader or the expiration of the transaction processing timeout. When the view change protocol starts, nodes cease to send and receive messages, and only messages related to changing the view are processed. BFT algorithms generally have heavy and complex view changes that can cause interruptions or loss of throughput. A long delay in processing transactions is not suitable for time-sensitive IoT systems. Moreover, as we said, in DBFT [26], a leader rotation mechanism on the existing nodes is provided per request to avoid byzantine leaders. Although this method allows the system to change to ensure the processing is safe but this rotation is random. The next leader in the rotation may be faulty, so it must be re-rotated to eventually lead to an honest node or enter the view change protocol, which means the nodes will face computational and time overhead. Leader rotation does not reduce the risk of the leader being attacked but only prevents the leader from acting maliciously. Also, due to the large number of requests in IoT, it is not cost-effective to rotate the leader for each request. Therefore, it is better to have a reliable, non-random method to select the leader in each shard to reduce the probability of changing the view. To as much as possible limit the possibility of the view change protocol, we suggest a non-random method for choosing the leader.

According to Algorithm 6, each shard has a leader to guide the intra-shard consensus after dividing the nodes according to their score. We select two nodes as leader and vice-leader, inspired by GreenPoW [29]. The drawback of GreenPoW is that it is more vulnerable to the selfish mining or collusion of nodes. In our method, the leader and vice-leader nodes are the two highest-scored nodes, respectively. Given that the node with the highest score has proven its correct activity over the rounds, it would be considered honest, so it has a very low probability of becoming byzantine, compared to other nodes. Choosing a vice-leader node that is a sure alternative option instead of a leader can avoid the costly view change protocol when the leader is compromised by a powerful attacker.

5.2. Agreement protocol

In IGDBFT, one of the $3f + 1$ replicas plays the leader's role, and the remaining $3f$ ones are backup replicas, where f is the number of faulty nodes. As shown in epoch 1 of figures 3 and 4 in fault-free and normal cases, the client sends its request to the replicas and the leader, simultaneously. The IGDBFT Agreement sub-protocol includes the following four epochs for executing a request:

1. First, the client sends $(\mathsf{request}(m, o, t, c)$ to the leader and backup replicas, where o is the operation during which the request must be processed, and t is an incremental timestamp. According to Algorithm 3, c is the client who activates a timer for a request and then sends it to all replicas.
2. Suppose the timestamp of $\mathsf{request}(m)$ is greater than the last request accepted. In that case, regarding Algorithm 4, the leader accepts m from c and assigns a sequence number n_s to request. Then, it sends a message in the form $(\mathsf{prepare}, d, v, n_s, h_{n_s})$ to the replicas. d is the digest of the request m, v is the current view in which the request for processing is ordered, and h_{n_s} is a summary of the ordered requests history. Similarly, the backup replicas accept a request from the leader if the timestamp is greater than the last timestamp of the accepted request.
3. As shown in Algorithm 5, when the replicas receive the prepared message from the leader, they confirm its integrity and validity, then accept the prepared message. The steps for checking the message confirmation are similar to [26], which are detailed, as follows:
    - The request signature is correct, and the digest d of message m is checked.
    - The request view number must match the current system view number.

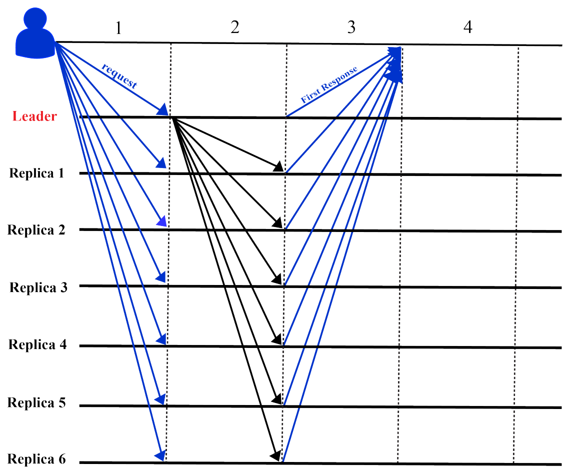


Figure 3: IGDBFT in Fault-free cases.

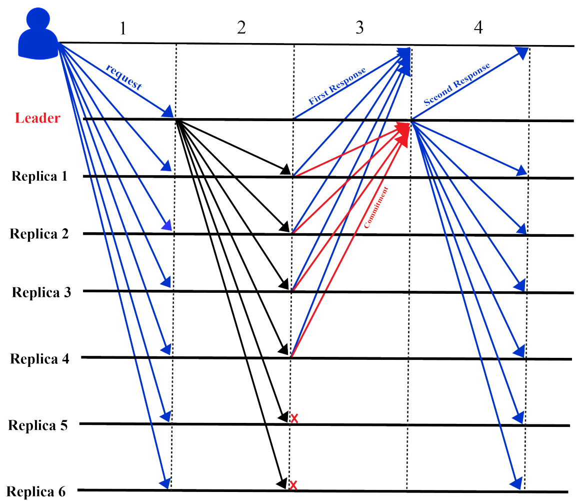


Figure 4: IGDBFT in Normal-cases.

- $n_s = n_{s_l} + 1$, where n_{s_l} is the last request accepted by a replica.
- $h_{n_s} = (h_{n_s - 1}, d)$.
- The replica does not accept two requests with the same sequence numbers and different contents in the same view.

once the prepare message is accepted, replica i puts the request in its log. Replica i executes the request with the assigned sequence number and sends message $(\mathsf{first-response}, v, s, h_{n_s}, r, t, c, i)$ to the client. r is the reply for the client. Suppose the client receives $3f+1$ first − response messages from distinct replicas with the same v, s, h_{n_s}, c, t, and r. In this case, they are compatible, so as shown in Fig. 3, the request is complete, and the client announces the completion of the request before the deadline. Suppose the client does not receive the $3f + 1$ first response after executing the request. It waits until the timer of the first response expires; refer to Algorithm 5.

4. As shown in epoch 3 of Fig. 4, in addition to sending the first response, the replicas must commit to executing the request correctly. The replicas, as stated in Algorithm 5, commit to the correct executing of the request in the form of a $(\mathsf{commit}, d, v, s, h_{n_s}, i)$ message and send it to the shard's leader.
5. After receiving a number of $2f$ compatible commitment messages from the replicas, the leader according to Algorithm 4, sends it along with his commitment to the client in a $(\mathsf{second-response}, v, d, n_s, h_{n_s}, c, l)$ message. It also sends the second − response to other replicas in the shard in epoch 4 to update their logs as shown in Fig. 4.

5.3. Checkpoint protocol

Replicas store the request history in log storage. In the IGDBFT, like DBFT [26], the replicas use the checkpoint protocol to discard additional messages in their log storage. Usually, replica i sends a checkpoint protocol by sending a $(\mathsf{checkpoint}, s_n, h_d, i)$ message after a certain number of requests. s_n is the highest sequence number relevant to the last checkpoint, and h_d is a summary of the history and current status of the network. Checkpoint protocol completes when replica i receives $2f + 1$ consistent $(\mathsf{checkpoint}, s_n, h_d, i)$ messages from other replicas.

Algorithm 3 IGDBFT Agreement Protocol for Client

```
Initialization: { Replicas $R$
                  Client $C$
                  Set of first responses $F$
                  Timestamp of last request sent by client $t_a$
                  Timer for first and second response $\delta_1, \delta_2$
                  Leader $L$
1:  for each request do
2:      $C : (\mathsf{request}, o, t, c) \longmapsto R, L$
3:      $t_a \leftarrow t$
4:      Set $\delta_1$ , $\delta_2$
5:      $R : (\mathsf{first-response}, v, s, h_{n_s}, r, t, c, i) \longmapsto C$
6:      Active $\delta_1$
7:      if $|F| = 3f + 1$ then
8:          Cancel $\delta_1$ and complete request
9:      else
10:         if $|F| = 2f + 1$ then
11:             Cancel $\delta_1$
12:             Active $\delta_2$
13:             $L : (\mathsf{second-response}, v, d, n_s, h_{n_s}, c, l) \longmapsto C$
14:              cancel $\delta_2$ and complete request
15:         end if
16:     end if
17: end for
```

Algorithm 4 IGDBFT Agreement Protocol for Leader

```
Initialization: $\begin{cases} \text{Replicas } R \\ \text{Client } C \\ \text{Timestamp of last request sent by client } t_a \end{cases}$
1: if $t > t_a$ then
2:     Leader accepts the request and assigns $n_s$ to request
3:     $L$ : (prepare, $d, v, n_s, h_{n_s}$) $\longmapsto R$
4: end if
5: In the first response phase
6: $L$ : (first − response, $v, s, h_{n_s}, r, t, c, i$) $\longmapsto C$
7: In the commitment phase
8: if $\#R = 2f$ replicas : (commit, $d, v, s, h_{n_s}, i$) $\longmapsto L$ then
9:     $L$ : (second − response, $v, d, n_s, h_{n_s}, c, l$) $\longmapsto C$
10:    Update log of $R$ with the second response
11: end if
```

Algorithm 5 IGDBFT Agreement Protocol for Replicas

```
Initialization: $\begin{cases} \text{Replicas } R \\ \text{Leader } L \\ \text{Client } C \\ \text{Timestamp of last request sent by client } t_a \\ \text{Timeouts for first and second response } \delta_1, \delta_2 \\ \text{Number of last executed request } n_a \\ \text{Number of received request } n_s \end{cases}$
1: In prepare phase
2: $C$ : (request, $o, t, c$) $\longmapsto L, R$
3: if $t > t_a$ and $n_s > n_a$ then
4:     $R$ and $L$ accept the request and assigns $n_s$ to request
5:     $L$ : (prepare, $d, v, n_s, h_{n_s}$) $\longmapsto R$
6: end if
7: In the first response phase
8: Active $\delta_1$
9: $L, \#R = 2f$ : (first − response, $v, s, h_{n_s}, r, t, c, i$) $\longmapsto C$
10: In the commitment phase
11: Cancel $\delta_1$
12: Active $\delta_2$
13: $\#R = 2f$ : (commit, $d, v, s, h_{n_s}, i$) $\longmapsto L$
14: In second response phase
15: $L$ : (second − response, $v, d, n_s, h_{n_s}, c, l$) $\longmapsto C$
16: $L \rightarrow$ Update log of $R$
17: Cancel $\delta_2$
```

6. The Lightweight-scalable Blockchain for IoT

This section describes how the proposed sharding-based blockchain works in the Internet of Things.

6.1. Assigning nodes to shards and Mini-Block generation

In the setup of the blockchain-based IoT, the nodes do not have a history of the transaction processing activity. So they can use a safe random method such as a verifiable random function (VRF) [4, 30] to divide the nodes into different shards.VRF is a Pseudo-random function with a public key that provides non-interactive proof of correctness [30, 31]. This random function is run off-chain and blue works as follows:

1. **Generate a secret key s_k and a public key p_k**:
   - Choose a random number $s \in \mathbb{Z}_p^*$ from the set of integers modulo p where p is a large prime.
   - Set the secret key to $s_k = s$ and the public key to $p_k = g^s$ where g is a generator of the group of integers modulo p.
2. **Prove** $s_k(x)$: compute the VRF output $F_{s_k}(x)$ and the proof of correctness $\tau_{s_k}(x)$ for input x and secret key s_k.
   - compute the VRF output that is $F_{s_k}(x)$ as follows. e is a bilinear pairing function; refer to [30].

$$F_{s_k}(x) = e(g, g)^{1/(x+s_k)} \tag{7}$$

   - Compute the proof of correctness as $\tau_{s_k}(x)$

$$\tau_{s_k}(x, s_k) = g^{1/(x+s_k)} \tag{8}$$

   - Output:

$$VRF_{output}(x, s_k) = (F_{s_k}(x), \tau_{s_k}(x)) \tag{9}$$

3. **Verification of the correctness of an output y and proof τ by using the p_k: $Ver_{p_k}(x, y, \tau)$.**
   - If both checks for input x pass according to 10 and 11, the output will be 1; otherwise it will be 0.

$$e(g^x \cdot p_k, \tau) = e(g, g) \tag{10}$$

$$y = e(g, \tau) \tag{11}$$

Therefore, each node computes its VRF output for a predetermined input (such as its IP address, public key, or an arbitary input) by using its secret key. The VRF outputs are used to determine that each node is assigned to which shard. The correctness of these assignments can be verified by anyone with access to the public keys and proofs of correctness.

After setup, when processing transactions is started, nodes are scored according to their activity history . At the end of a round, the shard's leader announces the scores to the parent shard. This shard sorts the scores in descending order. Parent shard assigns them to shards based on the node's number. Each shard must contain nodes with the highest and lowest scores in descending order to maintain the balance in computing power. According to the IGDBFT consensus, the transaction confirmation time to give score as shown in Equation 1 is when the client receives the first response from the replicas.

If a new node wants to be added to the shards, it has no activity history. This node can be randomly entered into one of the shards. Suppose it works appropriately after several rounds. In that case, it will be scored and used for configuration in later rounds. A node without a score does not upset the balance in the shard load.

In the reconfiguration of the subsequent rounds, it is not necessary to reshuffle all the nodes. According to the scores that the parent shard receives and the status of the shards as shown in Algorithm 6, only necessary nodes are reshuffled to maintain balance. By selecting the leader and vice leader, the leader in each shard receives the client's request and executes the transactions using IGDBFT. After processing and confirming a certain number of transactions by replicas based on their computing capacity, they inform the leader to generate the mini-block. By the leader confirming the authenticity of the mini-block, they add their mini-block to the shard chain.

6.2. Global Chain and Parent Block

As shown in Fig. 5, the mini-blocks are temporarily placed in the mini-block memory pool to maintain order. The parent shard nodes confirm the authenticity of the mini-blocks by using the collective threshold signature and its hash. When the latest node p_i, which has solved the puzzle in the parent shard, receives a new mini-block, it sends that to other nodes to check its validity. Suppose $2/3$ nodes in the parent shard send the compatible $(\mathsf{receive}(mini-block), Hash(M_{ik}))_{\sigma_j}$ message to p_i. M_{ik} is mini-block k_{th} belonging to shard i. $Hash(M_{ik})$ is the hash of the mini-block calculated by the parent shard nodes, and σ_j is the signature of node j in the parent shard on the mini-block. According to the threshold signature algorithm, p_i collects them in a final signature. Then, by the message $(\mathsf{AGREE}, Hash(M_{ik}))_{\sigma_n}$, p_i informs the nodes in the shard that it has received enough signatures. σ_n is the collective signature on the mini-block. Finally, p_i packs the mini-block in the parent block B. Refer to Algorithm 7. The size of the final signature collected by the threshold signature protocol will be about that of a single signature, so it will not have a significant storage overhead. Parent blocks are sent to the cloud layer's global chain for storage.

As displayed in Fig. 7, the parent block contains the hash of the mini-blocks, the collective signature of the agreement, the timestamp, and the hash of the previous mini-blocks to maintain integrity. The existence of a global chain in the cloud layer can, in addition to reducing the storage overload, provide access control to the information of all shards to perform cross-shard transactions. In order to perform cross-shard transactions, we need the coordination of several shards simultaneously [24, 23, 27]. The inputs and outputs in these transactions are not in a single shard. Because it is cheaper to transfer assets in a shard, we assume that the number of cross-shard transactions is limited, and their processing is the responsibility of cloud nodes. So, if the number of these transactions is high, it indicates a wrong configuration, and the nodes of the shards must be reshuffled. As a result, With the structure in Fig. 5 , we will have two types of transactions: intra-shard transactions carried out in the fog layer by shards and cross-shard transactions carried out by the nodes in the cloud layer.

Algorithm 6 Node assignment to the shards

```
Input: { Scores of $N$ nodes : $S$ }
Output: $\begin{cases} \textbf{shards } C = \{c_1, c_2, \dots, c_i\} \\ \textbf{leaders } L = \{l_1, l_2, \dots, l_i\} \\ \textbf{vice leaders } V = \{v_1, v_2, \dots, v_i\} \end{cases}$
1: $S_h \leftarrow$ the set of $i$ highest scores of $S$
2: $S_v \leftarrow$ the set of scores of $i$ selected vice leaders
3: $S_o \leftarrow S - S_h \cup S_v$
4: Round $r$:
5: $S \leftarrow$ sort $S$ in descending order
6: $S_h \leftarrow$ top $i$ elements of $S$
7: $L \leftarrow S_h$
8: Assign each $l_j \in L$ to a unique shard $c_j \in C$
9: for $j = 1$ to $i$ do
10:     $v_j \leftarrow$ select the next highest scoring node from $S_o$
11:     $S_v \leftarrow S_v \cup \{v_j\}$
12:     Assign $v_j$ to the same shard as its leader $l_j$
13:     $S_o \leftarrow S_o \setminus \{v_j\}$
14: end for
15: $S_o \leftarrow$ sort $S_o$ in descending order
16: Assign each remaining node in $S_o$ to shards in descend-
    ing order
17: Round $r + k$:
18: $S \leftarrow$ Sort $S$ in descending order
19: function FINDLEADER($S$, $i$)
20:     Let $S_h$ be the top $i$ elements of $S$
21:     return $L \leftarrow S_h$
22: end function
23: $c_{min} \leftarrow$ Shards that have the least total scores
24: Assign proper $l_j \in L$ to $c_{min}$ (reshuffle)
25: function FINDVICELEADER($S$, $L$, $c_j$)
26:     Let $v_j$ be the next highest scoring node in $S_o$
27:     Let $S_o$ be the set of scores of nodes not in $L$ or $V$
28:     $S_v \leftarrow S_v \cup \{v_j\}$
29:     return $v_j$
30: end function
31: Assign $v_j$ to $c_j$
32: $S \leftarrow S_h \cup S_v \cup S_o$ ▷ update $S$ with the new assignment
33: Round $r + k + 1$ ;
```

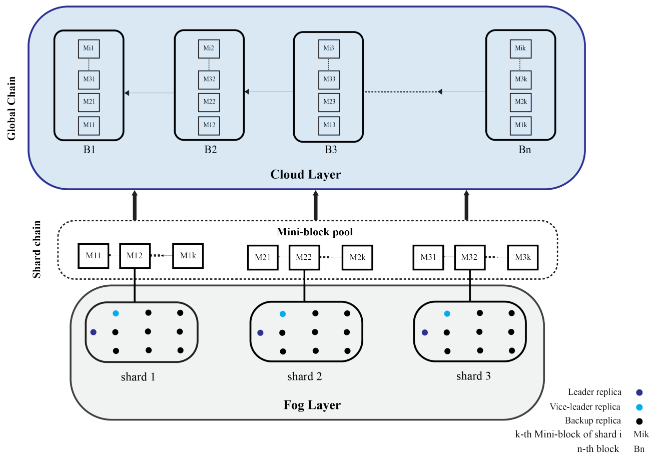


Figure 5: The scheme of global chain and parent block.

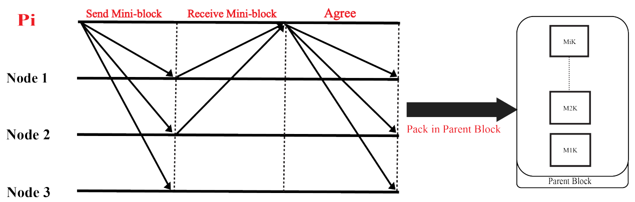


Figure 6: The steps of Adding the Mini-Block in the parent block B.

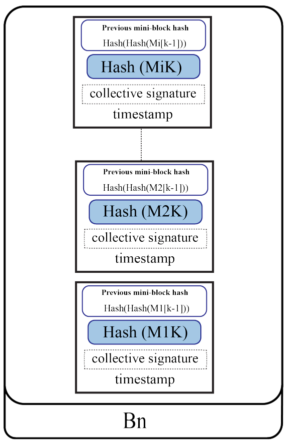


Figure 7: Parent block structure

So, to overcome conflicting transactions between shards, IGD-ScoreChain uses the IGDBFT consensus algorithm to reach an agreement between nodes on which transaction to include in the global chain. Also, the Parent shard stores mini-blocks in the cloud layer using a collective threshold signature, ensuring that the global chain is tamper-proof and immutable. As shown in Algorithm 7, 5 and Fig. 6, the parent shard nodes confirm the authenticity of mini-blocks and collect a final signature to pack the mini-block in the Parent Block. This approach helps prevent conflicts and ensures that the global chain is secure and reliable for executing transactions. .

Algorithm 7 Generating parent block

```
     Input      {mini-blocks $M_{ik}$}
     Output     {Parent block $B$}
 1:  Insert mini-blocks in the memory pool to maintain order
 2:  Let $p_i$ be the current parent shard node.
 3:  $P \leftarrow \{p_1, p_2, \dots, p_j, \dots, p_{i-1}\}$
 4:  private key of $p_j$ , $p_i \leftarrow s_{k_j}, s_{k_i}$
 5:  $p_i$ receives $M_{ik}$
 6:  $p_i$ send $M_{ik}$ to $P$
 7:  if $p_j$ receives $M_{ik}$ then
 8:      $h \leftarrow \text{Hash}(M_{ik})$
 9:      $p_j$ : $(\text{receivemini}-\text{block}, h)_{\sigma_j} \longmapsto p_i$
10:  end if
11:  for each $P \in \{p_1, p_2, \dots, p_{i-1}\}$ do
12:      if $\#(\text{receivemini}-\text{block}, h)_{\sigma_j} \geq \lceil \frac{2}{3}(i-1) \rceil$ then
13:          Let $\sigma_1, \sigma_2, \dots, \sigma_{i-1}$ be the valid $\sigma$
14:          if valid $\sigma \longmapsto p_i$ then
15:              $(\sigma_1, \sigma_2, \dots, \sigma_{i-1})_{s_{k_i}} \rightarrow TSS$
16:              $\sigma_n \leftarrow \text{TSS}$
17:          end if
18:          Return $(\text{AGREE}, h, \sigma_n) \longmapsto P$
19:      end if
20:  end for
21:  $p_i$ Add $(\text{Hash}(\text{Hash}(M_{i(k-1)})), h, \sigma_n, t)$ to B
```

7. Security analysis

It is crucial that blockchain is secure for its users. Three types of attacks are possible in these systems. The first one involves mathematical techniques used in blockchains, such as forking. The second one is the category of attacks related to the architecture of the peer-to-peer network, like the 51% , selfish mining, Sybil, and DDoS attack. The third one is the attacks on the blockchain platform, like replay attacks and double-spending attacks [32].

7.1. Join-leave Attack

Our approach uses scored nodes to address shard reconfiguration issues like Sybil and join-leave attacks. An attacker may be trying to take over a shard completely with random methods. For example, in a join-leave attack [23], the attacker enters the shard randomly to learn the structure of the shard, and if it fails, exits and tries again to enter the shard structure and take control of its affairs. By dividing the nodes into shards based on their scores, the possibility of the attacker succeeding without having an activity history will be decreased dramatically.

In the Sybil attack, the attacker, as a fake node tries to take over the shard by repeatedly joining and leaving the shard randomly. In this attack [25], the attacker disrupts the operation of the network using multiple fake identities. In random sharding methods, the possibility of this attack is high. However, by using the scoring method, penetrating a malicious node into a shard is avoided, and by controlling the shard's reconfiguration by the parent shard, the number of faulty nodes in a shard cannot be more than the safety threshold.

As we said, if a new node wants to join the shard and has no activity history, it can randomly enter one of them. This node can gain some score if it works according to the agreed consensus of the shard. If this node is malicious, it will not necessarily perform according to the common goal of the shard, and as a result, it will be given a zero score and recognized as malicious. If the number of these nodes exceeds the safety threshold during the rounds, the shard leader will remove it from the shard to maintain balance.

7.2. 1% Attack

A puzzle that divides the dataset into shards will expose it to a 51% attack in the PoW-based sharding method. The network can be stabilized by taking control of 50% of the hash rate, and it can be used to mount an attack. A 51% attack can provide long delays, selfish mining, DDoS attacks, or invalidating blocks. An attacker may assign malicious nodes to a shard through PoW to collect and control all malicious nodes within the shard. It takes only 51% of an attacker's total computing power to take over a shard when they use all of their computing power. As long as m is large enough, 51% is almost reduced to 1%. There are m shards in the database. Consequently, an attacker needs only $\frac{51}{m} \simeq 1\%$ of its computing power to take control of a shard. In large blockchain networks such as Bitcoin, 51% attacks are usually not cost-effective due to the enormous computing power required and the randomness of whether an attacker can solve the computing puzzle. It is usually more common in smaller sharding-based blockchains than in larger networks with powerful attackers. When nodes are assigned to shards at random, it is possible for an attacker with high computing power to penetrate one of the shards and control its affairs. This is called the 1% attack [27]. In our method, PoW will only be used to select parent committee nodes to evaluate the computational power of the nodes by solving a cryptographic puzzle. It will not include block generation, so the system is not exposed to the fork and 51% attack. Using this approach, the shards are guided by the nodes capable of solving the puzzle.

7.3. Selfish mining and Block losing

In the general review of the comparison of BFT with Nakamoto consensus algorithms, BFTs have certainty in their results, and there will be no possibility of losing a block. In these algorithms, the network can tolerate a certain number of faulty nodes without disrupting the performance. In Nakamoto-based algorithms, the valid block may be removed from the chain in two ways; The first case is the stale block that has been successfully mined but is not accepted in the current best chain. Selfish mining [25] is an attack where a node mines a new block but does not publish it to other nodes to be added to the end of the chain. The mining node creates a private chain in the existing public chain and, after reaching a longer chain than the existing public chain, publishes its chain and leads to a fork. In this way, the attacker produces a stale block and deprives the honest node of the reward for its honest activity.

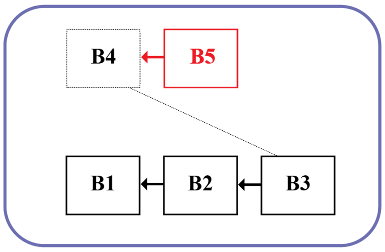


Figure 8: General overview of orphan blocks.

The second form of block loss in Nakamoto is the orphan block [25]. As shown in Fig. 8, block 5 refers to invalid block 4, which is separated from the chain and cannot be verified. This inconsistency can be caused by an attacker's operation or a race condition to solve the puzzle among the nodes. Therefore, using byzantine fault tolerance, in addition to creating safety in the production of the block in a definitive manner and not destroying it, eliminates the possibility of attacks such as selfish mining. Safety means that if the nodes complete request n with a history of h_n (history of ordered requests) and each request is ordered with a higher sequence number h_k, it will contain the history of h_k in which h_n is prefixed.

7.4. DDoS Attack

BFTs will also resist DDoS attacks that lead to forks, delays, and injecting fake blocks to prevent consensus [25]. In injecting fake blocks, the process allows malicious nodes to confuse the system by wasting time because accepting or rejecting them can be time-consuming. This issue will be problematic for time-sensitive IoT systems. In a DDoS, the goal is not to destroy the target service but to make the target server unable to provide service. These attacks are carried out by sending data packets to the target, which overwhelms the network's processing capacity by them and prevents users from accessing the service [27]. For example, a 51% attack on Bitcoin can lead to this attack. In particular, if a group of nodes acquires significant hash power, they can prevent adding mined blocks to the chain or invalidate current transactions. Also, this attack is due to the limited and fixed number of transactions that each block can process. For example, it takes 10 minutes for the Bitcoin network to generate a block whose maximum size is 1 MB. We proposed that the number of transactions per mini-block or mini-block size in different rounds could change depending on the network traffic pattern due to the IoT's dynamic structure but this size should not exceed the global capacity.

The IGDBFT is not client-oriented, unlike hBFT [33], in which, in addition to sending and receiving messages, the client also directs some consensus sub-protocols. In IGDBFT, the client is only responsible for sending and receiving messages. In a scenario such as DDoS, if a faulty client tries to send messages with the same or malicious content at the same time and beyond the capacity of the node, or if valid data transmission is repeated fraudulently, the system will be exposed to a replay attack [25]. In the replay attack, the attacker tries to disrupt the system by sending repeated messages continuously. In the structure of the IGDBFT, to accept the request, the leader must accept a request with an incremental timestamp, and the simultaneous requests with different content or from different verified users will be batched. As a result, the faulty client will be disabled and unable to disrupt the system. This incremental timestamp will also prevent double-spending attacks [25]. The attacker uses the input twice simultaneously in this attack. So in IGDBFT, replicas do not act on client feedback and use pre-defined processing, reducing the impact of a faulty client.

7.5. Leader corruption

One of the notable cases in BFTs security is the corruption of the leader by the attacker or a byzantine operation of the leader during rounds. In our method, the leader is the node with the highest score, unlike the conventional methods that choose the leader randomly or through rotation, such as [24, 26, 33, 34]. The high score of a node in the shard shows the node's excellent activity, honesty, and strength during the processing transactions. On the other hand, no attacker would take the risk of attacking a strong node. As a result, the possibility of attacking a leader that has proven its superior performance in processing power is low unless the attacker's computing power is much higher than the leader's. Also, sending the client's request to other replicas besides the leader at the beginning will quickly detect the malicious behavior of the leader, such as sending the request with delay or changing the content of the message by the replicas.

8. Performance Analysis

In this section, we discuss different aspects of the efficiency and performance of the proposed IGDBFT and compare it with the other schemes.

8.1. Efficiency analysis of Score-Based sharding

An important aspect of sharding is that dividing the nodes into shards should balance the computational load and operational throughput across all shards. For example, the sharding algorithm should not assign the high-power nodes to a certain number of shards, and weak nodes to others.In the random sharding methods, it is more likely that powerful nodes will be placed in specific shards, and the rest of the shards will include nodes that do not have powerful nodes. so, it would lead to outbalancing in aspect of throughput among shards. Due to this imbalancement, transactions are not coordinated between the shards. So, there are consequently delays, resulting in reduced throughput. However, in the proposed sharding method, all delays in responding to clients, even partial, have been taken into account for calculating the scores to provide a more accurate load balance across the shards. As a consequence, each shard contains both strong and weak nodes. All shards are functionally balanced, with nearly uniform computation and time overhead. This will result in an improvement in the system's throughput.

8.2. Efficiency analysis of IGDBFT

Due to the high resource consumption of Nakamoto-based consensus algorithms like PoW or PoS, are usually not recommended for IoT with limited resources. Unlike Nakamoto's algorithms, byzantine fault tolerance consensuses have a better speed for reaching consensus and do not require high energy consumption and computing power. It is essential to consider the complexity of communication between nodes to choose the appropriate BFT for IoT.

In IGDBFT terms of cost, which refers to the number of cryptographic operations used in the algorithm, is in the same category as DBFT, PBFT [28], and HBFT [33] algorithms ; can offer good flexibility against server failures with $3f+1$ replica to tolerate f faulty node . As we can see in Table 1, the communication complexity in IGDBFT compared to all three previous cases has been reduced from $O(n^2)$ to $O(n)$. IGDBFT and DBFT have four one-way message delays, while PBFT requires five one-way message delays. The one-way latency of HBFT is less than IGDBFT and DBFT. However, one of the main problems of this protocol is being client-centric in some sub-protocols [33]. The goal of IGDBFT is to reach a consensus with graceful performance degradation in the face of unpredictable events so that this reduction does not damage the network performance.

DBFT, as we mentioned, rotates the leader per request, which may cause interruptions and time overhead in the system's performance. Our method selects the two nodes with the highest scores as the leader and vice leader for IGDBFT, respectively, with a high level of computing power.

Consequently, if the leader is corrupted by a powerful attacker, the vice leader will be immediately replaced, and the system will continue to operate as usual. As a result of this approach, it is possible to reduce the probability of the view change protocol and allow the network to continue operating uninterrupted.

8.3. Comparing the number of communication messages in IGDBFT with previous BFTs

The number of IGDBFT communication messages in the number of faulty nodes is calculated as follows:

- Epoch 1: the client sends the request for all replicas: $3f + 1$ message.
- Epoch 2: leader forwards request to $3f$ replicas : $3f$ message.
- Epoch 3: replicas send $2f + 1$ first response to client and $2f$ commit message to leader : $(2f + 1) + 2f$ message.
- Epoch 4: leader sends Second response to client and $3f$ replicas: $1 + 3f$

So, the total communication messages in IGDBFT would be

$$(3f + 1) + (3f) + (2f + 1) + 2f + (1 + 3f) = 13f + 3$$

Table 1
Comparing IGDBFT with previous BFTs.

	PBFT	**HBFT**	**DBFT**	**IGDBFT**
Cost	3f+1	3f+1	3f+1	3f+1
Latency (one-way)	5	3	4	4
Communication complexity	O(n^2)	O(n^2)	O(n^2)	O(n)
Reference	[28]	[33]	[26]	This paper

Table 2
Consensus Complexities

Protocol	**Resiliency**	**Complexity**	**Sharding method**	**Reference**
Elastico	1/4	$O(n^2 + N)$	Random	[19]
Omniledger	1/4	$O(n^2 + N)$	Random	[20]
Rapidchain	1/3	$O(n^2 + n\log(N))$	Simple- active/inactive	[23]
ZyConchain	1/3	$O(n\log(N) + \log(n) + n)$	Random	[24]
Repchain	1/3	$O(n^2 + N)$	Based on Reputation	[35]
IGD-ScoreChain	1/3	$O(n\log(N) + n)$	Based on Score	This paper

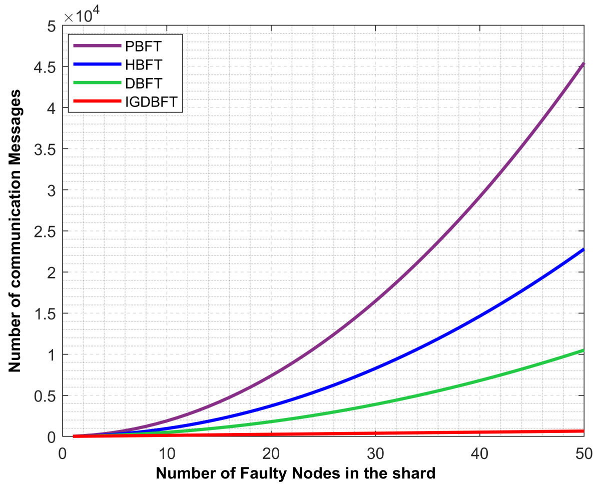


Figure 9: Comparing the number of communication messages in IGDBFT with previous BFTs with N = 1000.

which is linear in f. This parameter for the other BFT protocols are as follows:

- PBFT [28]:
  $1+3f+3f\cdot 3f+(1+3f)\cdot 3f+3f+1 = 18f^2+9f+2$
- HBFT [33]:
  $1+3f+3f\cdot 3f+1+3f = 9f^2+6f+2$
- DBFT [26]:
  $3f+1+3f+2f+1+2f\cdot 2f+2f+1 = 4f^2+10f+3$

all of which are quadratic in f. A schematic comparison of the number of communication messages for the BFT algorithms in terms of the number of faulty nodes is given in Fig. 9. In IGDBFT, the growth rate of the number of communication messages is significantly lower than that of the other three ones, which is in favor of the throughput of this protocol.

8.4. Total communication overhead per transaction process

We calculate the consensus and communication complexity of the system for each T_x, which finally adds to the parent block. It is supposed that the network has C shards that $C = N/n$. N is the total number of blockchain network nodes and n is the size of each shard. We assume that the number of shards is fixed in the rounds. The number of network nodes is dynamic and up to 1000. First, the user sends the T_x to a fixed number of client nodes at the edge of the IoT, which will send it to the desired shard, which has a $n\log(N/n) = O(n\log(N))$ communication overhead. Once T_x is sent to the shardhas been received by the shard, the computational overhead of running IGDBFT intra-shard consensus would be of $O(n)$. Finally, according to Algorithm 7, a mini block is added to the parent block, which has a complexity of $O(n)$. So, the total communication complexity is $O(n\log(N)) + O(n) + O(n) = O(n\log(N) + n)$.

In Table 2, we describe a summary of the evaluation of the communication complexity of our scheme and other previous methods based on sharding . We can also see this comparison in Figs. 10 and 11.

As shown in Fig. 10, we can see that the proposed method has a significant improvement in complexity and communication overhead compared to Elastico [19], Omniledger [20], Rapidchian [23], and Repchain [35]. As shown in Fig. 11, the communication complexity of our protocol is improved with a slight difference compared to Zyconchain [24]. In general, by increasing the network nodes, IGD-ScoreChain performs better in communication complexity, which can improve the system's performance.

8.5. Storage Complexity

The nodes of shards in the fog layer keep only the necessary information of the ledger, and the checkpoint protocol removes the additional information. Let the size of the ledger be $|L|$, then each node in the shard stores an

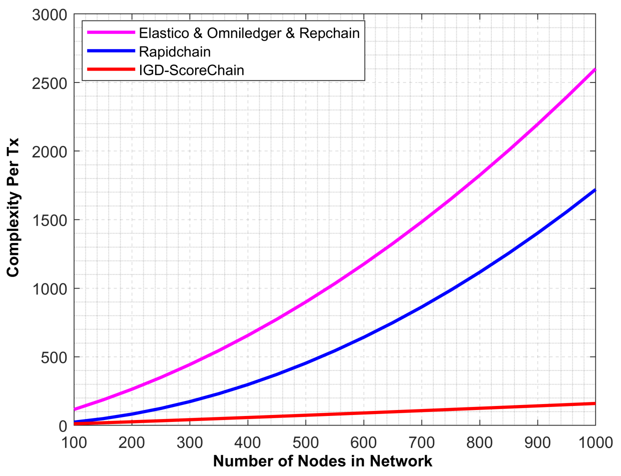
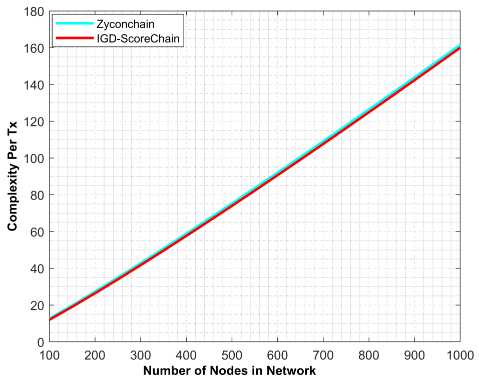
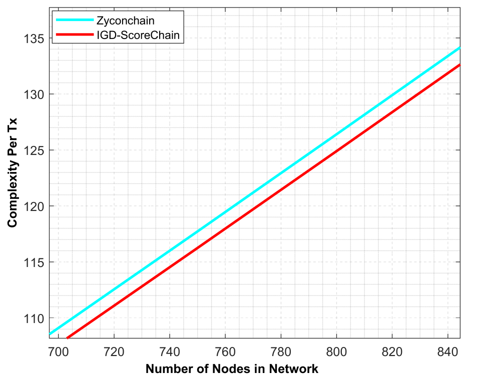


Figure 10: Comparison of the total communication complexity overhead per transaction



(a) Comparison of IGDBFT total communication complexity overhead per transaction with Zyconchain.



(b) Close up of Comparison of IGDBFT total communication complexity overhead per transaction with Zyconchain.

Figure 11: Comparison of IGDBFT total communication complexity overhead per transaction with Zyconchain.

amount of data of order $O(\frac{|L|}{C})$. The mini-block memory pool temporarily stores mini-blocks. In one mining round, the maximum number of mini-blocks held by this pool is $k \cdot C = k \cdot \frac{N}{n}$, where k is the number of mini-blocks produced by each shard.

We divided the ultimate storage layer into the computation and consensus layers. The cloud layer stores the parent shards located at global chain. The nodes of the cloud layer have access to all the information of the blocks that include the fog layer nodes' performance history. Because mini-blocks place temporarily in the memory pool, we can ignore their storage complexity. Therefore, the storage complexity in the fog layer is $O(\frac{|L|}{C})$.

9. Conclusions

This paper proposes a novel scalable blockchain for IoT. A new sharding method has been developed to balance the operational capacity of the system based on the performance of nodes during transaction processing. As a result of system problems and errors, the proposed IGDBFT algorithm will ensure graceful performance. In normal circumstances, there will be no significant reduction in network performance.

As part of this development, a new block storage method has been added to the cloud layer as a parent block. Therefore, we reduce the high computational load related to cross-shard transactions in the fog layer through routing algorithms by entrusting the processing of these transactions to cloud nodes. As a result of analyzing the security and efficiency of the proposed scheme, we concluded that the blockchain in our scheme would resist various common attacks. Thus, the possibility of system efficiency disruption will be reduced. Also, by analyzing the complexity of communication overhead and storage overheads, we showed that the performance of our protocol is significantly better than the other ones, as the number of nodes grows.

References

[1] Muhammad Salek Ali, Massimo Vecchio, Miguel Pincheira, Koustabh Dolui, Fabio Antonelli, and Mubashir Husain Rehmani. Applications of blockchains in the internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 21(2):1676–1717, 2018.

[2] Kai Lei, Maoyu Du, Jiyue Huang, and Tong Jin. Groupchain: Towards a scalable public blockchain in fog computing of iot services computing. *IEEE Transactions on Services Computing*, 13(2):252–262, 2020.

[3] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Ieee Access*, 4:2292–2303, 2016.

[4] Mingxiao Du, Qijun Chen, and Xiaofeng Ma. Mbft: A new consensus algorithm for consortium blockchain. *IEEE Access*, 8:87665–87675, 2020.

[5] Ming Li, Jian Weng, Anjia Yang, Wei Lu, Yue Zhang, Lin Hou, Jia-Nan Liu, Yang Xiang, and Robert H Deng. Crowdbc: A blockchain-based decentralized framework for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, 30(6):1251–1266, 2018.

[6] Jianjun Sun, Jiaqi Yan, and Kem ZK Zhang. Blockchain-based sharing services: What blockchain technology can contribute to smart cities. *Financial Innovation*, 2(1):1–9, 2016.

[7] Xiao Yue, Huiju Wang, Dawei Jin, Mingqiang Li, and Wei Jiang. Healthcare data gateways: found healthcare intelligence on

blockchain with novel privacy risk control. *Journal of medical systems*, 40(10):1–8, 2016.

[8] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with iot. challenges and opportunities. *Future generation computer systems*, 88:173–190, 2018.

[9] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Ieee Access*, 4:2292–2303, 2016.

[10] Lucas de Camargo Silva, Mayra Samaniego, and Ralph Deters. Iot and blockchain for smart locks. In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0262–0269. IEEE, 2019.

[11] Pietro Danzi, Anders Ellersgaard Kalor, Cedomir Stefanovic, and Petar Popovski. Analysis of the communication traffic for blockchain synchronization of iot devices. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[12] Pietro Danzi, Anders E Kalør, Čedomir Stefanović, and Petar Popovski. Delay and communication tradeoffs for blockchain systems with lightweight iot clients. *IEEE Internet of Things Journal*, 6(2): 2354–2365, 2019.

[13] Seyoung Huh, Sangrae Cho, and Soohyung Kim. Managing iot devices using blockchain platform. In *2017 19th international conference on advanced communication technology (ICACT)*, pages 464–467. IEEE, 2017.

[14] Jianli Pan, Jianyu Wang, Austin Hester, Ismail Alqerm, Yuanni Liu, and Ying Zhao. Edgechain: An edge-iot framework and prototype based on blockchain and smart contracts. *IEEE Internet of Things Journal*, 6(3):4719–4732, 2018.

[15] Guilherme Vieira Pinto, João Pedro Dias, and Hugo Sereno Ferreira. Blockchain-based pki for crowdsourced iot sensor information. In *International conference on soft computing and pattern recognition*, pages 248–257. Springer, 2018.

[16] Shi-Cho Cha, Jyun-Fu Chen, Chunhua Su, and Kuo-Hui Yeh. A blockchain connected gateway for ble-based devices in the internet of things. *ieee access*, 6:24639–24649, 2018.

[17] Yu Zhang and Jiangtao Wen. An iot electric business model based on the protocol of bitcoin. In *2015 18th international conference on intelligence in next generation networks*, pages 184–191. IEEE, 2015.

[18] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. {Bitcoin-NG}: A scalable blockchain protocol. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, pages 45–59, 2016.

[19] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 17–30, 2016.

[20] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[21] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. Lsb: A lightweight scalable blockchain for iot security and anonymity. *Journal of Parallel and Distributed Computing*, 134:180–197, 2019.

[22] Zijian Bao, Wenbo Shi, Debiao He, and Kim-Kwang Raymond Chood. Iotchain: A three-tier blockchain-based iot security architecture. *arXiv preprint arXiv:1806.02008*, 2018.

[23] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 931–948, 2018.

[24] Nasrin Sohrabi and Zahir Tari. Zyconchain: A scalable blockchain for general applications. *IEEE Access*, 8:158893–158910, 2020.

[25] Sachin Shetty, Charles A Kamhoua, and Laurent L Njilla. *Blockchain for distributed systems security*. John Wiley & Sons, 2019.

[26] Jingjing Zhang, Yingyao Rong, Jiannong Cao, Chunming Rong, Jing Bian, and Weigang Wu. Dbft: A byzantine fault tolerance protocol with graceful performance degradation. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[27] Yizhong Liu, Jianwei Liu, Marcos Antonio Vaz Salles, Zongyang Zhang, Tong Li, Bin Hu, Fritz Henglein, and Rongxing Lu. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *Computer Science Review*, 46:100513, 2022.

[28] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[29] Noureddine Lasla, Lina Al-Sahan, Mohamed Abdallah, and Mohamed Younis. Green-pow: An energy-efficient blockchain proof-of-work consensus algorithm. *Computer Networks*, 214:109118, 2022.

[30] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[31] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[32] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. A survey on the security of blockchain systems. *Future Generation Computer Systems*, 107:841–853, 2020.

[33] Sisi Duan, Sean Peisert, and Karl N Levitt. hbft: speculative byzantine fault tolerance with minimum cost. *IEEE Transactions on Dependable and Secure Computing*, 12(1):58–70, 2014.

[34] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 45–58, 2007.

[35] Chenyu Huang, Zeyu Wang, Huangxun Chen, Qiwei Hu, Qian Zhang, Wei Wang, and Xia Guan. Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding. *IEEE Internet of Things Journal*, 8(6):4291–4304, 2021.