Group BFT: Two-Round BFT Protocols via Replica Grouping

Xuyang Liu, Zijian Zhang*, *Senior Member, IEEE,* Zhen Li, Xin Lu, Meng Li*, *Senior Member, IEEE,* Lei Xu, *Member, IEEE,* Meng Ao, Liehuang Zhu, *Senior Member, IEEE,*

Abstract—This paper seeks to enhance the performance of large-scale leader-based Byzantine Fault Tolerant (BFT) systems by proposing a novel Group BFT scheme. The scheme utilizes a two-round message transmission process to distribute the load from a single leader across multiple replicas by dividing the entire consensus network into groups, each with an equal number of replicas. Each group has a leader to process the group's voting messages into a single aggregated voting message during the first round, which is then transmitted to the consensus leader in the second round (similar process for proposing). We establish a formal system framework for Group BFT protocols with a versatile set of base components and explicit definitions, addressing the challenges inherent in designing such a system. We further design and implement two highly efficient Group BFT protocols: one that supports inter-group member exchange and the other one that does not. We theoretically prove the safety, liveness, and responsiveness of the Group BFT protocols. We conduct a formal analysis of Group BFTs' tolerance and complexity. Experimental results show that the two Group BFT protocols significantly alleviate the processing bottlenecks of the leader and highly improve throughput in large-scale systems.

Index Terms—Distributed Ledger, Large-scale BFT System, Byzantine Fault Tolerance, Consensus, Grouping.

1 Introduction

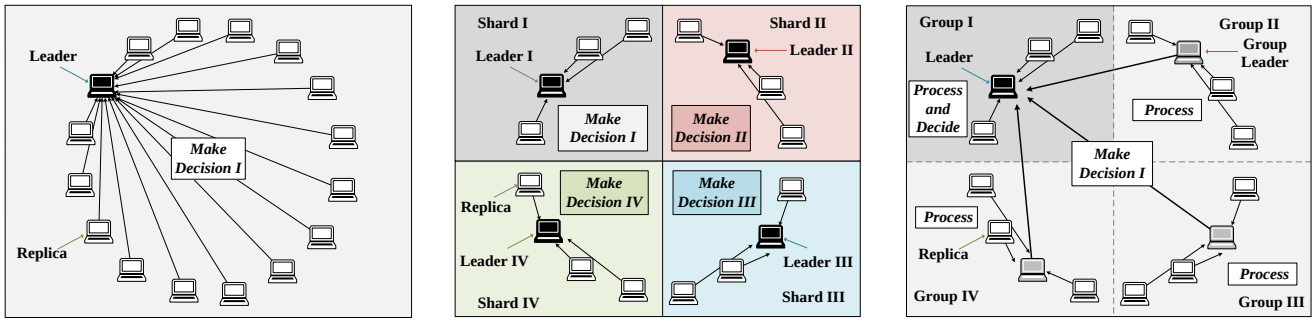
Lamport et al. introduced the Byzantine Generals Problem [1], a distributed network fault tolerance issue, in 1982. This problem emerges in a distributed computing system where inconsistencies can occur due to erroneous or misleading information from member computers or the network itself. Lamport's centralized solutions, however, lacked efficiency and feasibility. Castro and Liskov's Practical Byzantine Fault Tolerance (PBFT) consensus algorithm [2] reduced time complexity to $O(n^2)$, enhancing its practicality in distributed systems. Nonetheless, PBFT's high communication complexity spurred further advancements. Various consensus protocols [3], [4], [5], [6], [7], [8] aimed at boosting Byzantine Fault Tolerant (BFT) systems' performance were proposed.

Yet, existing works barely address the throughput degradation in large-scale BFT systems due to the immense communication overhead from excessive replica numbers. The mainstream approach in modern BFT protocol design utilizes leader-based approaches, where a designated replica coordinates each round of consensus. In these systems, the consensus flow (the exact steps may vary depending on the specific protocol) typically starts with a 'Leader Election' (which can be random [9], [10], round-robin [5], [11], or other possible forms). Then the leader proposes values, replicas vote on them, and decisions are made based on supermajority (typically more than $2/3$) agreement. If the supermajority is achieved, the network commits to the value and subsequently moves to the next consensus round. If not achieved, the network aborts and elects a new leader to restart the flow. This design offers significant advantages due to presence of a coordinating leader: reduced communication complexity; simplified protocol design, and increased efficiency. Despite having coordinators, these systems remain decentralized because leadership rotates among participants, malicious behavior is tolerated even from leaders, and decisions still require supermajority agreement.

The leader in leader-based BFT system bears the entire message load of the consensus network. Such a load increases in larger systems, causing the systems more prone to network congestion. Excessive message concentration can easily reach a bottleneck in the leader's message processing, leading to a drop in throughput. Despite several Directed Acyclic Graph [12] (DAG) schemes [9], [10], [13] proposed to alleviate this bottleneck, these solutions require abstracting the network communication layer from consensus, and also at the cost of additional memory consumption that gradually accumulates as consensus progresses. These issues raise the question: how can large-scale BFT system performance be improved without introducing graph structures?

Grouping approach. One potential solution is to distribute the load from a single leader across multiple replicas. This can be achieved by dividing the entire consensus network into groups, each containing an equal number of replicas. A suitable leader could be elected within each group to process the group's voting information. Then, each group leader would transmit the consolidated voting information

- *Xuyang Liu, is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, China; School of Computer Science, the University of Auckland, New Zealand (Email: liuxuyang@bit.edu.cn).*
- *Zijian Zhang, Xin Lu, Lei, Xu, Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, China (Email: {zhangzijian, luxin, xu.lei, liehuangz}@bit.edu.cn).*
- *Zhen Li is with the School of Computer Science and Technology, Beijing Institute of Technology, China (Email: zhen.li@bit.edu.cn).*
- *Meng Ao is with the Tencent Cloud (Email: mengao@tencent.com).*
- *Meng Li is with the Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education; School of Computer Science and Information Engineering, Hefei University of Technology; Intelligent Interconnected Systems Laboratory of Anhui Province (Hefei University of Technology) and HIT Center, University of Padua, Italy. (Email: mengli@hfut.edu.cn)*
- *Corresponding Author: Zijian Zhang, Meng Li.*



(a) Common approach (b) Sharding approach (c) Grouping approach

Fig. 1. A comparison between the common approach, the sharding approach and the grouping approach when voting.

to the consensus leader (similar process for proposing). This approach, while seemingly bearing resemblance to sharding [14], [15], holds a key distinction. Sharding divides the entire network into smaller parts (or "shards"), each capable of processing decisions independently; shards are relatively autonomous. In contrast, grouping divides the entire network into groups, but all groups work together to process the same decision, maintaining a close interlinkage. The differences between the common approach, the sharding approach, and the grouping approach are visually illustrated in Figure 1.

To better illustrate these approaches, consider a concrete example where there are some pending transactions in a network of 16 replicas: In the common approach [2], [5] (Figure 1(a)), when processing a transaction batch, a single leader receives all vote messages from every replica in the system. With 16 replicas, the leader must process 16 individual votes, creating a potential bottleneck. In the sharding approach [14], [15] (Figure 1(b)), the network might be divided into 4 shards of 4 replicas each. Each shard processes different transactions independently - for instance, Shard 1 might handle transactions for accounts starting with 'A', while Shard 2 handles accounts starting with 'B'. Each shard has its own leader and consensus process (which can also be viewed as using common approach within 4 shards concurrently), but shards must coordinate for cross-shard transactions (e.g., transactions from 'Ax' to 'Bx'). In our grouping approach (Figure 1(c)), replicas are organized into 4 groups of 4 replicas each, all groups collectively process the same transaction batch. Each group internally processes 4 votes and aggregates a single group vote through its group leader. The consensus leader then only needs to process 4 group votes instead of 16 individual votes, while still involving all replicas in the same decision.

Another concept akin to grouping is the use of communication trees [16] in BFT protocols [17], [18], [19], which also addresses the scalability limitations in large-scale BFT systems. The grouping approach can be viewed as a communication tree with exactly two levels, where each internal node (group leader) manages an identical number of leaf nodes. This balanced structure enables all group leaders to operate under equivalent parameters and principles, maintaining a unique position between the consensus leader and normal replicas. This uniform distribution provides simplicity and helps to further reduce communication overheads.

Technical challenges. Implementing the grouping approach, however, presents two significant challenges. Firstly, as inter-group communication exclusively occurs among group leaders, the group must function as a cohesive entity. In doing so, this facilitates the effective transmition of the group's consolidated voting messages via peer-to-peer communication between replicas. If we consider the process of sending votes from replicas to the group leader as the first round of voting, then the transmission of voting messages from the group leader to the consensus leader constitutes the second round of voting. A viable scheme must ensure that the second round fully embodies the voting functions of the first round. The second challenge in Group BFT is ensuring that all replicas within a group have equal access to the group votes. As inter-group communication occurs solely between the leaders of each group per round, there is a risk that non-leader members may not receive the aggregated group vote messages in a timely manner. The scheme should enable each replica to receive group votes promptly for convenient verification.

Our contributions. In this paper, we propose a novel Group BFT scheme to tackle with these challenges. The scheme involves a two-round message transmission process: one round within the group and the other one between group leaders and the consensus leader. We conduct an in-depth analysis of this specific two-level tree structure and demonstrate how it offers unique advantages. We illustrate the feasibility of this hierarchical communication process and demonstrate that each group can indeed function as a cohesive entity. We also introduce a non-blocking return phase to guarantee that all replicas within a group have equal access to the group votes promptly. Specifically, the contributions are summarized as follows.

- We establish a formal framework for Group BFT protocols with a versatile set of base components and explicit definitions, addressing the challenges inherent in designing such systems.
- We design the Standard Group BFT protocol, which divides replicas into multiple member groups and a leader group, each with an equal number of fixed replicas. Each group has a leader, who manages the group's voting and proposing messages and rotates in a round-robin manner. The leader group leader also serves as the consensus leader.
- We design the Dynamic Group BFT protocol to enhace the flexibility of the Standard Group BFT. Dynamic Group BFT allows membership exchange be-

tween groups by using a unique view-change protocol. We also introduce a data structure called the points table to maximize the number of non-faulty groups during the view-change process.
- We theoretically analyze the safety, liveness, and responsiveness of the Group BFT protocols and conduct a formal analysis of Group BFTs' tolerance and complexity. Our Rust implementation and experimental results exhibit the Group BFT protocols significantly reduce processing bottlenecks of the leader and improve throughput in large-scale systems.

2 Related Work

This section expands on the strategies outlined in Section 1 and introduces additional representative solutions.

Among the traditional protocols addressing the Byzantine Generals Problem [1], PBFT [2] was the first practical consensus protocol under partially synchronous model (as a generic approach that tolerates arbitrary failures, BFT can be categorized into synchronous BFT [20], partially synchronous BFT [21] and asynchronous BFT [22]). However, PBFT has drawbacks such as high communication complexity. Since blockchain [23], [24] emerges, many studies introduce various techniques to solve the block confirmation delay problem in blockchain consensus, trying to reach strong consistency. The possible improvement strategies for traditional BFT protocol is considered once again. RapidChain [14] divide the large network into several shards, where shards work in parallel to maximize performance while requiring significantly smaller communication, computation, and storage per node, allowing the system to scale to large networks. Under the open membership, Byzcoin [25] selects recently successful block miners and designates a leader to drive the BFT protocol, whereas Algorand [26] uses verifiable random functions (VRF) [27] to randomly select a set of users participating in the consensus according to their computing power (or stake) in the designed Byzantine Agreement (BA) [28] protocol to decide the next block.

Chain structure. An innovative idea is to integrate the chain structure into the BFT protocol. Casper [29] and HotStuff [5] utilize pipelining and propose a novel framework that forms a bridge between classical BFT foundations and blockchains. The solution changes the original all-to-all communication phases to simple votes collection from quorum replicas by the leader and then chains these phases to do more useful work. Sync-hotstuff [20] follows the same framework but relies on latency time to discover the leader equivocation and solve Byzantine fault to ensure safety in a weaker synchronous model. Streamlet [30] simplifies the structure design of chained BFT protocol and shows better performance under better network condition. Marlin [31] uses two rounds to achieve finality while maintaining linearity in view-change in the "happy" path. Dolphin [11] leverages non-blocking concurrent block generation to achieve higher throughput of chained BFTs in high-latency environments.

Chain structure provides a simplified foundational framework and enables higher levels of parallelism compared to traditional BFT approaches. Given these advantages, Group BFT also adopts the chain structure paradigm while introducing the novel grouping mechanism to address the scalability challenges in large-scale deployments.

Grouping-like techniques. Some studies explore the application of layering or grouping techniques within BFT protocols to cater to specific scenarios and requirements. SG-PBFT [32] improves PBFT by optimizing the consensus process and using a score grouping mechanism to achieve a higher efficiency in the IoV. Multi-layer PBFT [33] hierarchically grouping nodes into different layers and limiting the communication within the group to enable PBFT in large systems such as massive Internet of Things (IoT) ecosystems. Another concept akin to grouping is the implementation of communication trees [16] in the BFT protocol. Steward [17] presents the first hierarchical BFT architecture to confines the effect of any malicious replica to its local site. GeoBFT [18] uses a topological-aware grouping of replicas in local clusters, and introduces parallelization of consensus at the local level. Kauri [19] leverages a novel pipelining technique to perform scalable dissemination and aggregation on trees. Crackle [34] proposes a κ-level tree structure to accelerate consensus based on a sector-based communication mode.

Overall, most of the above protocols center around the idea of mobilizing non-leader replicas to help forward messages and aggregate votes (signatures) using a hierarchical communication structure. This paper also follows this pattern. However, state-of-the-art protocols typically employ deeper hierarchical structures that, while theoretically capable of further alleviate leader bottlenecks, face significant challenges with fault recovery: nodes of different levels have different positions, a single faulty internal node can compromise all its child nodes. Thus, these protocols often require complex reconfiguration mechanisms beyond simple view changes to handle faulty internal nodes [19], [33]. Additionally, some deeper tree structures establish parallel proposal relationships to mitigate increased latency [18], [34], which may result in ordering challenges. In contrast, Group BFT utilizes a carefully designed two-level hierarchical communication to maintain a balanced structure, where all group leaders (internal nodes) maintain a unique position and operate under equivalent parameters and principles, providing simplicity and efficiency. Group BFT can recover through simple view changes without additional reconfiguration strategies, while preserving simplified and clear transaction ordering through solely conventional chain structures.

In addition, some researches [35], [36] focus on the blocks themselves (primarily letting the leader encode the block into small chunks and tasking each replica with broadcasting a chunk), which represent another important approach to further alleviate network bottlenecks at leaders by balancing the incurred communication load among replicas. However, these approaches often come at the cost of doubling the communication diameter when proposing (adding an extra communication layer over the naive protocol in which the leader directly sends the whole block), and typically require each replica to send messages to all other replicas in this additional communication layer.

DAG solutions. Recently, a series of graph-based studies have made significant progress in addressing the throughput degradation of large-scale BFT systems caused by leader bottlenecks. One notable development is the introduction

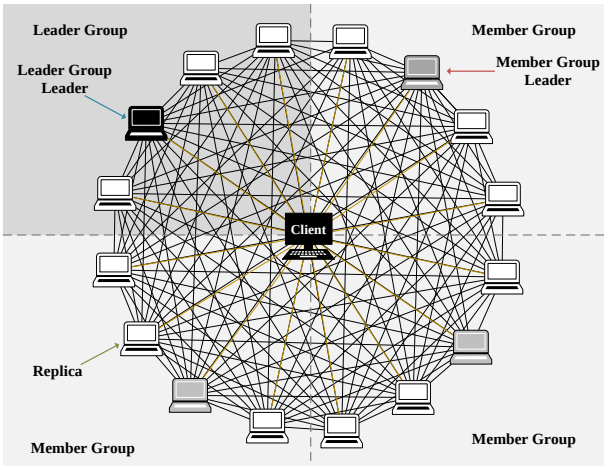


Fig. 2. The system model of Group BFT ($m = 4, n = 4$).

of the Directed Acyclic Graph (DAG) approach, pioneered by Aleph [37], which uses a graph structure instead of the traditional chain structure aiming to solve the problem of throughput decreasing proportionally with increasing latency in the tradtional BFT algorithm. On this basis, DAG-Rider [9], Narwhal [10] achieve zero communication overhead and high throughput in an asynchronous network condition while keep scalability and low latency by abstracting the network communication layer from consensus. Bullshark [13] gives the solution under partial synchronization. Wahoo [38] introduces Provable Broadcast to further reduce communication overhead.

While state-of-the-art DAG approaches also aim to alleviate leader bottlenecks in large-scale BFT systems, they require abstracting the network communication layer from consensus, introducing additional complexity. These solutions typically incur additional memory consumption that gradually accumulates as consensus progresses. Group BFT explores how to mitigate leader bottlenecks in large-scale BFT systems without introducing graph structures, offering a different trade-off in the design space.

3 The Proposed Framework

3.1 Overview

Conventional BFT protocols typically require the election of a global leader to drive consensus. Replicas can send messages or votes to the leader, who is responsible for collecting messages or votes and making decisions based on the protocol rules. Compared to conventional BFT, Group BFT needs to divide the replicas into an equal number of groups, leading to the requirement of more complex communication and coordination mechanisms between group members. We redefine the member roles as follows.

We assume that the system consists of m groups, with each group containing n replicas. Firstly, a leader group is designated among all groups to promote consensus, and the remaining groups are referred to as member groups. Secondly, each group will have a leader. The member group leader is responsible for managing all the replicas within the group, processing their voting messages into a single aggregated voting result and sending it to the leader group leader, while the leader group leader is responsible for managing all the replicas within the leader group, as well as collecting all the voting results sent by the member group leaders and making the final decision based on the protocol rules. In addition, all group leaders also needs to forward the messages sent by the leader group leader to all replicas within the group. In each period of time, called a view v, the system executes a multi-phase consensus protocol over the groups. The system model is shown in Figure 2.

Group BFT can be seen as a communication tree [16] with a level of two, which has prompted inquiries into its extension to hierarchical grouping for quicker consensus. One potential issue with hierarchical grouping is that any fault at a non-leaf node impacts all its child nodes. While two-level grouping also encounters this problem, Group BFT mitigates this with a unique solution where leaders rotate with each view change, preventing a faulty replica from remaining an internal node (or root node) for long. However, once the tree depth increases beyond two levels, the view-change mechanism cannot fully address this problem, as a faulty replica might not change from a non-leaf node to a leaf node after just one round of leader rotation or view-change (this can be assured in two-level grouping). In such cases, additional reconfiguration mechanisms would be needed, which would inevitably increase the system's complexity.

The group leaders (internal nodes) in Group BFT handle both message forwarding and votes (signatures) aggregation. They manage an identical number of leaf nodes, maintaining a unique position between the consensus leader and normal replicas. This balanced structure enables all group leaders to operate under equivalent parameters and principles, providing simplicity and efficiency. Furthermore, Group BFT's consensus flow is solely pipelined through a conventional chain structure, maintaining simplified and clear transaction ordering, whereas BFTs using deeper tree structures may establish parallel relationships between proposals (separating data transmission from consensus) that the latter proposal does not directly reference the previous one to mitigate the increased latency, potentially creating ordering challenges.

3.2 Communication and Network Assumptions

We assume that the replicas in the system process transactions submitted by clients and generate blocks while ensuring global consistency of the operation logs. Once a period of time has elapsed or the maximum batch size has been reached, transactions are packed into a block and handled by the replicas. Each replica r has a public/private key pair (pk_r, sk_r), with the public key serving as its identity. All authorized replicas connect to each other to form consensus groups. Each group also generates a master public key mpk (obtained using the public keys of all members) as the group's identity, and is used to verify the signature obtained from the combination of member partial signatures.

In a round-robin order, the group designated as the leader group for a particular view is determined by $(v + \lfloor v \div mn \rfloor)$ mod m to ensure that each group has a fair opportunity to act as the leader group. Within each group, the leader is determined by the intra group sequence number calculated by v mod n in order to ensure an even

distribution of workload among the replicas within the group. This round-robin fashion will be used for the implicit view change phase of Standard Group BFT. However, in Dynamic Group BFT, since the view change protocol involves dynamic adaptation of group members, it is necessary to redesign the leader and leader group election strategy.

Our Group BFT protocol is designed to operate in a partially synchronous network [39] that has a known finite time bound Δ and a global stabilization time (GST) event. After GST, all transmissions between two correct replicas arrive within time Δ and the adversary must cause the GST event to eventually happen after some unknown finite time. We assume that the network connections between the replicas are reliable and authenticated by their public identities, and messages are broadcasted to all replicas (including itself) in a point-to-point manner. We also assume that the replicas run their clocks at the same rate.

3.3 Threat Model

We consider a system where each group contains n replicas and there are m groups in total. We impose a maximum limit of f_1 compromised or inactive replicas per group, where f_1 is set to be less than one-third of the total number of replicas. A group is considered faulty if this limit is exceeded or if the group leader is compromised. Otherwise, a group is called a non-faulty group. The maximum number of faulty groups is denoted as f_2, where f_2 is set to be less than one-third of the total number of groups. For simplicity, we assume that $m = 3f_2+1$, and that for each non-faulty group, $n = 3f_1+1$.

The compromised replicas exhibit Byzantine faults [1], meaning they can behave arbitrarily, including violating the protocol or refusing to respond. While they can eavesdrop on network communication, they lack the ability to obstruct or alter messages during point-to-point transmission. On the other hand, the non-faulty replicas communicate honestly with each other and faithfully execute the protocol. They verify the authenticity and integrity of messages using digital signatures, which are generated by the sender's private key and verified by the receiver using the corresponding public key. The non-faulty replicas adhere to the protocol rules and do not deviate from the expected behavior.

In a faulty group, the number of compromised replicas is within a range, and in the most extreme scenario, Group BFT's tolerance for malicious replicas may surpass $1/3$ of the overall number of replicas: consider a scenario where there are 4 groups, with Group A being faulty and well over $1/3$ of the replicas of Group A are faulty. Group B, C, and D are non-faulty - there are correct leaders, and no more than $1/3$ of the members in each group are faulty. In this case Group BFT still manages to reach consensus correctly as long as any one of Groups B, C, and D is the leader group. We conduct a formal analysis of Group BFT's robustness for Byzantine faults in Section 5.5 and Section 6.5.

3.4 Design Goals

We design a system that consistently orders users' transactions in a chronological manner. To ensure the effectiveness of the protocol, it must satisfy two fundamental properties:

- **Safety**: Non-faulty replicas have the same view of the system and a consistent log of ordered transactions.
- **Liveness**: Correct client requests are eventually delivered and committed by all non-faulty replicas, enabling progress even in the presence of faults.

Consensus can only be reached in one case in Group BFT:
Case 1: The leader group leader is correct, and at least $2f_2+1$ groups are non-faulty (group leader is correct and number of correct replicas is greater than $2f_1 + 1$).

To satisfy Case 1, the following basic grouping conditions must be met: (1) having more than $2f_2 + 1$ groups with at least $2f_1 + 1$ correct replicas in each group, and (2) there must be at least one sequence number such that all replicas of this number in the $2f_2 + 1$ groups are correct. Meeting these conditions does not guarantee consensus, as each view may still include the following cases:
Case 2: The leader group leader is malicious, and at least $2f_2 + 1$ groups are non-faulty.
Case 3: More than f_2 but not more than $2f_2$ groups are non-faulty (this Case, along with Case 4, arises when certain groups become faulty as a result of the election of malicious replicas as group leaders).
Case 4: More than $2f_2 + 1$ groups are faulty.
To ensure liveness in Cases 2, 3, 4 while maintaining safety, we must demonstrate that the protocol can transition to a new view (case) and retain the snapshot state in such cases. Most importantly, Case 1 will occur eventually.

We also aim to achieve **Responsiveness**, which means the leader does not need to wait excessively for potential conflicts to be resolved before taking action. Our protocol design follows the FLP Impossibility [40].

We further have the following **goals**: firstly, each group in Group BFT can be considered as a cohesive entity, as the completion of the voting process within a group in Group BFT necessitates inter-group voting, requiring treating each group as a single replica. We give the following definition:

Definition 1 (Cohesiveness)**.** *A group can function as a cohesive entity if:*

(1) A non-faulty group only send valid group votes.
(2) All groups' votes are equivalent (all groups have the same voting threshold, apply the same validity conditions to group votes and follow the same voting policy).
(3) Each group is connected to other groups to enable reliable message exchange similar to replicas.
(4) Each group is identifiable and authenticated, ensuring the integrity of the communication.

Another **goal** in Group BFT is ensuring that all replicas have access to the group votes. In every round, inter-group communication takes place only between the leaders of each group. Therefore, we add a non-blocking return phase to the protocol so that each replica is able to receive group votes during this process for convenient verification.

4 Preliminaries

4.1 Threshold Signature

Threshold signatures [41], [42], [43] are a type of digital signature scheme in which a group of n signers collectively produce a signature. The group is defined by a threshold t, which represents the minimum number of signers required to produce a valid signature. Let $S = \{S_1, S_2, \ldots, S_n\}$

denotes the set of signers. Each signer S_i generates a public-private key pair (pk_i, sk_i). The group generates a master public key mpk using the public keys of all signers. To sign a message m, a subset $T \subseteq S$ of at least t signers use their private keys to produce partial signatures, which are then combined to produce a single signature σ. The signature can be verified using the master public key mpk and message m.

In Group BFT, the leaders of each group are responsible for performing a threshold signature after gathering partial signatures from the group members. However, this threshold signature process is not necessary for message transmission between groups. The leader group leader only needs to collect the signatures sent by all groups and then aggregate them (this step can be accomplished using aggregate signatures or other appropriate methods) if their quantity meets the requirements. For the block at height k, the result of this aggregation is denoted as s_k^v in view v.

4.2 Message Structure

The transmission of messages in Group BFT is conducted in two rounds. The first round involves communication between the replicas and the group leader, while the second round involves communication between the member group leaders and the leader group leader. We use public-key digital signatures to authenticate the source identity of messages. Let $\langle m \rangle_r$ denote a message m signed by replica r with the hash digest of the message. In blockchain system, every block is bound to a unique height. We denote B_k be the block at height k To generate such a BFT-supported blockchain, we utilize the following types of messages:

Proposal messages. A leader group leader L_0 creates a proposal message by packing a set of transactions into a block and broadcasting it to all leaders (including itself).

$$\langle \mathsf{propose}, B_k, v, h_{k-1}, s_{k-1}^v \rangle_{L_0}$$

Then, the group leader L_i send a proposal message to all replicas within the group.

$$\langle \langle \mathsf{propose}, B_k, v, h_{k-1}, s_{k-1}^v \rangle_{L_0} \rangle_{L_i}$$

h_{k-1} represents the hash value of the previous valid block B_{k-1}, and s_{k-1}^v is the set of group votes on the previous valid block. The leader signs the proposal to make it valid.

Vote messages. Upon receiving a proposal from the group leader, each replica validates the block and generate partial signature to vote for it. For the voting message of a block at height k in view v, replica r has the following format:

$$\langle \mathsf{vote}, v, h_k \rangle_r$$

After the group leader collects a sufficient number (more than $2f_1+1$) of vote messages containing partial signatures from replicas within the group, they are combined into a digital signature σ for block B_k. Then the member group leader send a group vote message containing this signature to the leader group leader:

$$\langle \mathsf{vote}, v, h_k, \sigma \rangle_{L_i}$$

h_k is the hash value of block B_k. A block is considered valid if (i)It has both the correct signature of the member group leader and the correct signature of the leader group leader, (ii) its predecessor block refers to a valid block, (iii) its packed transactions meet application-level requirements, and (iv) it has enough valid votes from other replicas.

Return messages. The group leader L_i needs to return σ and the voting block to the replicas within the group through the return message. This message is used by the replica to verify whether the next block is an extension of from the current block or to determine the expansion direction of blockchain when timeout. The format of the return message is similar to that of the vote message:

$$\langle \mathsf{return}, v, h_k, \sigma \rangle_{L_i}$$

Note that the return message only needs to be sent one-way to the replica, without requiring a replica response, so this is a non-blocking process.

5 Standard Group BFT

In a Group BFT system, a block, also referred to as a proposal, necessitates the acquisition of $2f_2 + 1$ group votes from the non-faulty groups and must receive two confirmations from subsequent blocks in order to be committed. Each block contains sufficient votes and references the preceding block, establishing a cohesive chain structure. In standard Group BFT, the rotation of leaders follows a round-robin pattern. Similar to Hotstuff [5], an explicit view change protocol was not explicitly designed; instead, it is seamlessly integrated into the state protocol. During the initialization of the protocol, members are randomly divided into distinct groups (or there is a preset grouping scheme) based on predefined parameters . Importantly, member swapping between groups is prohibited. To ensure the consistency and integrity of the chain structure, replicas are required to establish a lock on the return message received from the group leader in each view as well as storing the highest height valid block known to it and its aggregated vote. By validating the legitimacy of the subsequent block based on this locked information, the chain structure of the blockchain is maintained, thereby preserving consistency.

5.1 State Protocol

Figure 3 illustrates the iterative process of the State Protocol. Each iteration is driven by a leader group leader and several member group leaders (equal in number to the total number of groups) of the current view. The proposal phase and the voting phase are divided into two rounds, where round 1 represents the round between the leader group leader and member group leaders, and round 2 represents the round between the group leader and the replicas.

In standard Group BFT, it is crucial for the replica to establish a lock on the return message transmitted by the group leader. The inclusion of a signature within the return message serves as a means to determine whether the block proposed in the next view is an extension of the current block (as the proposal contains s_{k-1}^v). Additionally, the block enclosed in the return message allows the replica to identify the block it voted on during the previous attempt if consensus was not achieved or if a timeout occurred. This information helps determine the direction of the blockchain extension. In addition, it is imperative for all replicas to validate the proposal prior to casting their votes. A proposal

Let v be the current view number, replica L_0 be the leader group leader and set $\mathcal{L} = \{L_1, L_2, ...L_{m-1}\}$ represents the set of member group leaders of the current view. When entering view v by sending a vote on block B_{k-1} to the group leader L_i $(0 \leq i \leq m-1)$, a replica r starts a timer and runs the following protocol.

Vote (round 1) and return (non-blocking). if replica r is the leader group leader L_0 or is a member group leader in set $\mathcal{L}$, upon receiving at least $2f_1+1$ partial signatures $\langle \mathsf{vote}, v, h_{k-1}\rangle_r$ from replicas within the group, combines them into a single signature σ, sends $\langle \mathsf{vote}, v, h_{k-1}, \sigma\rangle_{L_i}$ to L_0 and broadcasts $\langle \mathsf{return}, v, h_{k-1}, \sigma\rangle_{L_i}$ to all replicas within the group.

Propose (round 1). if replica r is the leader group leader L_0, upon receiving at least $2f_2+1$ signatures $\langle \mathsf{vote}, v, h_{k-1}, \sigma\rangle_{L_i}$ from group leaders, construct the aggregate signature $s^v_{k-1} = \sum\langle \mathsf{vote}, v, h_{k-1}, \sigma\rangle_{L_i}$, batch the transactions into a block B_k, and broadcast $\langle \mathsf{propose}, B_k, v, h_{k-1}, s^v_{k-1}\rangle_{L_0}$ to all group leaders (including itself) where $h_{k-1} = H(B_{k-1})$. If a sufficient number of valid votes are not collected, then L_0 populates the h_{k-1} and s^v_{k-1} fields with the highest height valid block known to it and its aggregated vote.

Propose (round 2). if replica r is the leader group leader L_0 or is a member group leader in set $\mathcal{L}$, after receiving a valid proposal $\langle \mathsf{propose}, B_k, v, h_{k-1}, s^v_{k-1}\rangle_{L_0}$ from L_0, sign it again and send $\langle\langle \mathsf{propose}, B_k, v, h_{k-1}, s^v_{k-1}\rangle_{L_0}\rangle_{L_i}$ to all replicas with in the group (including itself).

Vote (round 2). Before the timeout, upon receiving a valid proposal $\langle\langle \mathsf{propose}, B_k, v, h_{k-1}, s^v_{k-1}\rangle_{L_0}\rangle_{L_i}$ from L_i, the leader of the current group, where $h_{k-1} = H(B_{k-1})$, store h_{k-1} and s^v_{k-1} replacement to the already stored values (if the corresponding block of h_{k-1} is higher than the currently stored valid block), send $\langle \mathsf{vote}, v+1, h_k\rangle_r$ to the group leader of view $v+1$ where $h_k = H(m_k)$ and reset the timer. Otherwise, after the timeout, send $\langle \mathsf{vote}, v+1, h_{est}\rangle_r$ to the group leader of view $v+1$ and reset the timer to try switching to the next view, where h_{est} is the highest valid block known to r.

Commit. When receiving a valid block B_k and this block at the current height emerging for the first time, it means that B_k's referred block has already got at least $2f_2+1$ group votes. Meanwhile, the block before this referred block obtains confirmations from the majority of replicas twice. Thus, commit the block at height $k-2$.

Fig. 3. The state protocol of Standard Group BFT

is deemed **valid** if it satisfies either of two conditions: its aggregated vote belongs to a higher numbered view (greater than the lock message of the replica) or it incorporates blocks that extend from blocks included in the lock message. The former condition adheres to the liveness rule, while the latter adheres to the safety rule. Further more, replicas also need to store the highest height valid block currently known to it and its aggregated vote. Specifically, replicas need to update the stored values in a timely manner upon receipt of a higher height valid message, which is essential to ensure the liveness of the protocol.

The view-change protocol is implicitly integrated into the state protocol in standard Group BFT. In this integration, the voting process always operates on blocks in the current view but is sent to the next view. Consequently, once replicas complete voting, the responsibility for subsequent tasks is transferred to the group leaders of the next view. The timer needs to be reset after the replica completes voting, which involves two possible scenarios. Firstly, if the protocol is running smoothly and the replica votes on a proposal at a higher height, the timer is reset accordingly, in which case the timer will be reset to its initial state. Secondly, the timer may run out and subsequently reset. In this case, the replica multiplies the timer's timeout duration and endeavors to transition to the next view. To initiate this transition, the replica increments the view number by 1 and dispatches a vote message containing the highest valid block known to it to the group leader of the new view. Given the significance of the voting phase, we use the second round of voting as the end of the iteration, while the first round of voting is placed at the beginning of the iteration process in Figure 3 for ease of understanding.

5.2 Proof of Cohesiveness

A tree is a hierarchical data structure in which every node has exactly one parent, except the root node which has no parent. A branch is a sequence of connected nodes starting from the root node and ending at a leaf node in the tree. Two branches are in conflit if neither is an extension of the other. Two nodes are in conflict if they point to conflicting branches. Considering the blockchain as a tree, blocks serve as the nodes, enabling us to define conflicting blocks. This concept will be utilized in subsequent proofs.

Our proof proceeds along the following lines: first, we show that a group in Group BFT can be considered as a cohesive entity to participate in consensus, as per Definition 1. We then show that our protocol satisfies safety, liveness and responsiveness.

Lemma 1. *Let G be a non-faulty group. Then $\forall gvote \in \mathcal{V}_G$, Valid$(gvote)$ holds, where $\mathcal{V}_G$ is the set of all group votes produced by G and Valid$(gvote)$ denotes that $gvote$ satisfies protocol-defined validity conditions.*

Proof. We establish this via the following sub-properties:

(1) $\forall r \in R_{correct}$, $\forall vote_r \in \mathcal{V}r$: $\mathsf{Valid}(vote_r)$, where $R_{correct}$ is the set of correct replicas and $\mathcal{V}_r$ is the set of votes produced by replica r.

(2) $\forall G \in \mathcal{G}_{non-faulty}$: $|R_{correct} \cap G| \geq 2f_1+1$, where $\mathcal{G}_{non-faulty}$ is the set of non-faulty groups.

(3) Let $\sigma_G(B) = \mathsf{Combine}(\{vote_r(B) | r \in G \wedge \mathsf{Valid}(vote_r(B))\})$ be the threshold signature for block B from group G. Then $\mathsf{Valid}(\sigma_G(B)) \iff |\{vote_r(B) | r \in G \wedge \mathsf{Valid}(vote_r(B))\}| \geq 2f_1+1$.

In a non-faulty group G, the group leader is correct and by (2), $|R_{correct} \cap G| \geq 2f_1+1$. By (1), all votes from correct replicas are valid. By (3), a valid group signature can be combined to make a group vote valid if and only if at least $2f_1+1$ replicas voted for it. Since at least $2f_1+1$ correct replicas exist in the group, and the group leader is correct, the group vote must be valid. Therefore, $\forall gvote \in \mathcal{V}_G$, $\mathsf{Valid}(gvote)$ holds. □

Lemma 2. *$\forall G_i, G_j \in \mathcal{G}$, the vote equivalence relation $\sim$ holds such that $\mathcal{V}_{G_i} \sim \mathcal{V}_{G_j}$, where $\mathcal{G}$ is the set of all groups and $\mathcal{V}_{G_i}$ represents the set of valid votes from group G_i.*

Proof. We define the vote equivalence relation $\sim$ as follows according to Property (2) in Definition 1: $\mathcal{V}_{G_i} \sim \mathcal{V}_{G_j} \iff$

- $\forall G_i, G_j \in \mathcal{G}$: $t_{G_i} = t_{G_j}$, where t_G is the voting threshold for group G.
- $\forall G_i, G_j \in \mathcal{G}, \forall gvote \in \mathcal{V}_{G_i} \cup \mathcal{V}_{G_j}$: $\mathsf{Valid}_i(gvote) \iff \mathsf{Valid}_j(gvote)$, where $\mathcal{V}_{G_i}$ is the set of all group votes produced by G_i and $\mathsf{Valid}_i(gvote)$ denotes that $gvote$ satisfies the validity conditions defined for group G_i.
- $\forall G_i, G_j \in \mathcal{G}$: $\tau_{G_i} = \tau_{G_j}$, where τ_G is the voting policy for group G.

In Group BFT, by protocol definition: all groups have the same voting threshold $t_{G_i} = 2f_1 + 1, \forall G_i \in \mathcal{G}$; All groups apply the same validity conditions to group votes; All groups follow the same voting policy. Therefore, we can conclude that $\forall G_i, G_j \in \mathcal{G}$: $\mathcal{V}G_i \sim \mathcal{V}G_j$. □

Lemma 3. *The communication graph $\mathcal{C}_\mathcal{G} = (\mathcal{G}, E_\mathcal{G})$ formed by groups is a complete graph, where $(G_i, G_j) \in E_\mathcal{G} \iff$ groups G_i and G_j can exchange messages reliably.*

Proof. Let $\mathcal{C}_\mathcal{R} = (\mathcal{R}, E_\mathcal{R})$ be the communication graph formed by replicas, where $\mathcal{R}$ is the set of all replicas and $(r_i, r_j) \in E_\mathcal{R} \iff$ replicas r_i and r_j can exchange messages reliably. By network assumption, $\mathcal{C}_\mathcal{R}$ is a complete graph, i.e., $\forall r_i, r_j \in \mathcal{R}$: $(r_i, r_j) \in E_\mathcal{R}$.

The group communication edge set $E_\mathcal{G}$ is defined as: $(G_i, G_j) \in E_\mathcal{G} \iff \forall r_i \in G_i, r_j \in G_j$ such that $(r_i, r_j) \in E_\mathcal{R}$. Since $\forall r_i, r_j \in \mathcal{R}$: $(r_i, r_j) \in E_\mathcal{R}$, and each group contains at least one replica, it follows that $\forall G_i, G_j \in \mathcal{G}$: $(G_i, G_j) \in E_\mathcal{G}$. Therefore, $\mathcal{C}_\mathcal{G}$ is a complete graph, establishing each group in Group BFT is connected to other groups to enable reliable message exchange. □

Lemma 4. *$\forall G \in \mathcal{G}, \exists$ unique identifiers ID_G and mpk_G such that: $\forall G_i, G_j \in \mathcal{G}, G_i \neq G_j \implies IDG_i \neq IDG_j$; $\forall gvote \in \mathcal{V}_G, \forall r \in \mathcal{R}$: $r.\mathsf{Verify}(gvote.\sigma_G, mpk_G) \wedge$ ($gvote$ signed by L) $\implies$ $gvote$ originated from G, where L is the leader of group G, $r.\mathsf{Verify}(gvote.\sigma_G, mpk_G)$ means replica r can verify the threshold signature in $gvote$ using master public key mpk_G.*

Proof. In Group BFT, each group has two types of identifiers:

- Group Leader Identifier: Let L_i be the leader of group G_i. Then $ID_{G_i} = pk_{L_i}$, where pk_{L_i} is the public key of L_i. Since each replica has a unique public/private key pair, and group leaders are distinct replicas, we have $\forall G_i, G_j \in \mathcal{G}, G_i \neq G_j \implies L_i \neq L_j \implies pk_{L_i} \neq pk_{L_j} \implies ID_{G_i} \neq ID_{G_j}$.
- Master Public Key: Let mpk_G be the master public key of group G derived (via an injective or collision-resistant function) from the public keys of all members in G. Since each group has a unique set of members, $\forall G_i, G_j \in \mathcal{G}, G_i \neq G_j \implies \{pk_r | r \in G_i\} \neq \{pk_r | r \in G_j\} \implies mpk_{G_i} \neq mpk_{G_j}$.

For any group vote $gvote \in \mathcal{V}_{G_i}$ signed by group G_i, we have: $gvote$ is signed by the group leader L_i using sk_{L_i}, verifiable using pk_{L_i}; $gvote$ contains a threshold signature generated from partial signatures of at least $2f_1 + 1$ group members, verifiable using mpk_G. Therefore, $\forall gvote \in \mathcal{V}_G, \forall r \in \mathcal{R}$: if r successfully verifies $gvote$ using mpk_G and pk_{L_i}, then $gvote$ must have originated from group G. □

Theorem 5 (Cohesiveness)**.** *Each group in Group BFT can be considered as a cohesive entity.*

Proof. We need to show that each group $G \in \mathcal{G}$ satisfies all four properties required for cohesiveness, as defined in Definition 1. From Lemma 1, we have $\forall G \in \mathcal{G}_{non-faulty}, \forall gvote \in \mathcal{V}_G$: $\mathsf{Valid}(gvote)$, which establishes that a non-faulty group only send valid group votes. From Lemma 2, we have $\forall G_i, G_j \in \mathcal{G}$: $\mathcal{V}_{G_i} \sim \mathcal{V}_{G_j}$, which establishes that all groups' votes are equivalent. From Lemma 3, we have that $\mathcal{C}_\mathcal{G} = (\mathcal{G}, E_\mathcal{G})$ is a complete graph, which establishes that each group is connected to other groups to enable reliable message exchange similar to replicas. From Lemma 4, we have $\forall G \in \mathcal{G}, \exists$ unique identifiers ID_G and mpk_G that allow for message authentication, which establishes that each group is identifiable and authenticated, ensuring the integrity of the communication.

Since all four required properties are satisfied, we conclude that each group in Group BFT can be considered as a cohesive entity. □

5.3 Proof of Safety, Liveness and Responsiveness

Let $\mathcal{G}_{basic}$ be the set of groups with $\geq 2f_1+1$ correct replicas, i.e. $(\mathcal{G}_{basic} \subseteq \mathcal{G}) \wedge (|\mathcal{G}_{basic}| > 2f_2)$. $\mathcal{G}_{non-faulty} \subseteq \mathcal{G}_{basic}$.

Lemma 6. *$\forall G_i \in \mathcal{G}_{basic}$: $\forall gvote_1, gvote_2 \in \mathcal{V}_{G_i}$: $(gvote_1 = \langle \mathsf{vote}, v_1, h_1, \sigma_1 \rangle_{L_i} \wedge gvote_2 = \langle \mathsf{vote}, v_2, h_2, \sigma_2 \rangle_{L'_i} \wedge h_1 \neq h_2) \Rightarrow v_1 \neq v_2$, where $\mathcal{V}_{G_i}$ is the set of all valid votes from G_i.*

Proof. To show a contradiction, suppose $\exists gvote_1, gvote_2 \in \mathcal{V}_{G_i}$ such that $h_1 \neq h_2$ and $v_1 = v_2 = v$. Since $|G_i| = 3f_1 + 1$ and each valid threshold signature (σ_1, σ_2) in $gvote_1, gvote_2$ can only be generated if $2f_1 + 1$ valid partial signatures are collected within G_i. Let $\mathcal{R}_1 = \{r \in G_i | r \text{ contributed to } \sigma_1\}$, $\mathcal{R}_2 = \{r \in G_i | r \text{ contributed to } \sigma_2\}$. $|\mathcal{R}_1| \geq 2f_1 + 1$ and $|\mathcal{R}_2| \geq 2f_1 + 1$. $|\mathcal{R}_1| + |\mathcal{R}_2| - G_i \geq f_1 + 1$. Even if all compromised replicas in G_i voted twice in view v, there still must be a correct replica who voted twice in v. This is impossible, because a correct replica only vote once per view to the group leader of the next view in the second round of the voting phase. Thus, $v_1 \neq v_2$ must hold. □

Lemma 7. *$\forall B_1, B_2 \in \mathcal{B}_{valid}$: $h(B_1) \neq h(B_2) \Rightarrow B_1, B_2$ must not be commited in the same view, where $\mathcal{B}_{valid}$ is the set of valid blocks*

Proof. We prove by contradiction. Suppose $\exists B_1, B_2 \in \mathcal{B}_{valid}$ both have been commmited in view v. Since $|\mathcal{G}| = 3f_2 + 1$ and a valid block can be commited only with $m - f_2 = 2f_2 + 1$ groups vote for it. Similarly to Lemma 6, there must be a $G_i \in \mathcal{G}_{basic}$ who submitted two valid group votes in the same v. Let these votes be: $gvote_1 = \langle \mathsf{vote}, v, H(B_1), \sigma_1 \rangle_{L_i}$ and $gvote_2 = \langle \mathsf{vote}, v, H(B_2), \sigma_2 \rangle_{L_i}$. Since $H(B_1) \neq H(B_2)$, this is impossible as the non-faulty group cannot submit two valid group votes under the same v according to Lemma 6. Thus B_1, B_2 must not be commited in the same view. □

Theorem 8 (Safety)**.** *$\forall B_1, B_2 \in \mathcal{B}$: if B_1, B_2 are conflicting blocks, then they cannot be both committed, each by a non-faulty group (or a group with at least $2f_2 + 1$ correct replicas), where $\mathcal{B}$ is the set of all blocks*

Proof. We can prove this theorem by contradiction. Let's assume $\exists B_1, B_2 \in \mathcal{B}$ such that B_1 and B_2 are conflicting blocks, with B_1 committed in view v_1 and B_2 committed

in view v_2. Based on Lemma 7, we know that $H(B_1) \neq H(B_2) \Rightarrow v_1 \neq v_2$. Without loss of generality, we assume that $v_1 < v_2$. Let $v_r = \min\{v | v > v_1 \wedge (\exists B_z \in \mathcal{B}$ such that B_z conflicts with $B_1) \wedge$ (there exists a valid proposal message for block B_z in view v from the leader group leader)$\}$, It is important to note that, by assumption, such a proposal must exist. E.g., it could be the proposal in v_2. Among the correct replicas that sent a partial signature $\langle \mathsf{vote}, v_1, H(B_1)\rangle_r$, let r be the first replica that contributed a partial signature $\langle \mathsf{vote}, v_z, H(B_z)\rangle_r$. Such an r must exist; otherwise, either $s_1^{v_1}$ (the set of group votes on B_1 when proposing B_z) or $s_z^{v_z}$ could not have been created.

During v_1, r is locked on the group leader's return message for the combined partial signature on B_1. Due to the minimality of v_r, the lock that replica r has on the branch led by B_1 remains unchanged upon receiving the proposal for B_z. At this point, the proposal is not recognized as valid because $s_1^{v_1}$ does not in a larger view than r's lock, and B_z is not an extension of the block referenced in the lock message (since B_z conflicts with B_1). Consequently, replica r does not vote for the proposal for B_z, leading to a contradiction. By reaching this contradiction, we can conclude that our initial assumption was incorrect. Therefore, B_1, B_2 cannot be both committed. □

We will now analyse the four cases outlined in Section 3.4. Our objective is to demonstrate the inevitability of Case 1, as it is crucial for ensuring liveness. Let Φ_{Case1} denote the basic grouping conditions for Case 1 as defined in Section 3.4: $\Phi_{Case1} \equiv (\mathcal{G}_{basic} \subseteq \mathcal{G}) \wedge (|\mathcal{G}_{basic}| > 2f_2) \wedge (\forall G \in \mathcal{G}_{basic} : |G \cap \mathcal{R}_{correct}| > 2f_1) \wedge (\exists j \in \{0, 1, ..., n-1\} : \forall i \in \mathcal{I}_{non-faulty}, r_{i,j} \in \mathcal{R}_{correct})$ where $\mathcal{I}_{basic}$ is the set of indices of groups in $\mathcal{G}_{basic}$, $r_{i,j}$ is the replica with sequence number j in group i.

Lemma 9. *If Φ_{Case1} holds, $\forall r \in \mathcal{R}_{correct}$, r can perform a view change successfully.*

Proof. In Case 1, the consensus is reached unless network conditions cause delays exceeding the timeout (moving to the following case). Therefore, view changes occur naturally as part of the protocol flow. In Cases 2-3, consensus might not be reached within the timeout period, and if so, the replica timer eventually times out in view v and sends $\langle \mathsf{vote}, v+1, h_{est}\rangle_r$ to the group leader of $v+1$. After sending the vote, the timer is reset and the view number is incremented by 1. In Case 4, there are at least $2f_2+1$ fault groups. However, since $|\mathcal{G}_{basic}| > 2f_2$, there exists at least $2f_1+1$ correct replicas in each of at least $2f_2+1$ groups. Therefore, no less than f_2+1 faulty groups are faulty due to the malicious group leader, consensus on the group votes of the malicious replicas cannot be reached. As a result, the timer of the correct replicas will eventually time out, reverting back to the previous case.

Therefore, in all cases, if Φ_{Case1} holds, correct replicas will eventually transition to the next view. □

Lemma 10. *If Φ_{Case1} holds, then case will update accordingly after a view change.*

Proof. Let us model the system as a grid $\mathcal{M}$ of size $m \times n$:

- $m = 3f_2 + 1$ (total number of groups)
- $n = 3f_1 + 1$ (number of replicas per group)
- $\mathcal{M}[i, j] \in \{0, 1\}$ where 1 indicates a correct replica and 0 indicates a compromised replica
- Row i represents group G_i, Column j represents all replicas with sequence number j in their groups

Since Φ_{Case1} holds, in $\mathcal{M}$, there exist at least $2f_2+1$ rows where at least $2f_1+1$ elements are 1s and there exists at least one column where all elements in the $2f_2+1$ rows of correct groups are 1s. In a round-robin order, the leader of G_i in view v is determined by: $L_i(v) = v \bmod n$. By Lemma 9, if at least $2f_2+1$ groups with at least $2f_1+1$ correct replicas in each complete a view change, the system transitions to view $v+1$ with a new leader group and new group leaders. This corresponds to shifting the "marked column" (representing current leaders) one position to the right in our grid model - also corresponding to a new case. □

Lemma 11. *If Φ_{Case1} holds, then case 1 will occur eventually.*

Proof. Let's still consider the grid $\mathcal{M}$ of size $m \times n$ where $m = 3f_2 + 1$ and $n = 3f_1 + 1$. Let $\mathcal{M}[i, j] \in \{0, 1\}$ where 1 indicates a correct replica and 0 indicates a compromised replica. Since Φ_{Case1} holds, in the worst-case scenario: (1) Exactly $2f_2 + 1$ rows have exactly $2f_1 + 1$ elements set to 1 (minimum number of correct replicas); (2) Only one column (denoted as COL*) contains all 1s in these $2f_2 + 1$ rows. According to Lemma 10, the "marked column" keep shifting as the view changes. Let v'' be the first view where the leader selection corresponds to column COL*:

$$v'' = \min\{v \in \mathbb{N} : v \bmod n = \text{COL}^*\}$$

The leader group in view v is determined by: $(v + \lfloor v \div mn \rfloor) \bmod m$. When the system reaches view v'', either: the leader group is non-faulty and has a correct leader (Case 1), or the leader group is faulty (Case 2, 3, or 4). For the latter case, we need to show that Case 1 will eventually occur. We prove this using the following mathematical claims:

Claim 1: For any positive integers a, b, and integer x such that $0 \leq x \leq 3a$, the sequence defined by $d_c = x + c(3b+1)$ for $c \in \mathbb{Z}_{\geq 0}$ covers at least $a+1$ different integers between 0 and $3a$, when evaluated in the form $(d_c + \lfloor \frac{d_c}{(3a+1)(3b+1)} \rfloor) \bmod (3a+1)$.

Claim 2: For any positive integers a, b, and integer x such $0 \leq x \leq 3a$, the sequence defined by $d_c = x + c(3b+1)$ for $c \in \mathbb{Z}_{\geq 0}$ yields a complete residue system modulo $3a+1$ when evaluated in the form $(d_c + \lfloor \frac{d_c}{(3a+1)(3b+1)} \rfloor) \bmod (3a+1)$.

Note that Claim 2 implies Claim 1. We prove Claim 2 by giving a subsequence of d_c: let $c = i(3a+1)$, where $i \in \mathbb{Z}_{\geq 0}$. Applying these claims with $x = v''$, $a = f_2$, and $b = f_1$, we obtain: let $\mathcal{S} = \{(v'' + i(3f_1+1) + \lfloor \frac{v''+i(3f_1+1)}{(3f_2+1)(3f_1+1)} \rfloor) \bmod (3f_2+1) : i \in \mathbb{Z}_{\geq 0}\}$. By Claim 1, $|\mathcal{S}| \geq f_2 + 1$. If the leader group in v'' is one of the f_2 faulty groups, then since $|\mathcal{S}| \geq f_2 + 1$, there exists at least one view $v' = v'' + i(3f_1+1)$ for some $i \in \mathbb{Z}_{\geq 0}$ such that: the leader selection is still column COL* (since $v' \bmod n = v'' \bmod n = \text{COL}^*$) and the leader group is non-faulty (since $|\mathcal{S}| \geq f_2 + 1$ and, $\forall i$, there are at least fixed $2f_2 + 1$ non-faulty groups in v'). Therefore, Case 1 will occur in v', even in the worst case scenario. So far, we have proved that Case 1 will occur eventually □

Lemma 12. *Let $r \in \mathcal{R}_{correct}$ and L'_0 be the leader group leader of view $v' > v$. If r receives a valid proposal $p =$*

$\langle propose, B'_{k+1}, v', h_k, s_k^v \rangle_{L'_0}$ from L'_0, check if the contained h_k is equal to $H(B_k)$, where B_k is the block in view v at height k, then $\exists \mathcal{G}_b \subseteq \mathcal{G}_{basic}$:$|\mathcal{G}_b| \geq f_2+1 \wedge (\forall G_i \in \mathcal{G}_b, G_i$ voted for $B_k)$.

Proof. For p to be valid, s_k^v must be a valid aggregated signature containing votes from at least $2f_2+1$ groups for block B_k in view v. Let $\mathcal{G}_v(B_k) = \{G \in \mathcal{G} : \text{G voted for } B_k \text{ in view } v\}$. By the validity requirement: $|\mathcal{G}_v(B_k)| \geq 2f_2+1$. Since $|\mathcal{G}_{basic}| \geq 2f_2+1$, by the pigeonhole principle:

$$\begin{aligned}|\mathcal{G}_b| &= |\mathcal{G}_v(B_k) \cap \mathcal{G}_{basic}| \geq |\mathcal{G}_v(B_k)| + |\mathcal{G}_{basic}| - |\mathcal{G}| \\ &\geq (2f_2+1) + (2f_2+1) - (3f_2+1) = f_2+1\end{aligned}$$

Thus, at least f_2+1 groups in $\mathcal{G}_{basic}$ voted for B_k in v. □

Theorem 13 (Liveness)**.** *After GST, there exists a time period T that all non-faulty groups remain in the same view v. A correct leader group leader in such a view can always broadcast a new block at a higher height.*

Proof. By Lemma 11, if Φ_{Case1} holds, then there exists a view v such that the system is in Case 1 during view v. In Case 1: the leader group leader L_0 is correct and $\forall G \in \mathcal{G}_{basic}, G \in \mathcal{G}_{non-faulty}$. After GST, all messages between correct replicas will arrive within Δ time. When starting view v, L_0 either collects at least $2f_2+1$ group votes for some block B_k at height k and assembles a valid aggregate signature s_k^v, or uses a valid $s_k^{v'}$ from a previous $v' < v$ for the highest-height block B_k with sufficient votes. By lemma 12, at least f_2+1 groups in $\mathcal{G}_{basic}$ voted for B_k, and have already sent them to L_0 in their vote messages. Thus, L_0 must learn a matching block and after collecting the necessary votes, L_0 creates a new block B_{k+1} extending B_k in the new proposal. L_0 broadcasts this proposal to all group leaders. Since L_0 is correct and the network is stable after GST, all correct group leaders receive this proposal within Δ time. The correct group leaders then forward the proposal to their group members, who verify and vote for it, leading to form a new block at a higher height. Since all correct replicas follow the protocol and network is stable, this process completes within a bounded time T. □

Theorem 14 (Responsiveness)**.** *When network becomes synchronous, Group BFT can generate correct group leaders to achieve consensus within the actual network delay, rather than the maximum delay.*

Proof. There is no explicit "wait-for-Δ" step in Group BFT. It adds a non-blocking return phase to a multi-round extension of the regular three-phase paradigm, and performs validation verification on received new messages to override stale locks and switch to more up-to-date proposals. Once $\geq 2f_2+1$ valid group votes are submitted through voting phase, the leader group leader can broadcast a new block to replicas, achieving Optimistically Responsive. □

5.4 Complexity

In Standard Group BFT, the transmission of all messages is conducted in two rounds. In the first round, the leader group leader sends messages to m group leaders. The communication complexity for this round is $O(m)$. In the second round, each group leader sends messages to n replicas. Since each group leader needs to send messages to n replicas, this operation is performed m times in total. Therefore, the communication complexity for this round is $O(m*n)$. Thus, our protocol also has $O(m*n)$ in total communication complexity. Only the block at depth 2 can be safely committed, and we still set unit time for spreading one block. The total execution time p can let us get $p-2$ decided blocks.

Standard Group BFT seems to have similar complexity to Hotstuff and its variant schemes, i.e., both are $O(m*n)$ when there are $m*n$ replicas in total. However, due to other factors such as network congestion, message processing overhead, and resource availability, Group BFT actually has a better performance, and the more the number of replicas, the more obvious the advantage. On the one hand, the strategy of the leader group leader sending messages to m group leaders and each group leader sending messages to n replicas not only potentially distribute the message load across multiple replicas and reduce the burden on the leader, but also parallelises the operation of a single leader to multiple group leaders compared to the leader sending messages directly to $m*n$ replicas. On the other hand, the efficiency of performing intra-group threshold signatures for m groups separately is higher than the efficiency of performing threshold signatures for $m*n$ replicas together.

5.5 Byzantine Tolerance

Group BFT achieves consensus as long as the number of faulty groups is below $1/3$ of the total number of groups. However, within a faulty group, the number of compromised replicas can range from f_1+1 to n. This implies that the protocol can tolerate a range of compromised replicas rather than a fixed value. For the sake of simplicity, we analyze the case where there are a total of $m = 3f_2+1$ groups, each containing $n = 3f_1+1$ replicas.

First, we calculate the lower bound for the number of compromised replicas. This lower bound represents the maximum number of compromised replicas that can be tolerated by any grouping scheme while still achieving consensus. Conversely, we can also find a specific grouping scheme in which having one more compromised replica than this lower bound would prevent consensus. To prevent consensus, it requires at least f_2+1 faulty groups, each having at least f_1+1 compromised replicas. Thus, the lower bound is $(f_2+1)(f_1+1)-1$.

When $f_1 \geq 2$ and $f_2 \geq 2$, the upper bound is $(2f_2+1)(2f_1+1)-1$. We can prove this by considering the following two properties: 1) When there are $(2f_2+1)(2f_1+1)$ compromised replicas, they can reach an erroneous consensus if they are evenly distributed among $2f_2+1$ groups. 2) There exists at least one grouping scheme in which the system can still achieve consensus when the number of faulty replicas is $(2f_2+1)(2f_1+1)-1$. This is due to the inequality $(3f_2+1)(3f_1+1)-[(2f_2+1)(2f_1+1)-1] \geq (2f_2+1)(2f_1+1)$, which guarantees that the system can satisfy the basic condition for consensus even after including this number of compromised replicas. By simplifying this inequality, we obtain $(f_1-1)(f_2-1) \geq 1$, which holds truewhen f_1, f_2 are both greater than 0 as $f_1 \geq 2, f_2 \geq 2$.

In summary, we can draw the following conclusions regarding the tolerance of Group BFT:

- Consensus can be achieved when the number of faulty groups does not exceed f_2 (Safety, Liveness).

- Any grouping scheme can achieve consensus as long as the number of compromised replicas does not exceed $(f_2+1)(f_1+1)-1$ (Safety, Liveness).
- When the number of compromised replicas exceeds $(f_2+1)(f_1+1)-1$ but is $\leq min([(3f_2+1)(3f_1+1)-[(2f_2+1)(2f_1+1)-1]], (2f_2+1)(2f_1+1))$, at least one grouping scheme enables consensus to be reached (Liveness under Dynamic Group BFT).

Use f_{sum} to denote the total number of compromised replicas. In fact, since the initialised grouping scheme in Standard BFT is random, there is a probability that we will not be able to reach a consensus when the number of compromised replicas falls between the upper and lower bounds in Group BFT. To quantitatively analyse this probability, we can map the grouping problem to a simplified problem known as the Ball-in-Box problem with finite box volume. The problem is described as follows:

Consider $m*n-f_{sum}$ identical balls that need to be distributed into m boxes. The boxes are identical and can contain a limited number of balls. Let m_1 denote the number of boxes that can hold a maximum of q_1 balls each, and $m-m_1$ denote the remaining boxes that can hold a maximum of q_2 balls each. The goal is to determine the number of possible scenarios. There are several ways to solve this problem, such as the generating function method, i.e., constructing a generating function to represent the cases in which each box can be placed from 0 to the maximum number of balls:

$$f(x,y)=\sum_{i=0}^{y}x^i \tag{1}$$

and then multiplying both types of generating functions to the coefficients of the corresponding number of times to get the total number of ways to place the balls:

$$g(f_{sum},n,m,m_1,q_1,q_2)= Coef[Exp[f(x,q_1)^{m_1}f(x,q_2)^{(m-m_1)}],x,m*n-f_{sum}] \tag{2}$$

where Exp represents finding the expansion of the expression $f(x,q_1)^{m_1}f(x,q_2)^{(m-m_1)}$, and $Coef$ represents finding the coefficients of $x^{(m*n-f_{sum})}$ in the polynomial that has been expanded.

When $q_1=q_2=3f_1+1$, we can calculate the total number of possible cases. Similarly, when $m_1=2f_2$, $q_1=2f_1$, and $q_2=3f_1+1$, we can determine the number of scenarios where consensus cannot be reached. By dividing the latter by the former, we obtain the probability of not being able to reach consensus under the random grouping approach. However, the diversity of parameters makes it difficult to analyse the exact range of this probability, but we can use the above method to analyse the most appropriate m,n when the total number of replicas is fixed. In situations where the total number of replicas cannot be divided evenly into groups of appropriate sizes, we adopt a strategy of rounding down the total number of replicas divided by the group size and leaving the remaining replicas as a separate group, though this group may not be able to satisfy the minimum number of votes required.

6 Dynamic Group BFT

In standard Group BFT, members are randomly divided into distinct groups based on predefined parameters during the initialization of the protocol and swapping between groups is prohibited. The rotation of leaders follows a round-robin pattern, ensuring fairness and load balancing among the members. However, the completely random grouping in this approach can lead to consensus failure when compromised replicas fall within a certain range. To address this issue, we therefore propose another possible architecture for Group BFT called dynamic Group BFT, where the view-change protocol is decoupled from the state protocol and operates independently with greater flexibility. It allows for the exchange of members between groups, enabling dynamic adaptation and improved resilience against compromised replicas. The goal of this standalone view-change protocol is to facilitate the proper functioning of the consensus. When it is not triggered, the normal state protocol continues to run seamlessly, and the leaders of the groups remain unchanged.

6.1 Overview

The view-change protocol ensures liveness while guarantees safety across views. In dynamic Group BFT, the view-change protocols can vary based on specific objectives. The primary objectives are as follows:

- Correct replicas within the same group should be able to identify the same group leader, ensuring consistency within each group.
- All groups should be able to identify the same leader group, facilitating coordination and communication between groups.
- The grouping approach aims to maximize the number of non-faulty groups, enhancing the fault tolerance of the system.

While considering the historical behavior of replica holders participating in other applications can be beneficial, we focus on a view-change scenario specifically tailored to the behavior of replicas in the consensus process.

We begin by defining a data structure called the points table, which captures the historical behavior of replicas participating in the consensus. This table includes two types of scores for each replica: the behavioral score and the additional score. The former reflects the number of successful participations in the consensus, while the latter serves as a tiebreaker when replicas have the same behavioral score. Initially, all replicas have a behavioral score of 0, and the replicas are ordered based on the additional score in descending order. Each replica is formed into a group of n replicas in the order of the points table. During each iteration, the return message sent by the group leader contains valuable information about the replicas that have successfully participated in the consensus process. Leveraging this information, replicas can automatically update the points table to reflect the contributions of each replica (e.g., adding a behavioral score to the replicas contributing the signature).

Note that compromised replicas can skip the bonus stage by ignoring this process, but they cannot arbitrarily falsify replicas points, given that replicas can be required to provide valid proof aggregated from in-group votes/signatures for any completed bonus stage, and any replica should be able to verify the validity of a points table. In addition, while a faulty group leader may selectively delete messages originating from correct replicas, it would need at least $2f_1+1$

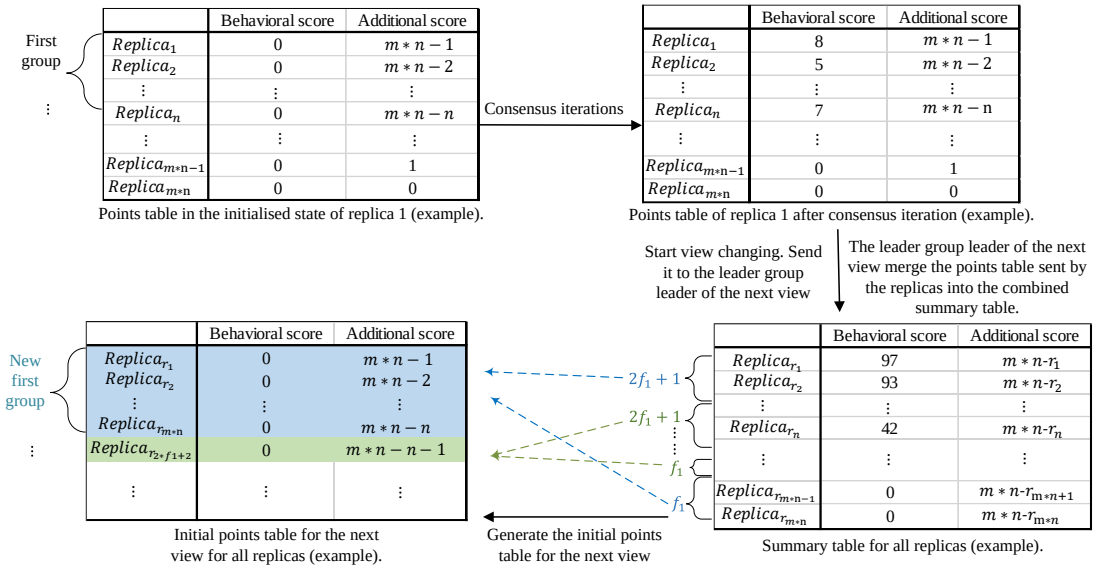


Fig. 4. Points table.

valid votes to construct an in-group voting proof to validate the current view's score, and all in-group replicas that know about the proof will add points to the contributing replicas for those $2f_1 + 1$ votes. Replicas typically focus on scoring within the group, resulting in benefits. Even if a final consensus cannot be reached in a certain grouping method, replicas can still earn points for accurately participating in the consensus through in-group proofs, thus laying the groundwork for correctly reaching a consensus after the next view-change.

When a view change occurs, the leader group leader of the next view collects the score tables of replicas, combines them into a single summary table, rearrange the table in order of behavioral scores from highest to lowest, and then return the table to all the replicas. The groups that contribute valid group votes (i.e. send them to the leader group leader before timeout, regardless of whether the votes are used to construct aggregated signatures) for all valid blocks of previous view remain unchanged and do not participate in the reorganization. The $2f_1 + 1$ replicas with the highest behavioral scores and the f_1 replicas with the lowest behavioral scores are combined to form a new first group (if all replicas in a group have a behavioral score of 0, they need to be re-randomized). This process continues until all replicas in the summary table have been assigned groups. The behavioral score for each replica is reset to 0 and the additional score is reset to the table length minus the ordinal number, as illustrated in Figure 4.

The points table provides a mechanism for ranking replicas based on their historical behavior, allowing for a fair and informed selection of leaders in the view-change process. It ensures that replicas with a strong track record in successful consensus participation are given priority in the new table. With the points table returned by the leader, the replica can easily find its own group in the next view. One of the simplest ideas for leader selection is to determine the leader group leader under view v by v mod $m * n$, and the rest of the group leaders are determined by the intra group replica with the highest additional points. The group where the leader group leader belongs becomes the leader group.

We then define the types of messages additionally required for view changing:

View-change message. If replicas are unable to commit any blocks in view v before the timeout, they can send this message to the next view to change the leaders. A view-change message has the following format

$$\langle \text{view-change}, v+1, h_{est}, pt_r \rangle_r$$

where h_{est} denotes the highest valid block known to r, pt_r denote the points table of the replica. In fact, the view-change message has the same functionality as the vote message except for specifying a different view.

New-view message. Once the majority of replicas (i.e. exceeding 2/3 of the total number of replicas, because voting here is still based on replicas, rather than non-faulty groups) in view v send view-change messages to the leader group leader of the next view, it broadcasts this message to change view. A new-view message has the following format:

$$\langle \text{new-view}, v+1, \pi, B_{last}, pt \rangle_L$$

where π denotes the proof of enough view-change messages in the previous view. B_{last} denotes the last block the current leader wants to extend from. pt denotes the summary table.

6.2 Normal State Protocol

In Dynamic Group BFT, the state protocol is almost the same as the standard BFT protocol, with the main difference being that views do not change during the iterative execution, and leaders remain unchanged. Only a timer timeout will trigger an attempt to change the view. The remaining changes ensure that a non-faulty group contributes valid group votes for all valid blocks. Figure 5 shows the protocol process, highlighting the variations from the standard state protocol.

6.3 View-change Protocol

When the leaders of the current view is unable to drive the consensus protocol to increase the blockchain height

Let v be the current view number, replica L_0 be the leader group leader and set $\mathcal{L} = \{L_1, L_2, ...L_{m-1}\}$ represents the set of member group leaders of the current view. After sending a vote on block B_{k-1} to the group leader L_i $(0 \leq i \leq m-1)$, a replica r starts a timer and runs the following protocol.

Vote (round 1) and return (non-blocking). if replica r is the leader group leader L_0 or is a member group leader in set $\mathcal{L}$, upon receiving at least $2f_1 + 1$ partial signatures $\langle \mathsf{vote}, v, h_{k-1} \rangle_r$ from replicas within the group, combines them to produce a single signature σ, sends $\langle \mathsf{vote}, v, h_{k-1}, \sigma \rangle_{L_i}$ to L_0 and broadcasts $\langle \mathsf{return}, v, h_{k-1}, \sigma \rangle_{L_i}$ to all replicas within the group.

Propose (round 1). if replica r is the leader group leader L_0, upon receiving at least $2f_2 + 1$ signatures $\langle \mathsf{vote}, v, h_{k-1}, \sigma \rangle_{L_i}$ from group leaders, construct the aggregate signature $s^v_{k-1} = \sum \langle \mathsf{vote}, v, h_{k-1}, \sigma \rangle_{L_i}$, batch the transactions into a block B_k, and broadcast $\langle \mathsf{propose}, B_k, v, h_{k-1}, s^v_{k-1} \rangle_{L_0}$ to all group leaders (including itself) where $h_{k-1} = H(B_{k-1})$. If a sufficient number of valid votes are not collected before timeout, then L_0 will attempt to enter the view-change phase.

Propose (round 2). if replica r is the leader group leader L_0 or is a member group leader in set $\mathcal{L}$, after receiving a valid proposal $\langle \mathsf{propose}, B_k, v, h_{k-1}, s^v_{k-1} \rangle_{L_0}$ from L_0 (or receiving a valid forward of the proposal from a replica), sign it again and send $\langle \langle \mathsf{propose}, B_k, v, h_{k-1}, s^v_{k-1} \rangle_{L_0} \rangle_{L_i}$ to all replicas with in the group (including itself).

Vote (round 2). Before the timeout, upon receiving a valid proposal $\langle \langle \mathsf{propose}, B_k, v, h_{k-1}, s^v_{k-1} \rangle_{L_0} \rangle_{L_i}$ from L_i, the leader of the current group, where $h_{k-1} = H(B_{k-1})$, store h_{k-1} and s^v_{k-1} replacement to the already stored values (if the corresponding block of h_{k-1} is higher than the currently stored valid block), send $\langle \mathsf{vote}, v, h_k \rangle_r$ to the current group leader where $h_k = H(B_k)$, forward the proposal to to group leaders in $\mathcal{L}$ and reset the timer. Otherwise, try to enter the view-change phase after the timeout.

Commit. When receiving a valid block B_k and this block at the current height emerging for the first time, commit the block at height $k-2$.

Fig. 5. The normal state protocol of Dynamic Group BFT

Let L and L' be the leader group leaders of views v and $v+1$, respectively. While waiting for the next block at a higher height, a replica r with a local timer runs the following steps.

Lock and change view. If the local timer expires in view v, lock the current block by rejecting any message from L, and send $\langle \mathsf{view\text{-}change}, v+1, h_{est}, pt_r \rangle_r$ to L' while start a new local timer, w where h_{est} is the highest valid block known to r, pt_r denote the points table of the replica.

New view. The new leader group leader L' collects at least $\lfloor 2/3 * (m * n) \rfloor + 1$ view-change messages to construct a proof π, picks a referred block B_{last} in all those messages by the rank from height to hash value, and broadcasts $\langle \mathsf{new\text{-}view}, v+1, \pi, B_{last}, pt \rangle_L$, where pt denotes the summary table.

First vote. Upon receiving a valid new-view message from L', check if the contained block B_{last} has the rank equal to or higher than the locked block, convert the summary table to the initial points table, find the group leader in the initial points table for view $v+1$, send him $\langle \mathsf{vote}, v+1, H(B_{last}) \rangle_r$ and enter view $v+1$ by resetting the timer.

Fig. 6. The view-change protocol

within a specified timeframe, it triggers replicas to initiate a view change, expecting to reach consensus in the next view. In step i) of the view-change protocol, replicas take action after a timeout by locking the last block they voted on and sending a view-change message to the leader group leader of the next view. A new local timer is set to wait for a response. If the timer expires again without receiving a response, the replicas retain the locked block and request assistance from the next view of the current view. This process repeats until the necessary assistance is obtained. To prevent frequent view changes, the waiting time is increased after each failure in this step.

In step ii) of the protocol, the new leader group leader constructs a valid new-view message to initiate the view change. The summary table is merged from the points table sent by the replicas. By broadcasting the new-view message, the new leader group leader notifies replicas to move to the next view before the timeout.

In step iii), if the new-view message is valid, meaning the enclosed proof is verified and the rank of the referenced block is equal to or higher than the locked block, the replica converts the summary table to the initial points table, finds the group leader in the initial points table, proceeds to enter the new view by casting its first vote on the referenced block.

6.4 Proof of Liveness

Our proofs for cohesiveness, safety and responsiveness are applicable to Dynamic Group BFT. However, due to the introduction of additional view-changing protocols, we need to provide a new proof for liveness.

Lemma 15. *Let $\mathcal{R}_{correct}$ be the set of correct replicas and L^v_0 be the leader group leader in view v. If $|\mathcal{R}correct| > 2/3 \cdot |\mathcal{R}|$, then: (1) the view-change protocol proceeds correctly; (2) In a rotation cycle, the leader group leader is correct for a certain views.*

Proof. In Dynamic Group BFT, view change messages are sent directly to the leader group leader of the next view, without aggregation by member group leaders. This means:

- A view change succeeds when $> 2/3 \cdot |\mathcal{R}|$ replicas send view-change messages to the leader group leader of the next view.
- Since $|\mathcal{R}_{correct}| > 2/3 \cdot |\mathcal{R}|$, correct replicas alone can initiate and complete a view change.

Let us analyze the two possible cases when replicas attempt to change from view v to view $v+1$: (A) $L^{v+1}_0 \in \mathcal{R}_{correct}$. The replica completes the view change and enters $v+1$; (B) $L^{v+1}_0 \notin \mathcal{R}_{correct}$, the replica will fail the view change to $v+1$ and attempt to change to $v+2$ after the timer times out again. This process repeats until a view with a correct leader (Case A) is reached. Since leader selection rotates through all replicas, and $|\mathcal{R}correct| > 2/3|\mathcal{R}|$, Case A is bound to happen (the number of views of Case A is infinite). □

Lemma 16. *If $|\mathcal{R}_{correct}| > 2/3|\mathcal{R}|$, Case 1 will occur eventually.*

Proof. We analyze the system state in view v according to the four cases defined in Section 3.4.

(Case 1) if the system is already in Case 1 in view v, then the Lemma is satisfied.

By Lemma 15, if $|\mathcal{R}_{correct}| > 2/3|\mathcal{R}|$, then correct replicas can successfully perform view changes, and there exists

views with correct leader group leaders, i.e., correct replicas must be capable of transitioning to view v' ($v' > v$, and note that during this period, a failed view change does not cause the points table of a correct replica to be reset) in Case 2-4, where $L_0^{v'} \in \mathcal{R}_{correct}$.

(Case 2) Since the groups that are non-faulty in v remain non-faulty in v' (non-faulty groups must contribute valid group votes for all valid blocks) and $|\mathcal{G}_{non-faulty}| > 2f_2$ in Case 2 (where $\mathcal{G}_{non-faulty}$ is the set of non-faulty groups), $|\mathcal{G}_{non-faulty}| > 2f_2$ in v'. As $L_0^{v'} \in \mathcal{R}_{correct}$, Case 1 will occur in v'.

(Case 3 or 4) Similar to Case 2, the number of non-faulty groups in v' must be greater than or equal to that in v, i.e. $j' \geq j$ assuming j non-faulty groups in v and j' in v'. Firstly, the j groups that are non-faulty in v remain non-faulty in v' (since non-faulty groups must contribute valid group votes for all valid blocks). Secondly, groups that have contributed valid votes to some (but not all) valid blocks in the previous view accumulate scores for the $n_1 \geq 0$ correct replicas. These replicas will be ranked higher in the summary table and may result in $j_1 \geq 0$ more non-faulty groups after the view change (the compromised replicas can skip the bonus stage, but cannot arbitrarily falsify replica points (as illustrated in Section 6.1)). Furthermore, groups with similar replica behavioral scores (including groups where all replicas have a behavioral score of 0) should have their replicas ranked similarly, which are then re-randomized upon a view change. Such randomness allows for j_2 more non-faulty groups after reorganization, where $j_2 \geq 0$. If $j' = j + j_1 + j_2 < 2f_2 + 1$, then v' is still in Case 3 or 4. The above process will be repeated until there are j'' non-faulty groups in v'' such that $j'' \geq 2f_2 + 1$ and enters Case 1 or 2 (since j_1 and j_2 will not always be 0, one of the two cases must occur in finite views). Case 2 will go to Case 1.

So far, we have proved Case 1 will occur eventually. □

According to Lemma 16, Case 1 will eventually occur, satisfying the basic premise of Liveness. The proof process for the rest of the theorem is similar to that of Theorem 13, and will not be repeated here.

Once the system enters Case 1, it will continue to run in Case 1 in the same view unless network conditions cause delays exceeding the timeout. The view-change protocol won't be triggered even if compromised replicas continuously send view-change messages to the next view's leader group leader, because it requires more than 2/3 of the total number of replicas to send view-change messages to produce a valid view-change proof. However, before Case 1 happens, compromised replicas can attempt to constantly force the protocol into expensive view-change processes (through timeouts) to delay the occurrence of Case 1. This requires them to collude to maximize their behavioral scores in the points table, otherwise they would be ranked at the bottom of the summary table after only a few view changes and consensus would quickly enter Case 1. Examples of such advanced adversarial behaviour include:

(1) All compromised replicas know the identities of other compromised replicas.
(2) (During normal operation) if a group leader is compromised, it colludes with compromised replicas within its group to select as many compromised replicas' votes as possible to construct the in-group proof. It sends the group vote containing the in-group proof back to replicas within the group during the return phase but doesn't (contribute) send it to the consensus leader (accumulating points while preventing consensus and attempting to trigger view changes). Note that correct replicas consistently earn points, as it requires at least $2f_1 + 1$ valid votes to construct a valid in-group voting proof to enter the bonus stage.
(3) (During normal operation) if the group leader isn't compromised, compromised replicas don't respond (responding would push consensus forward, contrary to their goal).
(4) (During view changes) if a compromised replica is the new leader group leader, it attempts to maximize compromised replicas' behavioral scores in the next view's initial points table (in a manner analogous to its actions in (2)), tries to achieve consensus once to make the new table effective (note that such behavior may also contribute to the protocol's liveness concurrently), then causes timeouts to trigger view change.

Such behavioral pattern is most effective before Case 1 occurs, as afterward, with correct leaders in place, Dynamic Group BFT becomes resistant to such attacks.

6.5 Complexity and Tolerance

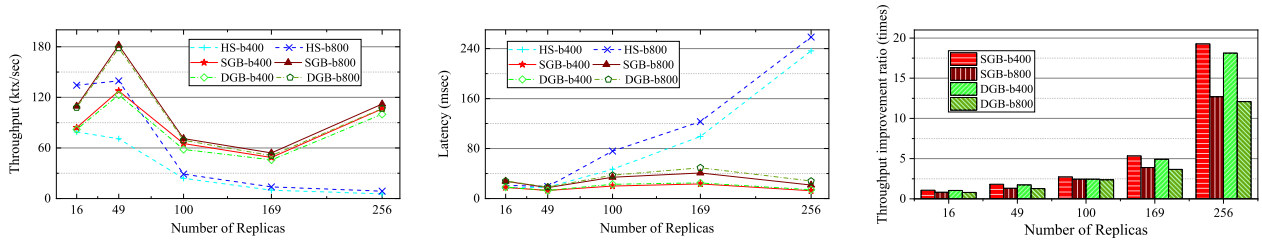
Dynamic Group BFT has the communication complexity of $O(m^2 * n)$ when there are $m * n$ replicas in total, and in practice it is slower than Standard Group BFT, especially when the number of replicas is large, due to the introduction of the additional points table and extra forwarding. However, Dynamic Group BFT offers improved stability in scenarios involving compromised replicas by maximizing the number of non-faulty groups and facilitating regrouping when the current grouping scheme fails to reach consensus. During group reorganization in Dynamic Group BFT, communication occurs at the replica level rather than the group level. As a result, Dynamic Group BFT maintains the same tolerance as the traditional BFT protocol, which is not exceeding 1/3 of compromised replicas. However, if the aim is to maintain the tolerance level of Standard Group BFT, it becomes necessary for all replicas to determine their respective groups before entering the new view phase. This presents an avenue for further exploration and investigation.

7 Experiments

We implemented Group BFT in about 4K lines of Rust code, of which the core consensus is about 600 lines. We first describe the implementation details and the experimental setup. We then examine the baseline scalability performance as the system size increases by comparing to a state-of-art system, HotStuff. We also evaluate the performance of the two protocols under various compromised replicas. Furthermore, we compare the impact of different grouping methods on Group BFT performance.

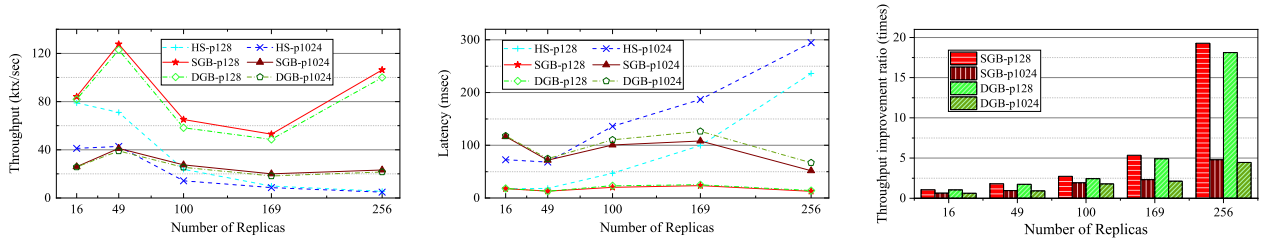
7.1 Setup

We conducted our experiments on a cluster of up to 16 Aliyun's Cloud Elastic Compute Service (ECS) u1-c1m2.4xlarge instances, each with 16 vCPUs and 32GB of



(a) Throughput (b) Latency (c) Throughput improvement ratio

Fig. 7. Scalability for batch size of 400 or 800, with payload of 128-byte and average RTT = 6.4ms.



(a) Throughput (b) Latency (c) Throughput improvement ratio

Fig. 8. Scalability for 128/128 payload or 1024/1024 payload, with batch size of 400 and average RTT = 6.4ms.

memory, running Ubuntu 22.04 LTS and deployed rustc 1.67.0-nightly. All replicas are evenly distributed across instances and connect others to build a consensus network.

Our prototype implementation of Group BFT uses the Ed25519[1], BLAKE3[2] for cryptographic operations (e.g. digital signatures), and utilises TCP connections for communication between replicas using Tokio library. Furthermore, our client is developed as an independent single-threaded process that transmits transactions to replicas at a specific frequency. Our replicas can retrieve configuration information from JSON files and command line arguments, and export statistics as JSON files for subsequent analysis. We have developed a module called CONFIG-GEN that automates the creation of configuration files for a range of experiments.

7.2 Performance Benchmarking

Our evaluation primarily focused on latency and throughput as key performance indicators. Each reported result is the average from 10 repeated trials, with each trial's result taken after the throughput stabilized. Latency was calculated from the time a client initiated a transaction to the time it was committed by the protocol. Throughput was measured as the number of committed transactions per second. We obtained the results presented in this section by adjusting the client sending rate from underload to overload conditions. All results correspond to the minimum delay at the maximum throughput under specific experimental conditions. The labels HS, SGB, and DGB represent the experimental results for HotStuff, Standard Group BFT, and Dynamic Group BFT, respectively.

We set the replica count from 16 to 256, with the total number of replicas equating to 16, 49, 100, 169, 256 ($f = 5, 16, 33, 56, 85$) for HotStuff and $m = n = 4, 7, 10, 13, 16$ ($f_1 = f_2 = 1, 2, 3, 4, 5$) for Group BFT [3]. These experiments were conducted under varying batch sizes (400 or 800), payload sizes (128 or 1024 bytes)[4], and additional network delays (5ms or 10ms) [5].

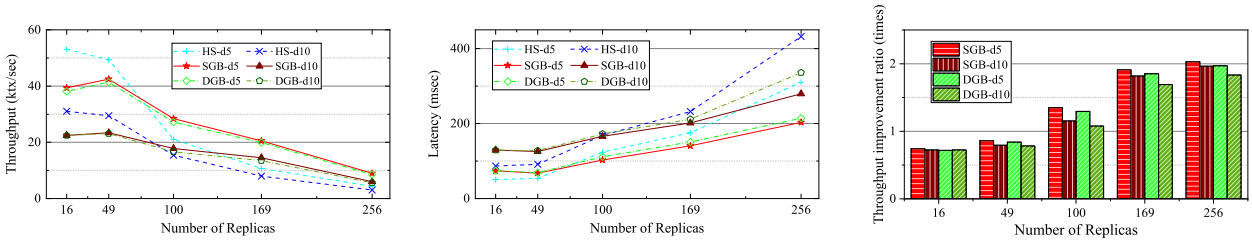
Figures 7(a) and 7(b) illustrate the first experimental setting, representing throughput and latency, respectively. Group BFT showed similar throughput to HotStuff with 16 replicas (with a batch size of 400, denoted as "b400"), and even has lower throughput and higher latency when the batch size is 800 (denoted as "b800"). However, as the replica count escalates, HotStuff's performance degrades at a more rapid pace compared to Group BFT, leading to inferior throughput and higher latency for replica counts of 49, 100, 169 and 256. Under the current experimental parameters, Group BFT's performance not only fails to decrease

1. https://github.com/dalek-cryptography/ed25519-dalek
2. https://github.com/BLAKE3-team/BLAKE3

3. We selected this range to provide sufficient data points for analyzing scaling behavior in BFT systems. The lower bound of 16 replicas represents a threshold beyond which system throughput begins to decrease rapidly as replica count increases, as observed in other works [5], [31]. The upper bound of 256 replicas was chosen because it approximately doubles the typical upper bounds settings (usually limited to around 50-150 replicas) in most BFT literature [5], [11], [13], [31], [34]. Since our work specifically targets large-scale BFTs, this extended upper bound allows us to better illustrate scaling properties. Additionally, these values map cleanly to Group BFT's configuration as square numbers, allowing balanced grouping with $m = n$.

4. The batch size and payload size selections are based on commonly used values in BFT literature (e.g., HotStuff [5], the protocol under comparison). We selected batch size 400 as our moderate baseline, while batch size 800 tests the system under higher batch processing demands. The 128-byte payload represents small transactions typical of cryptocurrency transfers or simple state updates (more commonly used), while the 1024-byte payload represents larger transactions such as complex smart contract [44] interactions or data-intensive operations.

5. Typical BFT deployment scenarios (e.g. consortium blockchains) generally operate within national boundaries, making a maximum additional network delay of 10ms sufficient to model most real-world scenarios. The 5ms additional delay approximates a medium geographic distribution of replicas, while the 0ms additional delay (where RTT averages 6.4ms) accurately reflects the inherent network conditions in our geographically proximity clustered server environment (with no artificial delay added).



(a) Throughput (b) Latency (c) Throughput improvement ratio

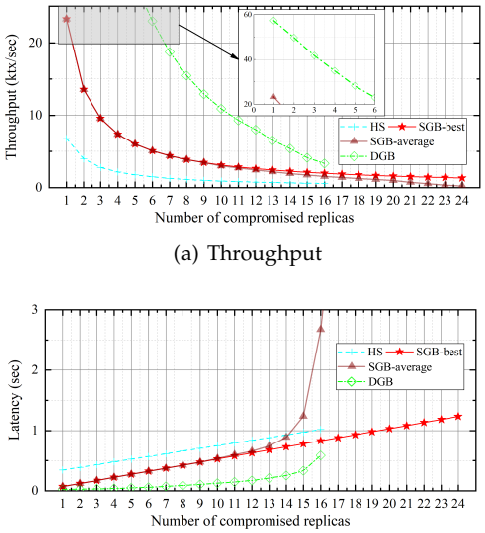
Fig. 9. Scalability for additional network delay 5ms (RTT = 13ms) or 10ms (RTT = 25ms), with 128/128 payload, batch size of 400.

monotonically with growing numbers of replicas, but it even experiences an upsurge at replica numbers 49 and 256. Moreover, the throughput improvement ratio compared to HotStuff escalates with the total number of replicas, reaching nearly 20 times with 256 replicas, as shown in Figure 7(c). The performance difference between Dynamic Group BFT and Standard Group BFT remains minimal, primarily because their state protocols are nearly identical. Without engaging the view-change protocol, Dynamic Group BFT incurs a minor additional overhead due to the extra forwarding and maintenance of point tables by the replicas.

The second experimental setup, with payload sizes of 128 or 1024 bytes, is denoted as "p128" or "p1024" in Figures 8(a) and 8(b) representing throughput and latency. The performance pattern mirrors the first setting when the payload size is 128 bytes. However, with a payload size of 1024 bytes, HotStuff and Group BFT showed similar throughput and latency with 49 replicas and the significant advantage was not evident until the number of replicas reached 100 and more. This could be attributed to larger payloads intensifying the intergroup communication overhead of Group BFTs. As a result, with a payload size of 1024 bytes, Group BFT only exhibits a throughput improvement of around 5 times over HotStuff with 256 replicas, as shown in Figures 8(c).

The third experimental configuration, with additional network delays of 5ms or 10ms, is represented as "d5" or "d10" in Figures 9(a) and 9(b) illustrating throughput and latency, respectively. We leveraged the Linux traffic control tool "tc" to simulate network delay on each instance. As network latency increased, Group BFT gradually lost its performance advantage in smaller scale systems. With a network delay of 5ms, Group BFT still trailed HotStuff with 49 replicas. With a network delay of 10ms, Group BFT exhibited similar latency to HotStuff with 100 replicas and only slightly outperformed it in terms of throughput. This suggests that an increase in network delay has a larger impact on Group BFT's performance. The results in Figure 9(c) also indicate that when subjected to a latency of either 5ms or 10ms, the enhancement ratio of Group BFT's throughput dwindles to about 2x in comparison to HotStuff.

These experiments collectively highlight the throughput and latency advantages of Group BFT in large-scale systems, effectively mitigating the processing bottleneck typically experienced by traditional leaders. As depicted in Figures 7(c), 8(c), 9(c), these advantages are progressively significant under various conditions with an increase in the system's size However, it's important to note that Group BFT does not offer advantages in smaller systems and is more vulnerable to network latency or load size. Furthermore, the performance gap between Standard Group BFT and Dynamic Group BFT widens as the number of replicas increases - Dynamic Group BFT does not offer an advantage when the system is free of compromised replicas.



(a) Throughput

(b) Latency

Fig. 10. Performance as fault count increases, with 128/128 payload, batch size of 400, 49 replicas and average RTT = 6.4ms.

7.3 Performance under Faults

This section explores the system performance as the number of compromised replicas escalates, under the conditions of a total of 49 replicas, a payload size of 128 bytes, a batch size of 400, and an addtional network delay of 0ms, as represented in Figures 10(a) and 10(b). For this experiment, we have set the timer timeout at 2.5s and have prescribed a delay of 2.5x49s when consensus cannot be achieved. An extra 490 repetitions (i.e., a total of 500 sets) were scripted for each data set in the case of Group BFT. For Dynamic Group BFT, we measured the average throughput when compromised replicas collude to maximize their behavioral scores and delay the occurrence of Case 1 (as specified in Section 6.4). The measurement period is the time it takes for the system to enter Case 1 and for the throughput to subsequently stabilize. As the other two protocols do not utilize the scoring mechanism, the compromised replicas behavior is still simulated as a denial of response (given that collusion does not extend the negative influence further),

and since the leaders are rotated, the performance exhibits periodicity, and the performance is measured over a cycle. Here, "SGB-best" denotes the best-case scenario and "SGB-average" represents the average outcome of the 500 trials.

As observed, the best-case fault tolerance for the Standard Group BFT protocol aligns with our theoretical analysis in Section 5.5, standing at $(2f_1 + 1)(2f_2 + 1) - 1 = 24$. Conversely, HotStuff and Dynamic Group BFT can only tolerate up to 1/3 of the compromised replicas (16 in this experiment). Additionally, with faults present, Group BFT demonstrates higher throughput and lower than HotStuff, especially in the case of dynamic group BFTs, which even surpass Standard Group BFTs significantly. The mechanism responsible for exchanging dynamic group members undoubtedly enhances the capability for failure recovery, although the infrequent leadership rotation of Dynamic Group BFT after Case 1 may also contribute to this performance improvement. But it is also evident that, as the number of compromised replicas rises, so does the impact of their collusion, as evidenced by Figure 10(a), which demonstrate a significant decline in the dynamic protocol's average throughput during the measurement period. Since we measure over time $t + t'$ (where t is the time to reach Case 1; t' is the time for throughput to stabilize subsequent to Case 1, and q is the total throughput after t until $t + t'$ (both t' and q exhibit minimal fluctuations), the average throughput can be calculated as $(k_1 t + q)/(t + t')$ = $(k_1 t/t' + q/t')/(t/t' + 1)$, where k_1 represents the average throughput during the first interval t and $k_1 << q/t'$. The declining performance shown in the Figure indicates that t increases substantially as the number of compromised replicas increases, meaning that during the first interval t, the system experiences more timeouts, triggering more view-change protocols and reaching consensus less frequently.

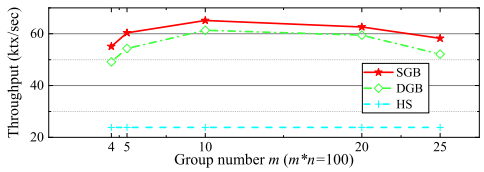
While "SGB-Best" shows better fault tolerance, it may not always be achievable. This is because, due to the randomness of the Standard Group BFT grouping, consensus failure can result from compromised replicas exceeding the number of $(f_1 + 1)(f_2 + 1) = 9$ under certain grouping methods. For greater stability, implementing Dynamic Group BFT is advisable in systems with compromised replicas.

7.4 Impact of m and n

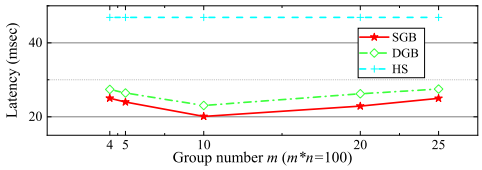
We further explored how different grouping methods (i.e., different m, n) impact system performance. Experiments were conducted with the total number of replicas $mn = 100$. The results are displayed in Figures 11(a) and 11(b), which indicate that the system performance initially rises and subsequently declines as the group count increases. When the total number of replicas is fixed, the performance of the system is optimised when m and n are closely matched.

8 Conclusions

We study Group BFT protocols, which utilizes a two-round message transmission by dividing the entire consensus network into groups. We establish a formal system model for the Group BFT protocols with a versatile set of base components and explicit definitions, addressing the challenges inherent in designing such a system. We further design and implement two highly efficient Group BFT protocols: one that supports inter-group member exchange and another that does not. We show that the Group BFT protocols significantly alleviate the processing bottlenecks of the leader and highly improve throughput in large-scale systems.



(a) Throughput



(b) Latency

Fig. 11. Performance with different m, n choices, with 128/128 payload, batch size of 400, 100 replicas and average RTT = 6.4ms..

While this work primarily focuses on optimizing voting and signature aggregation processes by using a hierarchical group structure, future work could extend it to also optimize block dissemination (rather than just forwarding the block), a process which may present an even greater challenge in terms of network bottlenecks at leaders. Specifically, integrating techniques for block encoding and chunking could further balance the incurred communication load among replicas and alleviate leader bottlenecks, though careful design would be needed to consider the traditional trade-offs in such approaches - particularly the all-to-all messaging requirements in the increased communication diameter when proposing. This would create a more comprehensive solution, potentially yielding greater scalability benefits for large-scale BFT systems.

Acknowledgements

This paper was supported in part by the National Natural Science Foundation of China (NSFC) under Grants U2468205, 62372149 and U23A20303, and in part by the China Scholarship Council (CSC), and in part by the Key Laboratory of Knowledge Engineering with Big Data (the Ministry of Education of China) under Grant BigKEOpen2025-04.

References

[1] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the works of leslie lamport*, 2019, pp. 203–226.

[2] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[3] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, 2016.

[4] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget." [Online]. Available: http://arxiv.org/abs/1710.09437

[5] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[6] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," *The Libra Assn., Tech. Rep*, vol. 1, no. 1, 2019.

[7] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang, "Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback," in *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*. Springer, 2022, pp. 296–315.

[8] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, "Damysus: streamlined bft consensus leveraging trusted components," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 1–16.

[9] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is dag," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021, pp. 165–175.

[10] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 34–50.

[11] X. Liu, K. Feng, Z. Zhang, M. Li, X. Chen, W. Lai, and L. Zhu, "Dolphin: Efficient non-blocking consensus via concurrent block generation," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 11 824–11 838, 2024.

[12] F. M. Benčić and I. P. Žarko, "Distributed ledger technology: Blockchain compared to directed acyclic graph," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1569–1570.

[13] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: Dag bft protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2705–2718.

[14] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 931–948.

[15] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 international conference on management of data*, 2019, pp. 123–140.

[16] V. King, J. Saia, V. Sanwalani, and E. Vee, "Scalable leader election," in *SODA*, vol. 6, 2006, pp. 990–999.

[17] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage, "Steward: Scaling byzantine fault-tolerant replication to wide area networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 1, pp. 80–93, 2008.

[18] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, "Resilientdb: global scale resilient blockchain fabric," *Proceedings of the VLDB Endowment*, vol. 13, no. 6, pp. 868–883, 2020.

[19] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 35–48.

[20] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 106–118.

[21] Z. Zhang, X. Liu, M. Li, H. Yin, L. Zhu, B. Khoussainov, and K. Gai, "Hca: Hashchain-based consensus acceleration via revoting," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[22] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1187–1201.

[23] M. Li, Y. Chen, C. Lal, M. Conti, M. Alazab, and D. Hu, "Eunomia: Anonymous and secure vehicular digital forensics based on blockchain," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 225–241, 2021.

[24] J. Xu, C. Wang, and X. Jia, "A survey of blockchain consensus protocols," *ACM Computing Surveys*, 2023.

[25] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th usenix security symposium (usenix security 16)*, 2016, pp. 279–296.

[26] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[27] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.

[28] P. Feldman and S. Micali, "Optimal algorithms for byzantine agreement," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, 1988, pp. 148–161.

[29] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[30] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*.

[31] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase bft with linearity," *Cryptology ePrint Archive*, 2022. [Online]. Available: https://eprint.iacr.org/2022/551

[32] G. Xu, H. Bai, J. Xing, T. Luo, N. N. Xiong, X. Cheng, S. Liu, and X. Zheng, "Sg-pbft: A secure and highly efficient distributed blockchain pbft consensus algorithm for intelligent internet of vehicles," *Journal of Parallel and Distributed Computing*, vol. 164, pp. 1–11, 2022.

[33] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, "A scalable multi-layer pbft consensus for blockchain," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1146–1160, 2020.

[34] H. Xu, X. Liu, C. Zhang, W. Wang, J. Wang, and K. Li, "Crackle: A fast sector-based bft consensus with sublinear communication complexity," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 1–10.

[35] I. Kaklamanis, L. Yang, and M. Alizadeh, "Poster: Coded broadcast for scalable leader-based bft consensus," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3375–3377.

[36] C.-D. Liu-Zhang, C. Matt, and S. E. Thomsen, "Asymptotically optimal message dissemination with applications to blockchains," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 64–95.

[37] A. Gągol, D. Leśniak, D. Straszak, and M. Świętek, "Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 214–228.

[38] X. Dai, Z. Zhang, Z. Guo, C. Ding, J. Xiao, X. Xie, R. Hao, and H. Jin, "Wahoo: A dag-based bft consensus with low latency and low communication overhead," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 7508–7522, 2024.

[39] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[40] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[41] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*. Springer, 2000, pp. 207–220.

[42] D. Abram, A. Nof, C. Orlandi, P. Scholl, and O. Shlomovits, "Low-bandwidth threshold ecdsa via pseudorandom correlation generators," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2554–2572.

[43] T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, and D. Schröder, "Roast: Robust asynchronous schnorr threshold signatures," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2551–2564.

[44] M. Li, Y. Chen, C. Lal, M. Conti, F. Martinelli, and M. Alazab, "Nereus: Anonymous and secure ride-hailing service based on private smart contracts," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2849–2866, 2022.