Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.040600

ARTICLE

CF-BFT: A Dual-Mode Byzantine Fault-Tolerant Protocol Based on Node Authentication

Zhiruo Zhang, Feng Wang*, Yang Liu, Yang Lu and Xinlei Liu

Department of Computer Technology, Henan University of Technology, Zhengzhou, 450001, China

*Corresponding Author: Feng Wang. Email: wfmail@sina.com

Received: 24 March 2023 Accepted: 07 June 2023 Published: 08 October 2023

ABSTRACT

The consensus protocol is one of the core technologies in blockchain, which plays a crucial role in ensuring the block generation rate, consistency, and safety of the blockchain system. Blockchain systems mainly adopt the Byzantine Fault Tolerance (BFT) protocol, which often suffers from slow consensus speed and high communication consumption to prevent Byzantine nodes from disrupting the consensus. In this paper, this paper proposes a new dual-mode consensus protocol based on node identity authentication. It divides the consensus process into two subprotocols: Check_BFT and Fast_BFT. In Check_BFT, the replicas authenticate the primary's identity by monitoring its behaviors. First, assume that the system is in a pessimistic environment, Check_BFT protocol detects whether the current environment is safe and whether the primary is an honest node; Enter the fast consensus stage after confirming the environmental safety, and implement Fast_BFT protocol. It is assumed that there are $3f + 1$ nodes in total. If more than $2f + 1$ nodes identify that the primary is honest, it will enter the Fast_BFT process. In Fast_BFT, the primary is allowed to handle transactions alone, and the replicas can only receive the messages sent by the primary. The experimental results show that the CF-BFT protocol significantly reduces the communication overhead and improves the throughput and scalability of the consensus protocol. Compared with the SAZyzz protocol, the throughput is increased by 3 times in the best case and 60% in the worst case.

KEYWORDS

Blockchain; consensus protocol; dual-mode; Byzantine fault tolerance; distributed system

1 Introduction

Early consensus protocols focused on ensuring data and operation consistency in distributed systems, resulting in mainly Crash Fault Tolerance (CFT) protocols [1]. However, the emergence of cryptocurrencies such as Bitcoin [2] in 2008 brought attention to the decentralized and tamper-proof features of blockchain technology, leading to its application in federated learning [3], data sharing [4], and other various fields [5–13]. At the same time, Byzantine Fault Tolerance (BFT) protocol [14] has gradually become the focus of consensus protocol research, which cannot only tolerate network

delays, node failures, and crashes but also handle Byzantine nodes intentionally disrupting system consistency.

Hence, BFT protocols make different "sacrifices" to prevent attacks, which can result in issues such as low consensus performance and high resource overhead. Proof of Work (PoW) uses computational competition for ledger rights and rewards to prevent malicious behavior and blockchain fork problems, but it results in excessive resource consumption. Proof of Stake (PoS) [15], on the other hand, uses stake competition and "Coin Days" to measure node rights but sacrifices fair competition and decentralization characteristics of the blockchain. Practical Byzantine Fault Tolerance (PBFT) protocol [16] uses cryptographic technology and mutual voting to maintain consistency, but its communication complexity and high demand have limited its scalability.

To solve these issues, this paper presents a dual-mode consensus protocol based on node identity authentication. The protocol divides the consensus process into two subprotocols: Check_BFT and Fast_BFT. By monitoring the behavior of the primary in Check_BFT to verify the primary's identity, the Fast_BFT protocol is initiated if appropriate to allow the primary node to handle transactions alone. The protocol greatly improves consensus efficiency and reduces communication costs without relying on extensive resource consumption, while ensuring consistency and security. The particular contributions of this paper are:

- This paper introduces a node identity verification process and divides the protocol into two modes: preparatory mode Check_BFT subprotocol and fast consensus mode Fast_BFT subprotocol, which improves the performance and throughput of the protocol.
- In Check_BFT, this paper introduces an information verification process that allows all nodes to monitor the behavior of the primary to determine if its identity is honest without requiring client participation.
- Based on the authentication result of the primary, the Check_BFT determines whether to enter the Fast_BFT. There is no need to perform the regular view change. Which reduces communication consumption.
- In CF-BFT, this paper introduces two useful mechanisms. One is the node reputation mechanism and the other is the fast mode consensus number random mechanism. Which ensures the safety and decentralization of the blockchain effectively.

The rest sections of this paper are organized as follows: The related work is discussed in Section 2; the overall design and implementation of CF-BFT are presented in detail in Section 3; the safety, consistency, and communication complexity of CF-BFT are comprehensively analyzed in Section 4; the performance evaluation and comparison results of CF-BFT are discussed in Section 5; the conclusion and prospect are concluded in Section 6.

2 Related Work

Most BFT protocols [17,18] have high complexity because they need to maintain consistency in the worst condition. However, the blockchain is not always in the worst condition. Because BFT consensus is based on the primary, some scholars have proposed improved methods based on primary selection and primary authentication. These protocols improve performance by avoiding selecting Byzantine nodes as primary. For example, Algorand [19], IBFT [20], Prosecurtor [21], and other protocols have added primary selection algorithms to select the honest node as primary by calculating the reputation of nodes. Another type of improvement method, such as Thunderella [22], Trust [23], and other protocols, improves protocol performance by monitoring primary behavior to identify

whether nodes are honest or malicious. Thunderella monitors primary by the clients, and if there are problems with transactions, the client will report errors to the blockchain system and send the transaction information they sent. The blockchain system judges whether the primary is an honest node by analyzing the messages sent by the client. Trust builds a blockchain data structure or third-party that filters the malicious behavior of Byzantine nodes and improves protocol performance. The characteristics of some representative protocols are summarized in Table 1.

Table 1: Comparison of BFT-based consensus protocols

Protocol	Adversadry tolerate	Throughput	Latency
PBFT	$f < n/3$	Low	High
HotStuff	$f < n/3$	Medium	Low
BFT-SMaRt	$f < n/3$	Low	High
Algorand	$f < n/3$	Medium	Medium
IBFT	$f < n/3$	N/A	Low
Prosecutor	$f < n/3$	Medium	Low
TrustBlock	$f < n/3$	High	Low

One kind of method for improving consensus efficiency is to use dual-mode or multi-mode protocols. These protocols have two subprotocols typically: a fast subprotocol for optimistic conditions to increase performance and a backup subprotocol for extreme conditions to ensure safety and consistency. The characteristics of dual-mode protocols are summarized in Table 2. CheapBFT [24] consists of three subprotocols: the normal case protocol CheapTiny, the transition protocol CheapSwitch, and the fallback protocol MinBFT. During optimistic conditions, CheapTiny only requires f + 1 active replicas to participate. Thunderella selects a committee and a primary called the Accelerator to quickly process messages if more than 3/4 nodes of the committee and primary are honest. Zyzzyva [25] assumes an optimistic condition but requires all nodes to be honest, with clients helping to converge to the total order of requests if nodes are faulty. AZyzzyva [26] separates the "fast path" and "slow path" of Zyzzyva using PANIC messages sent to replicas instead of "commit-certificate" messages sent to clients. SAZyzz [27] enhances scalability and reduces the communication complexity of both modes to $O(\log n)$ through a tree-based communication model, and SBFT [28] divides nodes into primary, commit collectors, and execution collectors, different sets can handle different phases without mutual voting.

The aforementioned dual-mode protocols are proposed based on the number of honest nodes in the system and the identity of nodes that assume certain roles. In contrast, protocols such as Bolt-Dumbo [29], Jolteon-Ditto [30], and Flexico [31] are proposed based on synchronous and asynchronous environments. When the designated nodes are in a synchronous or weakly synchronous state, the fast protocol is used. When nodes are in an asynchronous environment, the backup protocol takes over the blockchain system to ensure safety. The Flexico protocol divides nodes into active nodes and passive nodes while setting a weakly synchronous network condition, where most active nodes can communicate within a known upper bound.

Table 2: Comparison of dual-mode consensus protocols

Protocol	Fast-mode		Backup-mode		View change
	Adversadry tolerate	Communication complexity	Adversadry tolerate	Communication complexity	
CheapBFT	$f = 0$	$O(n^2)$	$f < n/3$	$O(n^2)$	√
Thuderella	$f < n/4$	N/A	$f < n/3$	$O(n^2)$	√
Zyzzyva	$f = 0$	$O(n)$	$f < n/3$	$O(n^2)$	√
Azyzzyva	$f = 0$	$O(n)$	$f < n/3$	$O(n^2)$	√
SAZyzz	$f = 0$	$O(logn)$	$f < n/3$	$O(logn)$	√
SBFT	$f = 0$	$O(n)$	$f < n/3$	$O(n)$	√
Bolt-dumbo	$f < n/3$	$O(n)$	$f < n/3$	$O(n^2)$	√
Jolten-ditto	$f < n/3$	$O(n^2)$	$f < n/3$	$O(n^2)$	√
Flexico	$f < n/3$	$O(n)$	$f < n/3$	$O(n^2)$	×

In conclusion, most consensus protocols have two directions for improvement. One approach is to enhance performance by increasing the selection algorithm for primary, which avoids selecting Byzantine nodes as primary. The other approach is to invoke appropriate subprotocols under different conditions, which ensures both security and consistency while improving protocol performance.

3 CF-BFT Protocol

This section introduces the CF-BFT protocol model, which is a dual-mode protocol based on node identity authentication and describes in detail the execution methods of the protocol in optimistic and pessimistic conditions.

3.1 CF-BFT Protocol Overview

CF-BFT protocol is divided into the Check_BFT subprotocol for the preparation and Fast_BFT subprotocol for the fast consensus, as shown in Fig. 1. In Check_BFT, all nodes perform message computation and processing through mutual voting. At the same time, replicas monitor the behavior of the primary and detect whether the primary has any behavior that violates consistency or reduces protocol efficiency. In Fast_BFT, only the primary needs to process messages, and replicas only need to passively replicate messages transmitted by the primary. Thereby improving the protocol's performance and reducing communication overhead.

This paper assumes that CF-BFT consists of $3f+1$ nodes, and the adversaries can control f nodes at most. Each node has a key pair (including a public key and a private key) and a data structure to record the address, reputation of nodes, and other local data. Adversaries can control and coordinate malicious nodes to perform any destructive behavior that affects system consistency. As shown in Fig. 2, consensus can be achieved using the Check_BFT or Fast_BFT protocol.

Firstly, after selecting the primary, the protocol initiates a round of message consensus which is transmitted by the client and performs voting calculations in conjunction with the replicas. When $2f+1$ nodes reach consensus, the message is considered to reply and enters the primary authentication phase. In the consensus phase, all replicas monitor the behavior of the primary. Subsequently, the

primary initiates authentication voting. If the primary does not engage in malicious behavior that violates system consistency, the replicas will provide support votes. Otherwise, the replicas will cast opposing votes.

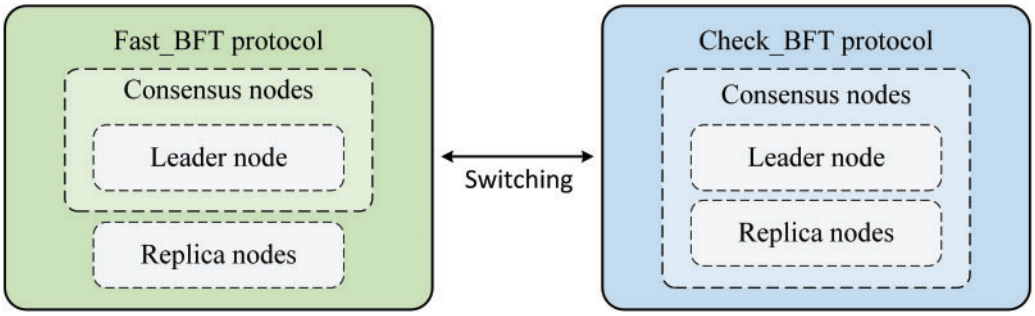

Figure 1: Consensus nodes for Fast_BFT and Check_BFT protocols

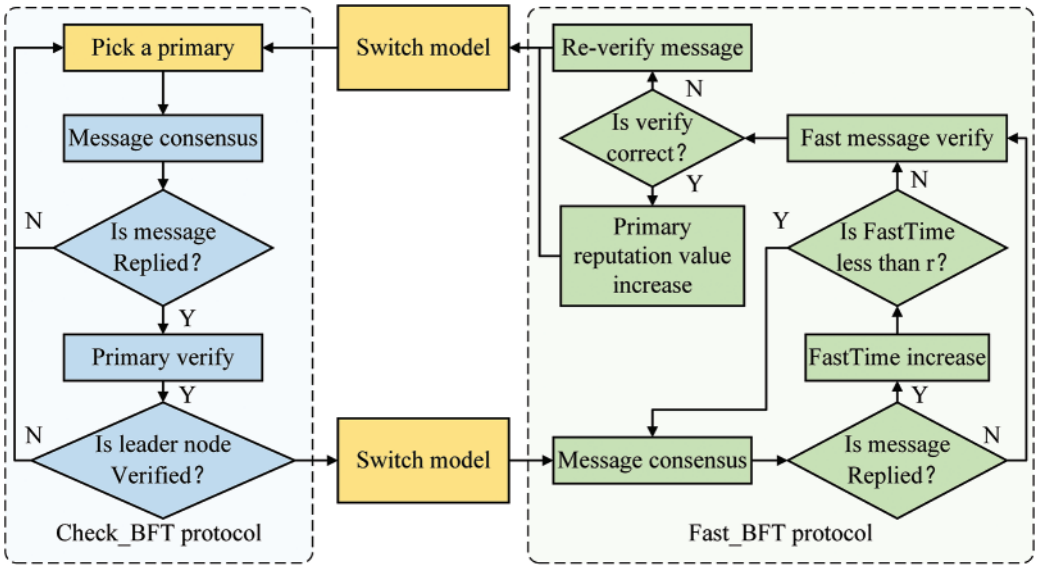

Figure 2: Overall operation of CF_BFT dual-mode protocol

When the primary receives support votes from $2f + 1$ nodes, CF-BFT begins the fast consensus phase and runs the Fast_BFT subprotocol. At this point, the protocol calculates a random number r, to determine the number of times the current primary can receive messages from the client. During fast consensus, only the primary needs to process messages, while other nodes passively replicate the messages sent by the primary. Each time a fast consensus is carried out, the number of consensuses, *FastTime*, increases, and when the *FastTime* reaches the specified upper limit, Fast_BFT ends. Subsequently, all nodes calculate the hash value of all messages in Fast_BFT and vote mutually to authenticate the result. If the result is consistent, the reputation value of the primary increases and the protocol switches to Check_BFT. Else, the most recent r messages among honest nodes are converged.

3.2 CF-BFT Protocol Overview

Check_BFT is a distributed consistency protocol based on Byzantine fault tolerance technology. The notations used are summarized in Table 3. The total number of nodes is $n = 3f + 1$, and it can tolerate f Byzantine nodes. Check_BFT is divided into four phases. The phases are *Prepare*, *Test*, *Commit*, and *Verify*, as shown in Fig. 3. In this protocol, replicas determine whether the primary is an honest node by monitoring its behavior.

Table 3: Notations of symbols

Symbol	Description
c	*Client*
n	*Number*
m	*Message*
i	*Number of nodes*
$< m >_{\sigma_i}$	*Message signed by node i*
o	*Operation*
v	*View*
t	*Timestamp*
s	*Result*
p	*Primary*
d	*Digest*
r	*Random*
FastTime	*Number of Fast_BFT messages replied*

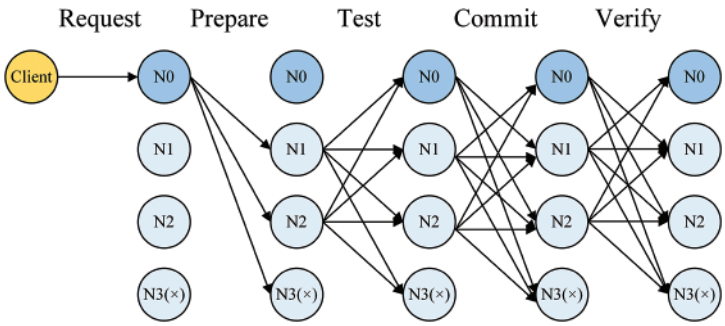

Figure 3: Check_BFT operation

Specifically, the protocol first selects a primary p = vmodn. The client c sends a $< REQUEST, o, t, c >_{\sigma_c}$ to the current primary p. Which contains the message's content and operation. The primary response to the client with a $< REPLY, v, t, c, i, s >_{\sigma_i}$ message. If the client does not receive *REPLY* message, it will send the *REQUEST* message to all nodes. And the nodes verify switching the primary or not.

In *Prepare* phase, the primary node p broadcasts the *PREPARE* message $<< PREPARE, v, n, d >_{\sigma_p}, m >$ to replicas after signing and encrypting the client message. If the primary fails to broadcast and

reply to the message in the specified time, the replicas consider it a Byzantine node and switch to electing a new primary.

In *Test* phase, the replicas receive and validate the *PREPARE* message sent by the primary. If the validation is successful, they broadcast a *TEST* message $< TEST, v, n, d, i >_{\sigma_i}$ to other replicas, and store the message in a temporary message pool. All nodes receive the *TEST* message and validate it. If a replica receives the *PREPARE* conflict message transmitted by the primary or receives the conflict message from $f+1$ or more replicas, it immediately packages it as evidence and sends it to other nodes. When the replicas verify that the primary is a Byzantine node, the protocol switches to a new view. If a replica receives $2f+1$ *TEST* messages with the same digest within the specified time, it enters the *Commit* phase.

In *Commit* phase, the primary and all replicas participate in message submission and validation. If nodes consider the message to have passed the *Test* validation, it sends a $< COMMIT, v, n, D(m), i >_{\sigma_i}$ message to other nodes and receives *COMMIT* messages from others. When a node receives $2f+1$ *COMMIT* messages, it writes the message to the local log. In *Verify* phase, the replicas still judge the behavior of the primary to determine whether it is an honest node, thus ensuring the consistency of the protocol.

In addition, the protocol introduces a random timeout mechanism to prevent the failure of the primary from disrupting the replication service. During the execution of each phase, the primary needs to complete the operation within a specified time. Otherwise, the replicas will consider it a faulty node and begin broadcasting to elect a new primary. This mechanism helps prevent such malicious behavior of the primary as the coordinating of faulty nodes or delayed communication. Which can disrupt the service's safety and consistency.

3.3 Fast_BFT Protocol

Most consensus protocols are inefficient and require high communication costs because nodes need to prevent malicious behavior in distributed protocols, therefore need to verify each other through voting. However, if proving that the primary is an honest node, message validation and reply only need to be completed through the primary. The other replicas no longer need to verify messages and vote for each other. They only need to be responsible for receiving messages sent by the primary. Fast_BFT can be divided into the preparation phase *Fast_Prepare* and the submission phase *Fast_Commit*, as shown in Fig. 4.

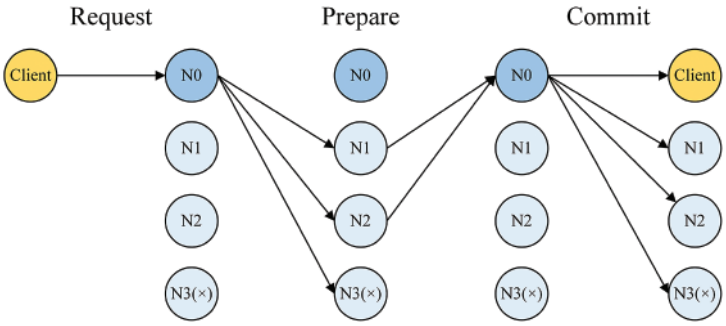

Figure 4: Fast_BFT operation

The specific process is as follows: a client sends a $< \ REQUEST, o, t, c \ >_{\sigma c}$ message to the primary p. In the *Fast_Prepare* phase, the primary verifies the message and sends the signed message to all replicas in the format of $<< \ FAST\_PREPARE, v, n, d, p, FastTime \ >_{\sigma p}, m \ >$ where *FastTime* is the number of messages processed by the primary in the Fast_BFT protocol. Upon receiving the *FAST_PREPARE* message, the replicas verify the signature and the consensus message count, store the message in a temporary message pool, and send the primary with a message $< \ FAST\_PREPARE, v, n, d, i, FastTime \ >_{\sigma_i}$. After the primary confirms that it has received *FAST_PREPARE* messages from $2f + 1$ nodes, it enters the *Fast_Commit* phase and replies message. The primary also sends a $< FAST\_COMMIT, v, n, D(m), FastTime >_{\sigma_i}$ to all replicas. Upon receiving the *FAST_COMMIT* from the primary, the replicas store the current message in their local logs.

Algorithm 1: Fast_BFT protocol

```
function handleClientRequest
    m = messageIDAdd()
    d = Hex(Request)                                  ▷ Get message digest
    if SWITCHFAST = true then                         ▷ FAST_BFT handle message
        go primaryFast_Prepare
function primaryFast_Prepare
    Primary.messagePool = Request
    Fast_Prepare = Sign(n, d, p, Request, FastTime)   ▷ Primary sign the message
    boradcast Fast_Prepare
function fast_Perepare
    if Sign = true then
        Node.messagePool = Request
        Fast_Prepare = Sign(m, d, i, FastTime)
        tcpDial Primary Fast_Prepare                  ▷ Send the Fast_Prepare to the Primary
function primaryFast_Commit
    if VoteCount ≥ 2f + 1 then
        FastTime++                                    ▷ Increased number of fast_BFT consensus
        Primary.localMessagePool = Message
        tcpDial client Reply                          ▷ Send reply to client
        Fast_Commit = Sign(m, d, i, FastTime)
        broadcast Fast_Commit
    if FastTime ≥ Random then
        go fast_Verify
function fast_Commit
    if Sign = true then
        FastTime++
        localMessagePool = Message
    if FastTime ≥ Random then
        go fast_Verify
```

3.4 Protocol Switching

Most dual-mode or multi-mode consensus protocols assume an optimistic environment, executing fast consensus protocols. Once a fork or other disruption to consistency and performance occurs, the protocol falls back to a backup protocol for error repair. In contrast, the CF-BFT protocol assumes

a pessimistic environment by default, using the Check_BFT protocol to detect whether the current environment is safe and whether the primary is an honest node. After confirming the security of the current environment, Fast_BFT is switched for processing messages. Therefore, in CF-BFT, protocol switching is simpler and faster than other multi-mode protocols.

CF-BFT protocol applies a mode-switching scheme. In which the modes are switched by triggering *SWITCHFAST* and *SWITCHCHECK* messages. The following two conditions need to be met to trigger the *SWITCHFAST* message:

- The primary successfully reply a message in the Check_BFT for this view;
- More than $2f$ replicas do not detect malicious behavior from the primary.

After replying to a message in Check_BFT, the primary initiates a protocol switching request, enters the node *Verify* phase, and broadcasts a *VERIFY* message $< VERIFY, v, p, r, d >_{\sigma p}$, where r is the number of times the primary processes messages in the fast protocol, consisting of the current reputation value of p and a random number. If the replicas don't detect any behavior from the primary that violates the protocol consistency, such as sending inconsistent messages or intentionally delaying communication, they consider the primary to be honest and broadcast a $< SWITCHFAST, v, p, r, i >_{\sigma_i}$ message. When a node receives $2f + 1$ *SWITCHFAST* messages, it considers the current primary to be a trusted node, switches the consensus protocol, and runs the Fast_BFT protocol. After the fast consensus ends, nodes broadcast a $< SWITCHCHECK, v, p, i, r, h >_{\sigma_i}$ message to switch to the Check_BFT protocol, where h is the hash value of all *FAST_COMMIT* messages during Fast_BFT, i.e., the hash value of the last r messages in the local log, used to re-verify whether the message contents of each node are consistent.

Algorithm 2: Operation mode switching

```
1: function verify
2:     if Primary.isReply[MessageID] = true then          ▷ Primary reply one message
3:         if Primary.isHonest = true then
4:             Random = random(p.reputation, t)
5:                                                        ▷ Calculate random by Primary reputation
6:             Verify = Sign(p, r, d)
7:             boradcast Verify
8:     if VoteCount ≥ 2f + 1 then
9:         SWITCHFAST = true                              ▷ Change mode to FAST_BFT
10:        boradcast SWITCHFAST
11:    else
12:        PrimaryID = H(strmessage, Timestamp)
13:                                                       ▷ Change Primary with hash operation
```

3.5 Fast_BFT Protocol's Verification

To further improve the safety of consensus and prevent adversary nodes from attacking the primary when it processes transactions alone, a fast protocol re-verify mechanism is introduced. When the number of *FastTime* reaches the upper limit r of Fast_BFT, all nodes consider the current term of primary to be over and enter the *Fast_Verify* phase. The normal nodes calculate the hash value h of the last r replied messages and broadcast a $< SWITCHCHECK, v, p, i, r, h >_{\sigma_i}$ message. The node checks whether the hash value h in the message is consistent with the locally calculated value. If $2f + 1$ messages with consistent results are received, the messages are considered correct, and the reputation of the current primary will increase. The more the primary that completes the *Fast_Verify*

tasks multiple times will gradually increase its reputation, the more its reputation values have been increased. Thereby its probability of processing events alone in Fast_BFT increases. If the number of h is different, it will fall back to the Check-BFT protocol to re-verify the correctness and consistency of the message to prevent forking. At the same time, the reputation of the primary is decreased, and the probability of processing events alone in fast consensus is decreased. All information including transactions and node information from the local message pool is stored in the current block and a blockchain database is constructed. The encrypted hash value of the total transactions in the current block and primary nodes information are stored in the block header. The hash value can be used to verify the data integrity of the database.

Algorithm 3: Fast_verify

```
function fast_Verify
    Hash = H(m_1, m_2, ..., m_r)          ▷ Calculate the hash of r messages
    Fast_Verify = Sign(i, r, Hash)
    if VoteCount ≥ 2f + 1 then
        Primary.reputation ++             ▷ The reputation value of the Primary increases
        SWITCHCHECK = true
        boradcast SWITCHCHECK             ▷ change mode to CHECK_BFT
    else
        Primary.reputation − −
        go re − verify(Random)            ▷ Re-verify r messages
```

4 Theoretical Analysis of CF-BFT

In this section, the safety, consistency, and communication complexity of the CF-BFT protocol is analyzed.

4.1 Safety

The Check_BFT and Fast_BFT subprotocols assume that there are $3f + 1$ nodes in the worst condition, where f is the number of Byzantine nodes. Safety is ensured through mutually voting and keys between nodes.

Lemma 1. In the Check_BFT protocol, if the primary is a Byzantine node, it can't pass authentication, and a view change is initiated to elect a new primary.

Proof. Assuming the primary is Byzantine, it signs and broadcasts verifiable messages, and the replicas receive and verify the results. Since there are at most f Byzantine nodes, replicas can receive at most f votes, which is not enough to enter the message submission phase. The message validation fails and a view change is initiated to elect a new primary.

Lemma 2. In the Fast_BFT protocol, if the primary does not work properly, replicas elect a new primary.

Proof. If the primary is attacked and becomes unresponsive, either by not responding to clients or not forwarding messages, the client sends a request message to all nodes after the timeout period. And the replicas initiate message validation and elect a new primary after detecting that the primary is unresponsive. This does not affect the final consensus result.

4.2 Consistency

Lemma 3. If the primary is a Byzantine node in the Check_BFT protocol and proposes different messages, conflicting messages cannot be submitted.

Proof. Assuming the primary is a Byzantine node, it sends different messages to $3f$ replicas. It sends message m to $A = f + 1$ honest nodes, sends message m' to $A' = f$ honest nodes, and coordinates f Byzantine nodes to vote for message m to the A node cluster and votes for message m' to the A' node cluster. The A node cluster receives $2f + 1$ votes for message m, and message m is successfully submitted. However, the A' node cluster receives at most $2f$ votes for message m', and message m' cannot be submitted. The A' node cluster cannot submit the message, and it will warn other replicas that the primary is faulty. The A node cluster will receive the warning message and replicate message m to the A' node cluster to ensure consistency.

Lemma 4. In the Fast_BFT protocol, if the primary tries to destroy consistency, the Fast_BFT validation phase ensures consistency.

Proof. If the primary is attacked and sends different messages under the Fast_BFT protocol, it will temporarily cause different messages among the nodes. However, during the Fast_BFT validation phase, inconsistent messages can be detected, and the inconsistent messages will be rolled back for re-verification. And correct messages will be verified for completeness and consistency while conflicting messages will be deleted to ensure consistency.

4.3 Communication Complexity

Here the communication costs of the protocols are calculated and compared. In the Check_BFT protocol, this paper set the average communication cost to be Z_{check} and the average communication cost to be Z_{fast} in the Fast_BFT.

The average communication cost of the protocol is defined as Z, where Z_{min} represents the communication cost in the best condition and Z_{max} represents the communication cost in the worst condition. A random variable μ represents the number of times the primary processes messages in the Fast_BFT protocol. It is easy to prove that in the best condition:

$$Z_{min} = \frac{Z_{check} + \mu Z_{fast}}{1 + \mu} \tag{1}$$

In the worst condition:

$$Z_{max} = \frac{(4 + \mu)\, Z_{check} + 3\mu Z_{fast}}{3 + 3\mu} \tag{2}$$

Compared to traditional BFT protocols which can tolerate up to 1/3 Byzantine nodes, in CF-BFT protocol, the worst case is when all 1/3 of the nodes are attacked and turn from honest to malicious during their terms. Therefore, the conditions for CF-BFT protocol to reach the worst case are more stringent, and the requirements for the optimistic condition are lower, as long as the malicious nodes do not attack during their extremely short terms.

5 Performance Evaluation

In this section, by comparing with traditional PBFT protocol, BFT-SmaRt, and other improved BFT protocols, as well as SAZyzz dual-mode protocol, the throughput and latency will be analyzed. One of the main objectives of test evaluation is to detect whether CF-BFT protocol still has an

advantage over some traditional protocols in the worst condition of tolerating Byzantine nodes. The tests were conducted using a 2.10 GHz Intel(R) Xeon(R) 5218R CPU and 8 GB RAM.

5.1 Performance Comparison with BFT Protocols

Consensus latency is one of the most critical metrics for evaluating consensus protocols. In the experiment, this paper varied the block size and ran tests with 4, 8, 16, 32, and 64 nodes to measure the throughput and latency, using 512, 1024, 2048, 4096, and 8192 transactions per block. The results are shown in Fig. 5. As expected, the impact of increasing the number of nodes on protocol latency is not high when the block size is moderate.

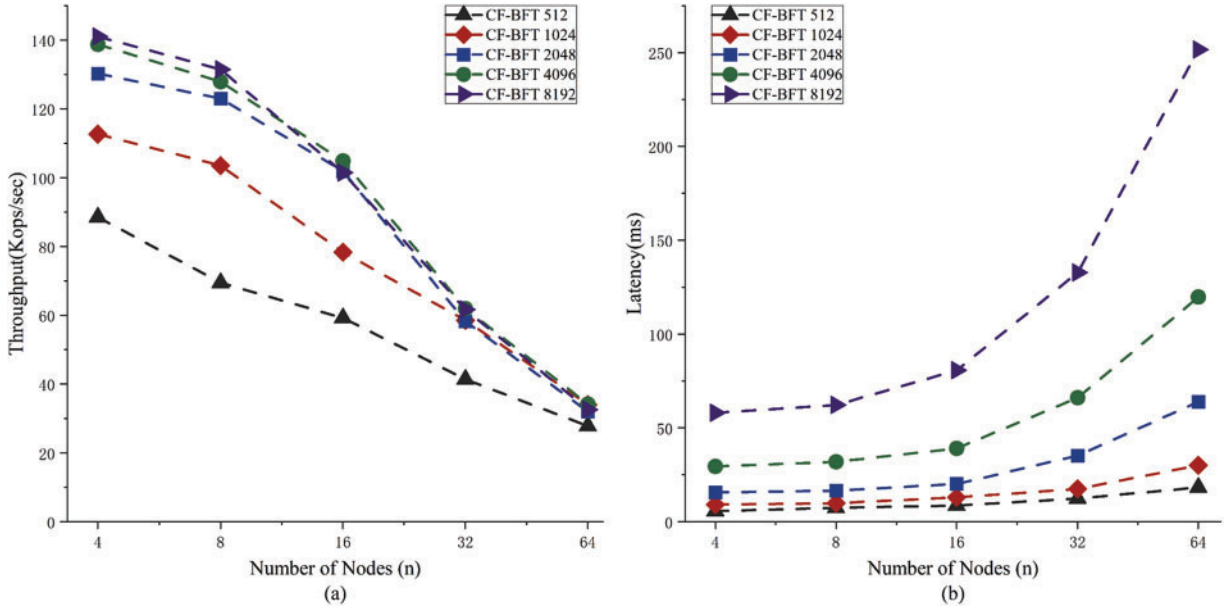

Figure 5: Performance analysis of CF-BFT protocol

Fig. 6 shows the impact of launching different numbers of nodes on the throughput and latency of CF-BFT, PBFT, BFT-SMaRt, and Hotstuff under the Bitcoin benchmark (256 bytes per transaction, 1024 transactions per block). The experiment results show that the advantage of the CF-BFT protocol increases as the number of nodes increases. Due to the use of full broadcast, PBFT and other protocols are heavily influenced by the number of nodes, while CF-BFT protocol is less affected. Moreover, even in the worst-case scenario, CF-BFT still has advantages.

The theoretical communication times of CF-BFT and PBFT are calculated as shown in Fig. 7. Here, μ represents the number of transactions processed by the primary in the Fast_BFT. The impact of $\mu = 10$ and $\mu = 20$ on the number of communication are compared. The data shows that the larger number of nodes, the greater advantage of the CF-BFT protocol.

5.2 Performance Comparison with Dual Mode Protocol

This section compared CF-BFT in optimistic and worst conditions with the SAZyzz dual-mode protocol by setting a block size to 100–400 KB under 4, 7, 10, 25, and 46 deployed nodes. Results are shown in Fig. 8, CF-BFT protocol consistently outperformed SAZyzz, particularly with more nodes.

CF-BFT has stricter worst condition requirements, where each malicious node must be flipped during its term, leading to a high usage rate of the fast protocol in practical applications.

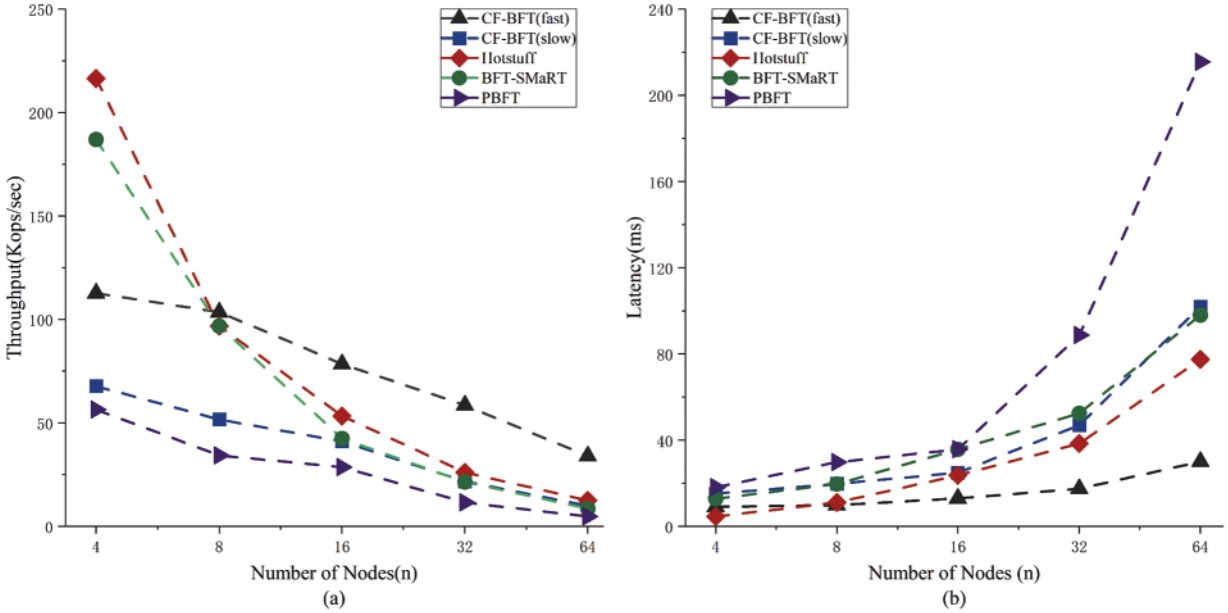

Figure 6: BFT protocols performance comparison

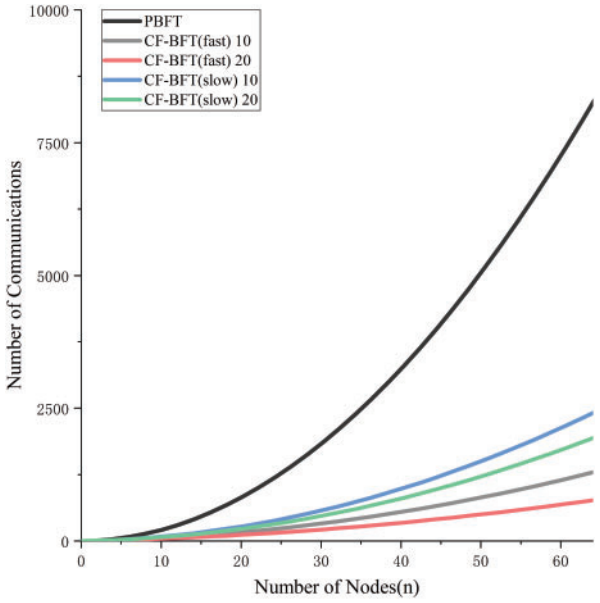

Figure 7: Number of communications

CF-BFT switches modes by triggering *SWITCH* messages, and the protocol switching time can be ignored. Compared to traditional dual-mode protocols that require a lot of resources to switch protocols, CF-BFT has a significant advantage in protocol switching speed.

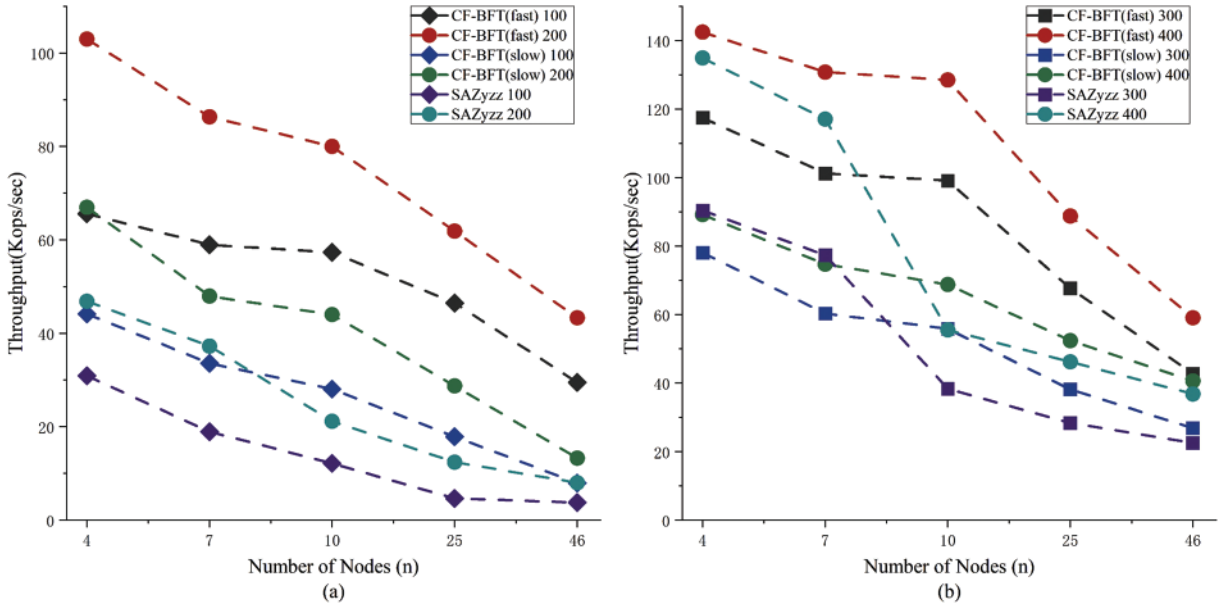

Figure 8: Dual-mode protocols performance comparison

6 Conclusion

This paper presents a node-authenticated CF-BFT dual-mode consensus protocol, consisting of Check_BFT and Fast_BFT subprotocols. By authenticating nodes before entering the fast protocol and implementing a simpler protocol switch, it outperforms previous BFT consensus protocols and other dual-mode protocols in terms of throughput and consensus efficiency. Unlike traditional dual-mode protocols, the CF-BFT protocol defaults to a pessimistic condition and switches to the optimistic fast protocol only after node identity is verified. Therefore, even in the worst condition, CF-BFT also can significantly improve throughput and consensus efficiency. The advantages of the protocol become more pronounced as the number of nodes increases. In addition, to enhance the safety of the protocol and prevent attacks from Byzantine nodes, the CF-BFT provides the replica alert mechanism that can package evidence of the primary's malfeasance and provide it to other nodes to ensure the correctness of main node identity authentication. Furthermore, the protocol introduces a node reputation mechanism, where nodes with higher reputation values are more likely to process transactions alone in the Fast_BFT. Test results show that CF-BFT outperforms traditional BFT protocols and other dual-mode protocols in all performance indicators. Under the premise of ensuring consistency and safety, it dramatically reduces communication costs and improves performance, providing more efficient and secure consensus services for blockchain systems.

Based on the feature of CF-BFT protocol that does not require client nodes to participate in consensus and its high performance, it has potential application prospects in public chains, enterprise-level chains, Internet of Things (IoT), financial services, and public services. It can be used in distributed systems in IoT, such as sensor data collection and processing, and security control; in distributed ledgers in financial services, such as insurance contract execution and securities trading settlement; as well as in distributed systems in public services, such as election voting and supply chain management.

Currently, CF-BFT has been tested with a maximum of 64 nodes. However, in the case of a large-scale Internet of Things network composed of more nodes, further research is needed to improve the performance of the CF-BFT. The next step is to improve the protocol in three aspects. Firstly, by using a pipelined consensus approach, reducing the computational difficulty of each primary. Secondly, the blockchain structure can be optimized by introducing a DAG graph-style structure [32] to improve efficiency. Thirdly, we can optimize the structure of the nodes by changing the communication method [33] to reduce consumption between nodes.

Acknowledgement: The authors would like to thank the support of the Distributed Computing & BlockChain Lab, Henan University of Technology and Government, and the hard work of the editors and reviewers.

Funding Statement: The authors are supported by the Key Laboratory of Network Password Technology in Henan Province, China (LNCT2022-A20); the Major Science and Technology Special Project of Henan Province, China (Nos. 201300210100, 201300210200); the Key Scientific Research Project of Higher Education Institutions in Henan Province, China (No. 23ZX017); the Key Special Project for Science and Technology Collaborative Innovation in Zhengzhou City, Henan Province, China (No. 21ZZXTCX07); and the Key Science and Technology Project of Henan Province, China (No. 232102211082).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Zhiruo Zhang, Xinlei Liu, Feng Wang; data collection: Zhiruo Zhang, Yang Lu, Feng Wang; code development: Zhiruo Zhang, Xinlei Liu; analysis and interpretation of results: Zhiruo Zhang, Feng Wang; draft manuscript preparation: Zhiruo Zhang, Feng Wang; supervision: Feng Wang, Yang Liu; project administration: Feng Wang, Yang Liu. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The datasets used and analysed during the current study are available from the first author on reasonable request.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Y. Yuan and X. C. Ni, "Blockchain consensus algorithms: The state of the art and future trends," *Acta Automatica Sinica*, vol. 42, no. 4, pp. 481–494, 2016.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, vol. 31, pp. 21260, 2008.

[3] H. Wang, C. Xu, C. Zhang, J. Xu, Z. Peng *et al.,* "vChain+: Optimizing verifiable blockchain boolean range queries," in *2022 IEEE 38th Int. Conf. on Data Engineering*, Kuala Lumpur, Malaysia, pp. 1927–1940, 2022.

[4] Z. Peng, J. Xu, H. Hu, L. Chen and H. Kong, "BlockShare: A blockchain empowered system for privacy-preserving verifiable data sharing," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 1, pp. 14–24, 2022.

[5] Z. Li, S. Gao, Z. Peng, S. Guo, Y. Yang *et al.,* "B-DNS: A secure and efficient DNS based on the blockchain technology," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1674–1686, 2021.

[6] X. Li and Y. Liu, "Research and implementation of intellectual property trading platform based on blockchain," *Computer Engineering and Applications*, vol. 59, no. 3, pp. 308–316, 2023.

[7] J. Lu, J. Shen, P. Vijayakumar and B. B. Gupta, "Blockchain-based secure data storage protocol for sensors in the industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 8, pp. 5422–5431, 2021.
[8] G. N. Nguyen, N. H. Le Viet, M. Elhoseny, K. Shankar, B. Gupta *et al.,* "Secure blockchain enabled cyber-physical systems in healthcare using deep belief network with ResNet model," *Journal of Parallel and Distributed Computing*, vol. 153, no. 2, pp. 150–160, 2021.
[9] B. B. Gupta, K. C. Li, V. C. Leung, K. E. Psannis, S. Yamaguchi *et al.,* "Blockchain-assisted secure fine-grained searchable encryption for a cloud-based healthcare cyber-physical system," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 12, pp. 1877–1890, 2021.
[10] I. Cvitić, D. Peraković, M. Periša and B. Gupta, "Ensemble machine learning approach for classification of IoT devices in smart home," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 11, pp. 3179–3202, 2021.
[11] Y. Liu, W. Yang, Y. Wang and Y. Liu, "An access control model for data security sharing cross-domain in consortium blockchain," *IET Blockchain*, vol. 3, no. 1, pp. 18–34, 2023.
[12] Z. Peng, J. Xu, X. Chu, S. Gao, Y. Yao *et al.,* "Vfchain: Enabling verifiable and auditable federated learning via blockchain systems," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 173–186, 2021.
[13] H. Wu, Z. Peng, S. Guo, Y. Yang and B. Xiao, "VQL: Efficient and verifiable cloud query services for blockchain systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1393–1406, 2021.
[14] L. Lamport, R. Shostak and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.
[15] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *Self-Published Paper*, vol. 19, no. 1, pp. 1–6, 2012.
[16] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *3rd USENIX Symp. on Operating Systems Design and Implementation (OsDI1999)*, New Orleans, Louisiana, NOLA, USA, vol. 99, pp. 173–186, 1999.
[17] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proc. of the 2019 ACM Symp. on Principles of Distributed Computing*, Toronto, Canada, pp. 347–356, 2019.
[18] A. Bessani, J. Sousa and E. E. Alchieri, "State machine replication for the masses with BFT-SMART," in *2014 44th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, Atlanta, GA, USA, IEEE, pp. 355–362, 2014.
[19] Y. Gilad, R. Hemo, S. Micali, G. Vlachos and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. of the 26th Symp. on Operating Systems Principles*, Shanghai, China, pp. 51–68, 2017.
[20] H. Moniz, "The istanbul bft consensus algorithm," arXiv preprint arXiv: 2002.03613, 2020.
[21] G. Zhang and H. A. Jacobsen, "Prosecutor: An efficient BFT consensus algorithm with behavior-aware penalization against byzantine attacks," in *Proc. of the 22nd Int. Middleware Conf.*, New York, NY, USA, pp. 52–63, 2021.
[22] R. Pass and E. Shi, "Thunderella: Blockchains with optimistic instant confirmation," in *Advances in Cryptology-EUROCRYPT 2018: 37th Annual Int. Conf. on the Theory and Applications of Cryptographic Techniques*, Tel Aviv, Israel, Springer, pp. 3–33, 2018.
[23] G. Danezis, L. Kokoris-Kogias, A. Sonnino and A. Spiegelman, "Narwhal and tusk: A dag-based mempool and efficient bft consensus," in *Proc. of the Seventeenth European Conf. on Computer Systems*, Rennes, France, pp. 34–50, 2022.
[24] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle *et al.,* "Cheapbft: Resource-efficient byzantine fault tolerance," in *Proc. of the 7th ACM European Conf. on Computer Systems*, New York, NY, USA, pp. 295–308, 2012.

[25] R. Kotla, L. Alvisi, M. Dahlin, A. Clement and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," in *Proc. of Twenty-First ACM SIGOPS Symp. on Operating Systems Principles*, New York, NY, USA, pp. 45–58, 2007.

[26] R. Guerraoui, N. Knežević, V. Quéma and M. Vukolić, "The next 700 bft protocols," in *Proc. of the 5th European Conf. on Computer Systems*, Paris, France, pp. 363–376, 2010.

[27] N. Sohrabi, Z. Tari, G. Voron, V. Gramoli and Q. Fu, "Sazyzz: Scaling azyzzyva to meet blockchain requirements," *IEEE Transactions on Services Computing*, vol. 6, pp. 1–14, 2022.

[28] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas *et al.,* "Sbft: A scalable and decentralized trust infrastructure," in *2019 49th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, Portland, OR, USA, pp. 568–580, 2019.

[29] Y. Lu, Z. Lu and Q. Tang, "Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined bft," in *Proc. of the 2022 ACM SIGSAC Conf. on Computer and Communications Security*, Los Angeles, CA, USA, pp. 2159–2173, 2022.

[30] 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 Int. Conf., FC 2022*, Grenada, Spain, pp. 296–315, 2022.

[31] S. Ren, C. Lee, E. Kim and S. Helal, "Flexico: An efficient dual-mode consensus protocol for blockchain networks," *PLoS One*, vol. 17, no. 11, pp. e0277092, 2022.

[32] L. Baird, "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance," *Swirlds Tech Reports SWIRLDS-TR-2016-01*, vol. 34, pp. 9–11, 2016.

[33] S. Gao, Z. Peng, F. Tan, Y. Zheng and B. Xiao, "SymmeProof: Compact zero-knowledge argument for blockchain confidential transactions," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 2289–2310, 2022.