SCIENCE CHINA
Information Sciences

• REVIEW •

January 2026, Vol. 69, Iss. 1, 111102:1–111102:27
https://doi.org/10.1007/s11432-024-4431-y

From PBFT to the present: a thorough overview of blockchain consensus protocols

Liaoliao FENG[1,2], Xiang FU[1,2*], Huaimin WANG[1,2*], Keming WANG[1], Peichang SHI[1,2], Feng JIANG[1,2] & Moheng LIN[1,2]

[1]*College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China*
[2]*State Key Laboratory of Complex & Critical Software Environment, Changsha 410073, China*

Received 22 November 2024/Revised 11 February 2025/Accepted 30 April 2025/Published online 24 October 2025

Abstract The emergence of Bitcoin has introduced blockchain technology, which has been widely applied in various scenarios, prompting the continuous pursuit of more efficient, energy-saving, and secure blockchain consensus protocols. Byzantine fault tolerance (BFT) consensus serves as an effective solution in specific contexts. From the Byzantine generals problem in 1982 to the first practical BFT protocol, PBFT in 1999, an increasing number of BFT protocols have been proposed and applied to blockchain systems. The surge in protocols has made it difficult to navigate the BFT landscape, let alone discern which specific protocols optimally align with diverse application requirements. To facilitate a better understanding, design, and application of blockchain consensus protocols, this paper adopts PBFT as a baseline to critically assess the design of BFT variants in this paper. We categorize the design philosophies of existing blockchain BFT consensus into two primary types: the redesign of fundamental consensus processes (RFCP) and the addition of supplementary optimization mechanisms (ASOM). For RFCP, taking the consensus processes of PBFT as a reference, we analyze the design considerations such as the number of commitment phases, the number of replicas, and view-change strategies. These structural elements interact with each other, affecting the protocol's fundamentals and correctness. For ASOM, we outline several optimizations, such as committee-based consensus, optimistic mechanisms, and pipelining, which can be layered without affecting the fundamental consensus processes. Finally, according to the proposed design philosophies, we present a detailed framework to rapidly analyze or design a specified blockchain BFT consensus protocol.

Keywords blockchain, Byzantine fault-tolerant, consensus protocol, rapid analysis framework

Citation Feng L L, Fu X, Wang H M, et al. From PBFT to the present: a thorough overview of blockchain consensus protocols. Sci China Inf Sci, 2026, 69(1): 111102, https://doi.org/10.1007/s11432-024-4431-y

1 Introduction

The remarkable success of Bitcoin [1] has drawn significant attention to its underlying technology, blockchain, which has seen increasing adoption across a wide range of domains [2–9]. Fundamentally, blockchain serves as a distributed ledger that integrates cryptographic protocols, peer-to-peer networks, and other decentralized technologies. Consequently, the consensus challenges traditionally encountered in distributed systems also persist in blockchain environments. Consensus is a fundamental process in distributed systems that enables nodes to agree on a shared state, ensuring secure and reliable service delivery. In state machine replication (SMR), each node starts from the same initial state, executes instructions in the same order, and ultimately reaches the same final state. SMR requires consensus on the execution order of commands, whereas in blockchain systems, consensus instead pertains to the sequence of blocks.

Compared with crash faulty nodes, Byzantine faulty nodes can behave arbitrarily, such as executing incorrect code, sending erroneous or inconsistent messages. Byzantine fault tolerant (BFT) protocols are designed to ensure that, as long as the proportion of faulty nodes remains below a certain threshold, all correct nodes can reach agreement on a target value while preserving system liveness. Byzantine fault tolerance represents a stricter requirement than crash fault tolerance and is essential for use cases like permissioned blockchains, firewalls, and SCADA systems. It is particularly suited for blockchain environments, which are complex and open, increasing the risk of Byzantine faults. Although blockchain consensus includes many proof-of-X protocols, such as proof of work (PoW) and proof of stake (PoS), this paper focuses exclusively on BFT-based consensus algorithms. The strong fault tolerance of BFT protocols comes at the cost of high complexity, making correct implementation inherently more difficult than

* Corresponding author (email: fuxiang13@nudt.edu.cn, whm_w@163.com)

Table 1 Comparison of related work.

Related work	Features	Difference
Berger and Reiser [14] and Fu et al. [13]	Investigated blockchain consensus protocols from various dimensions.	Focus on all blockchain consensus, not just BFT.
Correia et al. [15]	Focused on the necessity of reaching consensus in the presence of potential Byzantine replicas. Their survey primarily analyzes different consensus protocols at a theoretical level.	Focus on asynchronous Byzantine consensus.
Platania et al. [16]	Classified BFT protocols based on their system model (server side versus client side).	Focus on distinguishing BFT protocols based on the roles assigned to clients.
Distler [17]	Discussed some important components of a BFT system (i.e., clients, agreement protocol, execution stage), then focused on selected approaches and mechanisms addressing specific tasks such as checkpoint and recovery.	Focus on the BFT system architecture (i.e., clients, agreement protocol, execution stage), with an emphasis on process over theoretical correctness.
Zhang et al. [18]	Categorized consensus protocols based on efficiency, availability, and robustness.	Focus on consensus protocol characteristics rather than design principles.
Gupta et al. [19]	Reviewed consensus protocols optimized with trusted hardware.	Focus on the improvement that trusted hardware brings to consensus protocols.
Amiri et al. [20]	Proposed a platform for generating consensus protocols, breaking down design elements to help scholars customize consensus protocols at a granular level.	Many design choices are given but their impact on protocol correctness is not revealed.
Sui et al. [21]	Constructed mathematical models for BFT consensus protocols, capturing various consensus methods.	Model only linearizes BFT and proves the correctness of the variant, but lacks an intuitive explanation.

that of crash fault tolerant (CFT) protocols [10–12]. BFT protocols cover a wide range of design aspects, from infrastructure (e.g., data structures, communication topologies) to technical features (e.g., commitment strategies, view-change strategies) and social choice properties (e.g., order fairness). In recent years, numerous protocols have been proposed to improve performance, availability, or resource efficiency, making it difficult to navigate the BFT field, let alone determine which protocol best suits specific application needs.

Our preliminary findings indicate that consensus protocols inherently involve trade-offs among decentralization, efficiency, and security, implying no single consensus algorithm is optimal for all scenarios (e.g., the financial sector often prioritizes efficiency and security over decentralization) [13]. Therefore, designing and selecting consensus protocols tailored to specific applications remains a pressing challenge.

Many studies have conducted in-depth analyses of blockchain consensus protocols, as shown in Table 1 [13–21]. However, none has systematically examined the design philosophies of these protocols and their variants through the lens of their fundamental processes. Additionally, the interplay among different structural elements and the stackability of optimization mechanisms remain underexplored. Furthermore, many blockchain BFT consensus protocols are variants of PBFT. Therefore, we anchor our analysis in PBFT to examine the design processes of various blockchain consensus protocols, analyzing why different designs are correct or efficient. This effort aims to help researchers and users better understand consensus protocols. We strive to answer the following questions.

RQ1. What are the key differences between their consensus processes and those employed by PBFT?

RQ2. How do these differences ensure safety and liveness?

RQ3. What are the considerations for introducing these differences?

RQ4. What are the advantages and disadvantages of these differences?

RQ5. Which designs/optimizations influence each other and cannot be layered, and which ones can be layered?

RQ6. How to determine the optimal form of the consensus protocol under existing optimization methods?

RQ7. How to introduce suitable optimizations based on the scenario's features to derive a consensus protocol that meets the requirements?

The main contributions of this paper are shown in Figure 1.

• **We analyzed the redesign of fundamental consensus processes (RFCP) from the perspective of PBFT.** Deconstructing it into three core components: the number of commitment phases, the number of replicas, and the view-change protocol. Our analysis systematically explored the interactions among these elements by identifying factors influencing changes in phase count, deriving the mathematical constraints governing replica numbers, and dissecting the message structure of the view-change protocol. We emphasized that achieving consensus correctness requires coordinated modifications across these components, aiming to provide a rigorous characterization of their interdependencies.

• **We summarized the addition of supplementary optimization mechanisms (ASOM) that operate independently of the fundamental consensus processes.** In particular, we identified five mechanisms that

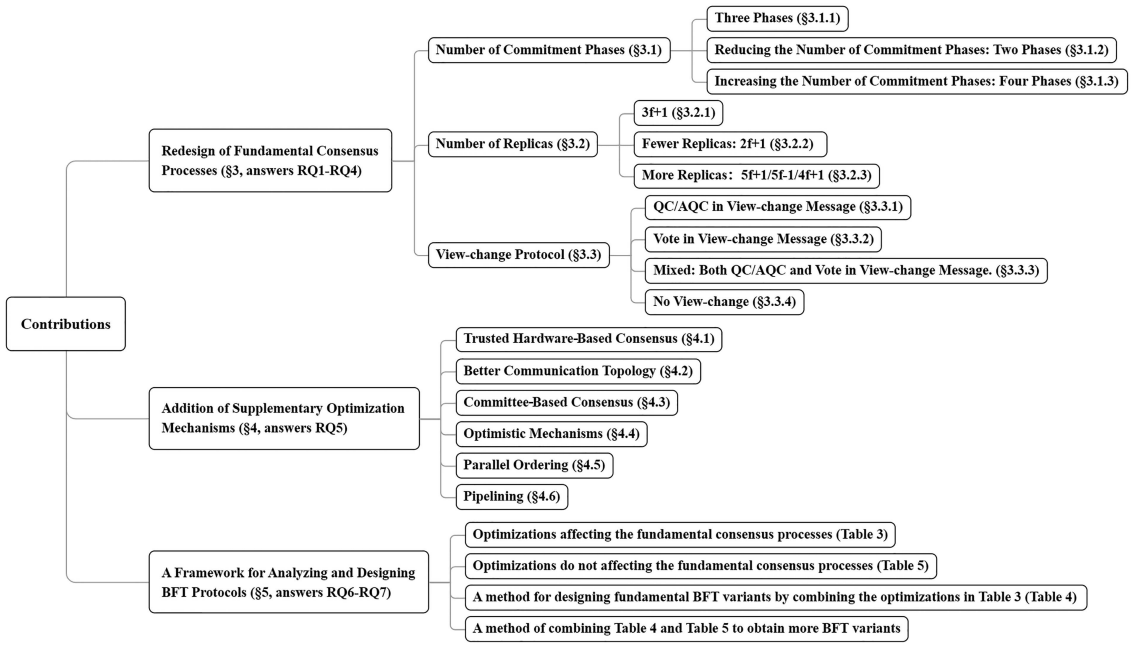

Figure 1 The main contributions of this paper.

can be layered on top of the core consensus processes to enhance protocol performance without altering their fundamental structure.

• **We proposed a framework for rapidly analyzing or designing blockchain consensus protocols**. This framework decomposes design choices affecting fundamental consensus processes into atomic components, detailing their individual impacts. By systematically applying permutations and filters, it generates all feasible protocol variants along with their associated performance metrics. Incorporating ASOM further expands the design space, enabling the creation of additional protocol variations. This approach allows efficient analysis of existing protocols and facilitates the design of tailored protocols to meet specific application requirements and leverage available optimizations.

This paper is organized as follows. Section 2 introduces background on commonly used assumptions in blockchain BFT consensus. Section 3 reviews existing consensus protocols and design principles from the perspective of fundamental processes. Section 4 categorizes stackable optimizations and surveys corresponding protocols. Section 5 introduces a framework for analysing and designing consensus protocols, which captures existing designs and facilitates the construction of new variants. Section 6 concludes with a summary and future research directions.

2 Background

This section introduces the basic concepts that underpin the blockchain BFT consensus. The BFT consensus protocol plays a vital role in blockchain by ensuring agreement among nodes in the presence of Byzantine faulty. In this overview, we delve into several key elements of the blockchain BFT consensus protocol, clarifying system models and underlying assumptions.

2.1 State machine replication and Byzantine fault tolerant protocols

BFT SMR is an important building block for constructing permissioned blockchain systems [22].

SMR is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas. A state machine, central to SMR, consists of state variables and commands. State variables depict the system's current state, while commands govern state transitions [23]. SMR leverages distributed consensus protocols to keep multiple replicas consistent despite replica failures or network partitions [24].

Byzantine fault-tolerant SMR can tolerate Byzantine failures. Byzantine failures are the most severe type of failure in SMR [25]. Unlike benign/crash failures where messages are unaltered and state changes are detectable [26], Byzantine faults allow servers to behave arbitrarily, including message tampering and state mutations. Faulty servers may also collaborate maliciously [27].

2.2 Network assumption

Consensus protocols can operate under different network assumptions, typically classified as synchronous, asynchronous, and partially synchronous networks [28]. In synchronous networks, there is a fixed upper bound on message delivery time (denoted as Δ) and a fixed upper bound on the discrepancy between processes or clocks (denoted as δ). In contrast, asynchronous networks have no fixed upper bound on message delivery time (i.e., Δ does not exist) or on processes or clock discrepancies (i.e., δ does not exist) [29]. A network is considered partially synchronous if both Δ and δ exist but are unknown, or if Δ and δ become known after a globally stable time (GST) [28, 30]. This paper focuses primarily on partially synchronous protocols. Some of these protocols, like PBFT [31], continuously increase the timer value after a timeout until it exceeds the unknown delay Δ. Other protocols, such as Tendermint [32], ensure safety in asynchronous settings and resume liveness only after the GST.

2.3 Correctness

The correctness guarantee can be explained in terms of safety and liveness properties [33].

(1) Safety ensures that no two non-faulty replicas will decide different orders for request execution, even in the presence of failures.

(2) Liveness guarantees that the program eventually enters a desirable state. In the context of consensus, it means clients will eventually receive replies to their requests [31]. In the context of blockchain consensus, it means that the blockchain will eventually output new messages to all peers [34].

In brief, the safety property ensures that something bad will never happen, while the liveness property ensures that something good eventually happens, despite failures.

2.4 Complexity

Complexity is a key metric for evaluating consensus protocols, commonly assessed through message complexity and time complexity.

Message complexity refers to the total number of messages sent. For instance, in a fully connected network of n replicas, if one replica broadcasts a message over k rounds, the message complexity is $O(kn)$. If every replica broadcasts a message over k rounds, it increases to $O(kn^2)$.

Time complexity measures the number of message-passing rounds required to reach consensus. For example, if server S_i sends a message to server S_j and S_j replies to S_i, the time complexity is 2.

In some cases, certain messages in consensus protocols carry significantly more data than others. For instance, in PBFT, a view-change message may include $2f$ matching prepare messages for each prepared request and multiple checkpoint messages. Therefore, message complexity alone may not accurately capture the network load. Factoring in message size within the complexity metric provides a more comprehensive evaluation. Authenticator complexity can effectively address this issue. It measures the total number of authenticators (partial or full signatures) received by a replica. Unlike message complexity, authenticator complexity abstracts away unnecessary transmission topology details. For example, n messages each carrying one authenticator have the same authenticator complexity as one message carrying n authenticators, and both have similar transmission times. Hence, while message complexity is more of a logical metric, authenticator complexity more accurately reflects network load. Moreover, since cryptographic operations such as generating or verifying signatures are typically the most computationally intensive tasks, authenticator complexity also provides insight into the protocol's computational burden.

2.5 Cryptographic primitives reducing the authenticator complexity

Certain cryptographic primitives, such as threshold signatures [35–37], aggregated signatures [35, 38, 39], and secret sharing [40], can help reduce the authenticator complexity in BFT consensus.

In a (n, t) threshold scheme, an n-member group has a pair of public and private keys, allowing the signatures of any t members to be combined into a verifiable group signature. This mechanism is well-suited for composing aggregated quorum certificates (AQCs) in consensus protocols, thus reducing communication complexity. Blockchain consensus protocols use threshold signatures including SBFT [41], HotStuff [42], and RBFT [43].

In protocols utilizing aggregated signatures, the primary can condense signatures received from multiple replicas into a single, constant-sized aggregated signature. Given the aggregated signature, along with the set of plaintext messages used to generate it and the corresponding public keys of the message senders, the aggregated signature can be verified. The number of messages involved in the aggregation ranges from $2f + 1$ to n. To authenticate the identities of message senders, each replica maintains the public keys of all other replicas. Consensus protocols employing aggregated signatures include Fast-Hotstuff [44], Dfinity [45], and D-SBFT [46].

In protocols utilizing secret sharing. The administrator generates and splits keys in the TEE and then distributes the shards and hash values to the replicas, destroying the original keys. During consensus, the primary links the key to a block and collects key shards from replicas. Once enough shards are collected, the primary reconstructs the key, broadcasts it, and replicas verify its authenticity to confirm the vote. Consensus protocols employing secret sharing include FastBFT [47] and TBFT [48].

Combined with linearization, those cryptographic primitives can both reduce the complexity in consensus protocols significantly. However, they also exhibit several differences: aggregated signatures have low building costs, whereas constructing threshold signatures incurs quadratic costs. In contrast, threshold signatures have minimal verification costs, whereas the verification costs for aggregated signatures are linear. Additionally, the identities of message senders remain unknown in threshold signature schemes, whereas protocols employing aggregated signatures require knowledge of the senders' identities.

The application of cryptographic primitives in blockchain is not limited to partially asynchronous BFT consensus. Some asynchronous BFT consensus protocols [49–51] also employ threshold signature techniques to reduce message complexity. Beyond consensus, some cases [52, 53] also apply cryptographic algorithms to the data privacy domain in blockchain.

2.6 PBFT

The PBFT protocol, introduced by Castro and Liskov in 1999 [31], improves upon Lamport's 1982 Signed Message (SM) BFT protocol [27], reducing its complexity from exponential to quadratic.

The PBFT protocol comprises three sub-protocols: normal-case operation, garbage collection, and view-change. Ideally, the system functions under the normal-case operation protocol during stable network conditions. The garbage collection protocol eliminates obsolete log messages, while the view-change protocol is activated by the misbehavior of the primary node or network failures. A view represents a configuration in which one node acts as the primary, while the remaining nodes serve as backups. In successive views, nodes rotate their roles as primary, with view numbers incrementing sequentially.

The normal-case operation protocol consists of three phases: the pre-prepare phase, the prepare phase, and the commit phase. Upon receiving a request from the client, the primary node initiates the pre-prepare phase, assigning a unique identifier to each request and broadcasting a pre-prepare message that includes the mapping of identifiers to requests. The backup nodes cast votes during the pre-prepare and prepare phases. After receiving enough votes in two voting rounds, the nodes decide and execute the request, then send a response back to the client. The pre-prepare and prepare phases ensure consistency within the same view, while the prepare and commit phases guarantee consistency across views. Additionally, view-change ensures system liveness in the event that the primary node is faulty.

In PBFT, there are three critical thresholds. Assuming there are at most f faulty nodes in the system, at least $3f + 1$ nodes are needed to guarantee the protocol's correctness. For each voting phase, a node requires at least $2f + 1$ votes to make a decision, while the client needs at least $f + 1$ identical matching responses from different backup nodes to confirm the result. If a node receives $2f+1$ prepare votes that match the request in the pre-prepare message during the prepare phase, the request is considered "prepared", and the node broadcasts its commit vote. In the commit phase, if the node collects $2f + 1$ commit votes that match both its own prepare vote and the pre-prepare message it received, the corresponding request is considered "committed-local", and the node decides that request.

The correctness of the protocol encompasses both safety and liveness. In a system with $3f+1$ nodes, any subset of at least $2f+1$ nodes has an overlap containing at least one correct node. Consequently, during the prepare phase, no two conflicting requests can simultaneously receive $2f + 1$ prepare votes. Similarly, no two conflicting requests can receive matching $2f + 1$ commit votes, ensuring intra-view consistency. If a malicious primary prevents correct nodes from deciding on a request for an extended period, the view-change protocol will be triggered until most correct nodes decide the same request, thus guaranteeing liveness. If a request has already been partially decided by some correct nodes before the view change, it will be carried into the new view. Eventually, all correct nodes will decide this request, preventing any correct node from deciding conflicting requests, thereby ensuring cross-view consistency. This mechanism extends consistency from a single round to multiple rounds through the view-change protocol, ultimately ensuring the protocol's safety.

The view-change message carries stable checkpoints, along with $2f + 1$ votes as its proof, as well as a set of pre-prepare messages for requests that have been prepared at the sender, accompanied by their quorum certificates (QCs). For PBFT, a QC for a pre-prepare message consists of $2f$ matching prepare votes. The new primary's new-view message includes the received view-change messages and the pre-prepare messages calculated based on

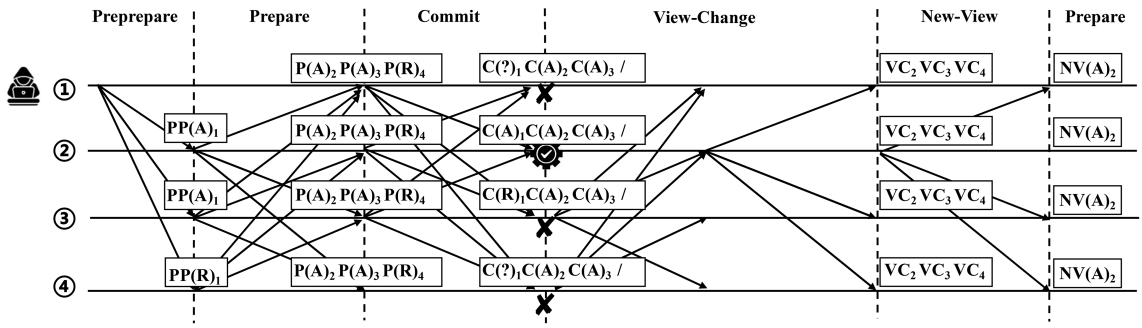

Figure 2 The switching between the normal-case operation protocol and the view-change protocol in PBFT [31]. We use combinations of initial letters to represent the abbreviations for vote types, with the request indicated in parentheses and the sending node denoted by a subscript. For example, $\mathrm{PP}(A)_1$ denotes a pre-prepare message for request A sent by node 1. This notation is employed to describe the set of messages received by each node in each phase. In this case, the Byzantine primary (node 1) sends request A to nodes 2 and 3, while sending request R to node 4. During the prepare phase, all nodes receive prepare votes for request A from nodes 2 and 3, and a prepare vote for request R from node 4. At this point, nodes 2 and 3 receive $2f$ prepare votes along with a pre-prepare vote for request A, and thus send commit votes for request A. Node 4, however, does not receive a pre-prepare vote for request A and receives fewer than $2f$ prepare votes for request R, so it does not send a commit vote. Meanwhile, the Byzantine primary may send commit votes for arbitrary requests (denoted as $C(?)_1$). During the commit phase, if node 2 receives the primary's commit vote for request A while node 3 receives the commit vote for request R, node 2 decides on request A. Nodes 3 and 4 will then send view-change messages (node 4 may have initiated a view-change protocol upon detecting Byzantine behavior of the primary during the prepare phase or do so after timeout). During the view-change phase, when node 2 receives view-change messages from nodes 3 and 4, it collects $f+1$ messages and sends its own. If node 2 becomes the primary in the new view, it will have received $2f+1$ view-change messages, including one pre-prepare vote and $2f$ prepare votes for request A from nodes 2 and 3. Node 2 then broadcasts the calculated new-view message including request A, transitioning to the new-view phase (replacing the pre-prepare phase) to start a new normal-case operation protocol.

this set of view-change messages. Figure 2 illustrates an execution scenario where both the normal-case operation protocol and the view-change protocol are activated.

3 Redesign of fundamental consensus processes

This section analyzes the foundational consensus processes of BFT protocols, particularly PBFT and its variants. Section 4 will summarize more supplementary optimizations, such as optimistic and trusted hardware. Consequently, discussions of optimistic protocols in this section refer to their slow-path version, which corresponds to the pessimistic version of the protocol. This distinction arises from the complexity of the fundamental consensus processes, where even minor modifications can compromise protocol correctness. Thus, the fundamental consensus processes reveal protocols' essence more effectively. In contrast, supplementary optimizations can be easily layered and are generally applicable, allowing designers to select a fundamental protocol process and overlay multiple supplementary optimizations.

It is worth noting that this paper reviews consensus protocols from a blockchain perspective, and we will treat blocks as the consensus object in the following discussion. Although consensus protocols like PBFT were not originally designed for blockchain, they can be effectively applied within blockchain systems. We will also refer to the normal-case operation protocol as the commitment protocol below.

3.1 Number of commitment phases

The number of commitment phases or good-case latency [54] of a BFT SMR protocol indicates the number of phases needed for all non-faulty replicas to decide when the primary is non-faulty, and the network is synchronous. E.g., PBFT executes in 3 phases.

First of all, it is essential to differentiate between the concepts of phase and step. A phase refers to a significant stage in the protocol processes, typically encompassing multiple steps. In contrast, a step represents the smallest action taken by all nodes within a specific phase. For instance, the PBFT comprises three commitment phases (pre-prepare, prepare, commit), corresponding to three steps. In comparison, the HotStuff protocol consists of four phases (prepare, pre-commit, commit, decide) and seven steps, as shown in Figure 3. The discrepancy in the number of phases and steps arises primarily from HotStuff's use of threshold signature techniques, which often subdivides certain phases into two steps. It is noteworthy that the original text states, "It takes three phases for a basic HotStuff primary to commit a proposal", as it must collect votes in three phases: prepare, pre-commit, and commit. However, we consider HotStuff to be a four-phase protocol. Given that PBFT is commonly referred to

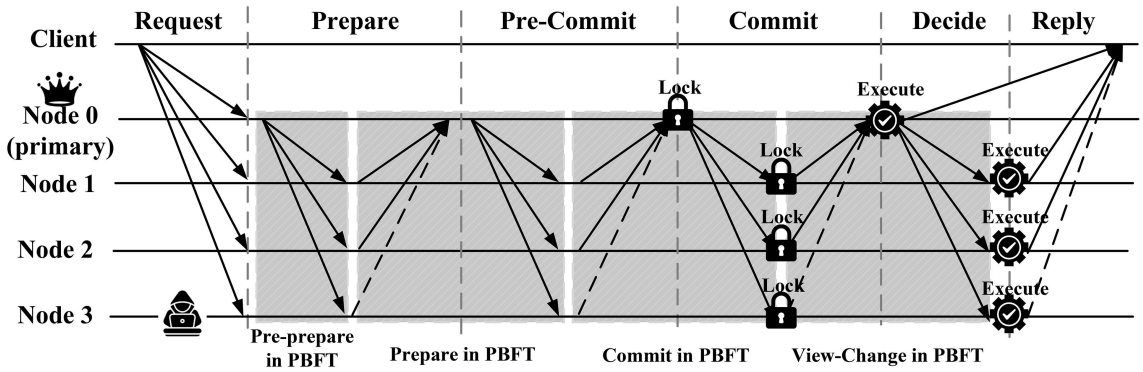

Figure 3 The commitment phases of Hotstuff [42].

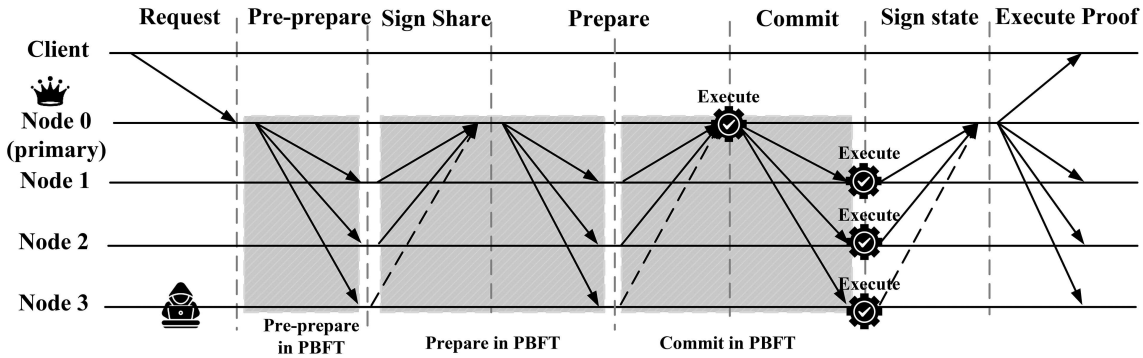

Figure 4 The commitment phases of SBFT (slow path) [41].

as a three-phase protocol and Paxos as a two-phase protocol. To provide a clear comparison of each protocol with PBFT, we define the step where the primary proposes a block as the first phase of the protocol, similar to PBFT.

Moreover, the three-phase HotStuff is a variant of the four-phase HotStuff that lacks optimistic responsiveness. We refer to these two protocols as HotStuff3 and HotStuff4 when necessary. HotStuff3 is somewhat analogous to the linear PBFT, with the distinction that it employs a locking mechanism and changes the primary each round.

3.1.1 *Three phases*

CFT protocols, such as Paxos [11] and Raft [12], utilize a two-phase commitment process. In contrast, PBFT introduces a three-phase commitment process to address Byzantine issues, with the additional phase specifically designed to mitigate ambiguity from a Byzantine primary. If we designate the collection of more than a quorum of votes in the same phase for block b as the QC for block b, this extra phase ensures that block b can only be committed once it is confirmed that a sufficient number of correct nodes have received its QC. Therefore, if a node decides to block b then view changes, its QC will be communicated to the new primary by at least one correct node, which in turn will relay block b's QC, ensuring that all other nodes can also commit to block b.

Many protocols retain the three-phase commitment design. For example, Tendermint's commitment processes closely mirror PBFT but improve liveness by starting a timer after receiving $+2/3$ messages and stopping it once $+2/3$ consistent messages are received to behave better in an asynchronous network. SBFT (slow path) is a linearized version of PBFT, consisting of 3 phases and 5 steps, with the primary aggregating and returning the execution results to clients, as shown in Figure 4.

The three-phase commitment is a classic choice, as much of the work (e.g., various protocols mentioned in Section 4) does not aim to improve the fundamental protocol processes of PBFT but focuses on optimizing other aspects.

Crackle [55] increases protocol parallelism by transforming the linear PBFT into a $2k + 1$ phase process, maintaining the same correctness properties as linear PBFT.

While some optimistic protocols, such as Zyzzyva [56] and SBFT [41], can reduce the number of commitment phases under favorable network conditions, they may revert to linear PBFT mode in the event of a node or network failure.

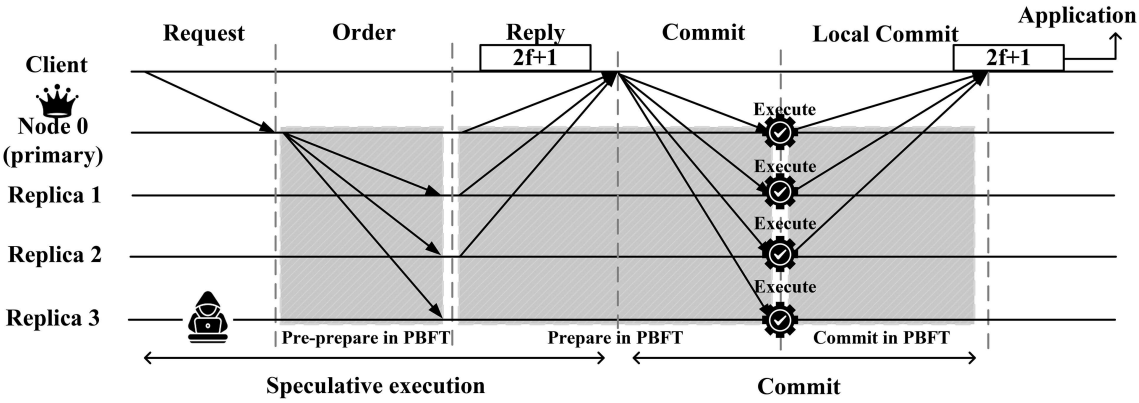

Figure 5 The commitment phases of Zyzzyva (slow path) [56].

3.1.2 *Reducing the number of commitment phases: two phases*

In systems with favorable network conditions, the view-change protocol is rarely triggered, so the system mainly runs the commitment protocol. Simplifying the commitment protocol can significantly enhance the runtime performance of the system, though it may require added redundancy or trusted hardware. There are currently three main approaches to streamlining the commitment phase.

Shifting complexity from the commitment protocol to the view-change protocol. In Zyzzyva's slow path, the commitment protocol involves 2 phases and 4 steps. The primary packages the client's request into a block and sends it to replicas, which forward their votes to the client. If the client receives between $2f + 1$ and $3f$ consistent votes, it broadcasts a commit certificate (CC) to the replicas. Replicas commit the block and send local-commit messages to the client upon receiving the CC. The client considers the block decision after receiving over $2f + 1$ local-commits. This protocol simplifies linear PBFT by omitting the final step, resulting in a two-phase process in Figure 5.

This simplification arises because Zyzzyva's commitment phase concludes at the client rather than at the replicas. However, this introduces several challenges.

(1) The definition of security varies. For PBFT, security means that once a correct replica decides a block, it will ultimately be accepted by all correct replicas, and submissions cannot be overturned. In contrast, for Zyzzyva, a submitted block does not constitute a final submission and may be overturned. A block will only be considered irreversible if the client decides it.

(2) This design creates an asymmetry of information between replicas and clients, leading to the "liveness problem". To address this, the view-change protocol adds a new phase, reducing one phase of the commitment protocol while increasing one in the view-change protocol. This adjustment is considered an optimization only when the view-change frequency is low.

(3) This approach is not especially suitable for blockchain scenarios, as a single block often contains numerous transactions from multiple clients. However, the primary can easily relieve the client.

The liveness issue stemming from the simplification of the commitment protocol occurs when a Byzantine primary colludes with $f - 1$ Byzantine backups, leading $f + 1$ correct nodes to commit the block and send local-commit messages to the client. Simultaneously, the other f correct nodes do not receive the block and broadcast view-change messages, without meeting the minimum conditions for the view-change protocol however. Then Byzantine nodes can remain silent, preventing the client from receiving $2f + 1$ local commits, thus failing to decide the block. In contrast, linear PBFT mitigates this issue by allowing the primary to synchronize the commit votes in the third phase with the backups, enabling replicas to decide based on the situation. Zyzzyva addresses this liveness issue by introducing a new "I hate the primary" phase in the view-change subprotocol, ensuring that if a correct node initiates a view-change message, all correct nodes will eventually send view-change messages.

Phase reduction through redundancy. In PBFT, the three-phase in commitment protocol ensures that at least one correct node forwards the QC corresponding to the block decided by some nodes to the new primary during a view change. The additional prepare phase allows each non-faulty replica to gather prepare messages for the block in order to form a QC, ensuring that more than $2n/3$ replicas hold the QC before the block is decided by any node, so that at least one correct node forwards the QC to the new primary during a view change. An alternative approach would allow the nodes sending QCs to directly transmit enough prepare messages to the new primary. However, this method is constrained by the total number of nodes in PBFT, where $n = 3f + 1$ and

$Q = 2f + 1$ (with n representing the total number of nodes and Q the quorum required for decision-making). This limitation means that, if a block was decided before a view change, the intersection of the sending nodes of the Q commit messages and the Q nodes triggering the view-change protocol can guarantee at least one correct node. Thus, this node must forward a QC, rather than other message type, to ensure the uniqueness of the block (i.e., no other block could have been decided in the previous view). If we increase the total number of system nodes and quorum values, resulting in multiple correct nodes in the intersection of the two quorum sets, could this enable the new primary to directly construct some proof of decided block in the old view, rather than relying on correct nodes to forward QC/AQC? This represents the idea of increasing the replica number to reduce the phase number.

Increasing the replica number to $5f + 1$ ensures that if block b was decided in the previous view, a majority of the Q nodes triggering the view-change will be correct nodes that voted prepare for b. They just need to forward their prepared votes in view-change messages; the new primary will propose b. Protocols of this type include FaB [57] and Zyzzyva5 [56]. In FaB, the new primary queries all nodes, and each responds with its accepted value (corresponding to PBFT's prepared value). For details on replica number, see Subsection 3.2.3, and for view-change protocol specifics, refer to Subsection 3.3.2.

Trusted hardware. Trusted hardware can mitigate the equivocation behavior of Byzantine nodes [19]. Once the primary's equivocation is addressed, view-change messages do not need to include a QC; the primary's pre-prepare message serves as sufficient evidence, as it cannot issue conflicting pre-prepare messages. Thus, the commitment protocol can be streamlined into a two-phase process. In PBFT, if the commit phase is reduced, the block decision indicates that Q nodes have sent consistent prepare messages. If a view-change is initiated, the intersection of the Q nodes that sent prepare messages and those triggering the view-change will include at least one correct node to carry the pre-prepare message into the new view. With trusted hardware, a Byzantine primary cannot produce other valid pre-prepare messages.

Although MinBFT introduces an additional phase to the view-change protocol, it could have implemented a view-change protocol with phases similar to PBFT. The new req-view-change phase serves a purpose akin to Zyzzyva's "I hate the primary", but does not enhance security. While Zyzzyva increases the phases of the view-change to address the liveness problem arising from information asymmetry between clients and replicas, MinBFT does not face this issue. Consequently, MinBFT's view-change protocol can be simplified; i.e., the reduction in commitment phases in MinBFT is due to optimizations from trusted hardware, not from shifting complexity to the view-change protocol.

Trusted hardware effectively simplifies commitment phases. It can reduce the number of commitment phases without compromising other protocol processes. This category of protocols includes Hybster [58], FastBFT [47], and CheapBFT [59]. Some optimizations, such as optimistic, can also reduce commitment phases to two under ideal network conditions, like SBFT (fast path), while others can reduce it to one phase, as seen in Zyzzyva (fast path) and Q/U (fast path). These optimizations will be discussed in detail in Section 4, because only their slow-path version ensures the correctness.

3.1.3 *Increasing the number of commitment phases: four phases*

PBFT's view-change protocol is complex and error-prone. Once initiated, it introduces considerable communication overhead, even in medium-sized systems. Protocols like Tendermint and Casper simplify this by rotating the primary after each round, effectively eliminating the view-change protocol. However, they forego a hallmark of most practical BFT SMR solutions, namely optimistic responsiveness [42, 60], which requires that a non-faulty primary, once designated, can drive the protocol to consensus timely in beneficial circumstances here, after GST is reached.

HotStuff4 achieves optimistic responsiveness through the introduction of one more decide phase and the simplification of the view-change protocol, as detailed in Subsection 3.3.1. Although the additional phase may increase latency and reduce throughput, the linearization and standardization of each phase enable efficient pipelining, minimizing throughput degradation. Furthermore, simplifying the view-change protocol mitigates traffic surges during view changes, thereby enhancing the system's robustness.

Many protocols based on HotStuff4 employ the same four-phase commitment protocol. For instance, in Themis [61], all replicas first submit their locally ordered transactions to the primary, which computes a fair order before running HotStuff4 to reach block consensus. Kauri [62] leverages a tree topology to evenly distribute computational and network load among nodes, while its core protocol remains based on HotStuff4.

3.2 Number of replicas

In this section, we discuss the number of replicas and thresholds for different protocols, along with their theoretical derivation.

3.2.1 *3f+1*

Byzantine fault tolerance protocols generally require at least $3f+1$ replica nodes. This applies to protocols such as PBFT and its numerous variants, including Tendermint [32], HotStuff [42], Zyzzyva [56], SBFT [41], Sphinx [63], Vaap [64], and others. It may originate from the proofs in [27] and represents the optimal fault tolerance achievable without additional mechanisms.

Let the total number of replicas be n, with at most f Byzantine nodes. Each node requires at least Q (Quorum size) messages to make a decision. We now follow the design principles of the PBFT protocol to establish the relationship among the n, Q, and f.

To ensure the protocol's liveness, it is required that $Q \leqslant n-f$ (1), ensuring that sufficient votes can be received even if all Byzantine nodes fail to send messages. For safety, the condition $2Q-n>f$ (2) must be met. Safety can be categorized into intra-view and cross-view safety.

Intra-view safety: If there are Q messages supporting block b, it is impossible to have Q messages supporting a conflicting block b'. The intersection of two sets of Q nodes (has $2Q-n$ nodes) must exceed f nodes; thus, there must be at least one correct node that cannot simultaneously support both conflicting blocks, which leads to the condition $2Q-n>f$.

Cross-view safety: If a node decides block b, it means at least Q nodes (denoted as set A) must have sent commit messages for b. If the view-change protocol is triggered successfully after that, Q nodes (denoted as set B) have sent view-change messages. The intersection of sets A and B must contain at least one correct node, which needs to forward the QC collected before its commit phase. This also requires $2Q-n>f$.

By combining (1) and (2), we obtain $n+f<2Q\leqslant 2n-2f$ (3), leading to the conclusion that $n>3f$. If $n=3f+1$, then $Q=2f+1$. If $n=5f+1$, the condition $3f+1\leqslant Q\leqslant 4f+1$ holds.

3.2.2 *Fewer replicas: 2f+1*

For a long time, protocols with $3f+1$ replicas have been considered to possess an optimal fault tolerance rate. Recently, many protocols have emerged that use hardware to constrain the equivocation behavior of Byzantine nodes, thereby increasing the fault tolerance rate from 1/3 to 1/2. A detailed description of this optimization mechanism can be found in Subsection 4.1. Notable protocols include TTCB [65], A2M-PBFT [66], MinBFT [67], FastBFT [47], and CheapBFT [59], among others.

In such protocols, to ensure the protocol's liveness, it is required that $Q\leqslant n-f$ (1). To ensure security, the condition $2Q-n>0$ (2) must hold, meaning that the intersection of two sets of Q nodes must contain at least one node. While this intersecting node may be a Byzantine node, the trusted hardware ensures that it cannot generate two conflicting messages.

By combining (1) and (2), we obtain $n<2Q\leqslant 2n-2f$ (3), leading to the conclusion that $n>2f$. if $n=2f+1$, then $Q=f+1$. If $n=4f+1$, the condition $2f+1\leqslant Q\leqslant 3f+1$ holds.

3.2.3 *More replicas*

$5f+1$. As described in Subsection 3.2.1, in PBFT, if a node decides block b, Q nodes (denoted as set A) must have sent commit messages for b. If the view-change protocol is triggered successfully after that, Q nodes (denoted as set B) have sent view-change messages. With a total of $3f+1$ replicas, the intersection of sets A and B can ensure at least one correct node. To convince the new primary, this node must deliver "strong evidence", namely the QC. To construct this QC, PBFT adds an additional phase to the commitment protocol. This approach contrasts with the "phase reduction through redundancy" discussed in Subsection 3.1.2.

We raise the question similar to that in Subsection 3.1.2 again: what happens if increasing the number of replicas ensures that the intersection of sets A and B contains multiple correct nodes? Can the new primary then make decisions based on the voting messages rather than the QC? If it can be ensured that the number of correct nodes in set B exceeds half, the new primary will be able to make sound decisions effectively.

In FaB, the commitment protocol consists of 2 rounds: one for proposing and the other for voting, thereby reducing a voting round compared to PBFT. Let n be the total number of nodes in the system, with at most f Byzantine nodes. A node must receive Q_1 consistent voting messages to decide, and Q_2 view-change messages to successfully view change. Thus, FaB can achieve consensus in two phases once the number of Byzantine nodes

does not exceed t (where $t \leqslant f$), ensuring liveness. When the number of Byzantine nodes does not exceed f, the protocol guarantees safety. Specifically, if there are no more than t Byzantine nodes, FaB can reach consensus in the two-phase commitment protocol and can successfully change the Byzantine primary. If the number of erroneous nodes is between t and f, consensus cannot be achieved in the commitment protocol, but safety can be ensured. If the number of erroneous nodes exceeds f, the correctness of consensus cannot be guaranteed.

To ensure the protocol's liveness, it is required that $Q_1 \leqslant n - t$ (1) and $Q_2 \leqslant n - f$ (2). To ensure safety, the condition $Q_1 + Q_2 - n - f > \frac{Q_2}{2}$ (3) must hold. Inequality (3) means that the number of messages from correct nodes in the intersection of Q and Q_2 must exceed half of the messages in Q_2, allowing the new primary to compute a correct proposal based on Q_2. If a node decides on block b, then it follows that more than half of the messages in Q_2 must have voted for b.

By combining (1)–(3), we obtain $n \geqslant 3f + 2t + 1$, $Q_1 \leqslant 3f + t + 1$ and $Q_2 \leqslant 2f + 2t + 1$.

If $t = f$, then $n = 5f + 1$, $Q_1 = Q_2 = 4f + 1$.

Other related protocols include Zyzzyva5 [56], BG[1,1,2] with DP1 [21], among others.

$5f-1$. Abraham et al. [54] established a lower bound of $5f-1$ for two-phase consensus protocols. The distinction from the previously mentioned $5f+1$ is that this paper points out that if the primary is a Byzantine node, and Q_2 contains only $f-1$ Byzantine nodes, the condition (3) transforms to $2Q - n - (f-1) > \frac{Q}{2}$ (without differentiating between Q_1 and Q_2), leading to $n \geqslant 5f - 1$. Protocols like FLB and FTB [20] fall into this category. However, in practical scenarios, view changes can occur not only due to the primary's malicious behavior but also due to network issues, which suggests that the lower bound for two-phase consensus protocols should still be set at $5f + 1$.

$4f + 1$. Sui et al. [21] proposed the first consensus protocol with $4f + 1$ replicas, namely BG[1,1,2] with DP2, featuring a three-phase commitment protocol and a locking mechanism. Compared to PBFT, this protocol increases the number of replicas while still employing a three-phase commitment protocol, which does not align with the "phase reduction through redundancy" approach in Subsection 3.1.2. What, then, is the role of the additional replicas?

Since the number of nodes is less than $5f+1$, the new primary cannot ensure that the majority of the messages in the set of Q view-change messages originate from correct replicas. As a result, the new primary cannot definitively determine the block that was decided in the previous view based on the voting messages. Consequently, the votes set cannot be used to ensure cross-view safety of the protocol. To guarantee the protocol's safety, once a block is decided in the previous view, the new view obtains the block through the QC forwarded by nodes (similar to PBFT). Therefore, the safety and liveness conditions are the same as in PBFT, i.e., $2Q - n > f$ (1) and $Q \leqslant n - f$ (2).

If the set of view-change messages that triggers the view-change mechanism does not contain the QC from the previous view, it indicates that no block was decided in the previous view. However, it is still possible that many nodes have locked onto a block. Therefore, the new primary should ideally propose the block locked in the previous view to mitigate potential liveness issues caused by the locking mechanism (as discussed in Subsection 3.3.1). If a block b was locked in the previous view, more than Q nodes (set A) must have voted for block b. Among the Q view-change messages (from nodes in set B), at least more than f voting messages must come from correct nodes to make further judgments; otherwise, Byzantine nodes could easily collude to launch an attack. To meet this requirement, the intersection of sets A and B must contain more than f correct nodes. Thus, the condition $2Q - n - f > f$ (3) must be satisfied.

By combining (1)–(3), we obtain $n \geqslant 4f + 1$ and $\frac{n+2f+1}{2} \leqslant Q \leqslant n - f$. If $n = 4f + 1$, then $Q = 3f + 1$.

If $5f + 1$ nodes ensure that a locked block can be exclusively carried into the new view through the set of voting messages (as in BG[1,1,2] with DP1), then $4f + 1$ nodes can only ensure that a locked block is non-exclusively carried into the new view. In other words, the locked block will definitely be brought into the new view, but there is a possibility that other unlocked blocks may also be brought in, making it difficult for the primary to distinguish between locked and unlocked blocks. In this case, the primary might propose a new block. Fortunately, this uncertainty only serves to mitigate liveness issues caused by the locking mechanism and does not compromise the protocol's safety.

Therefore, although the protocol improves the liveness issues caused by the locking mechanism, its effectiveness is not as strong as that of Hotstuff or BG[1,1,2] with DP1 (with $5f + 1$ replicas). This protocol may compromise optimistic responsiveness, as Byzantine nodes from the previous view can easily generate conflicting sets of $f + 1$ voting messages in the view-change messages. This situation can prevent the correct primary in the new view from proposing the most suitable value (i.e., a value that all correct nodes would vote for, including locked nodes).

Table 2 Protocols classification according to the design of the view-change protocol. We classify the design of view-change protocols in current consensus protocols based on two dimensions: whether a locking mechanism is used and what content is included in the view-change messages. VC is an abbreviation for view-change in the table, and NV stands for new-view.

	Only QC/AQC ($3f+1$)	Only vote ($5f+1$)	Mixed ($4f+1$)	No view-change (liveness issue)
No lock mechanism (fixed primary) (NV phase forwarding VC messages)	PBFT, Fast-Hotstuff, RBFT, Prime, etc.	FaB, Zyzzyva5, etc.	–	Vaap, Sphinx, etc.
Lock mechanism (rotating primary) (NV phase without VC messages)	Hotstuff, LibraBFT, Narwhal and Tusk, etc.	BG[1,1,2] with DP1	BG[1,1,2] with DP2	Tendermint, Casper, Tezos, etc.

3.3 View-change protocol

In each view, nodes set an initial timer. Receiving specific message sets reset the timer; if it expires, the node stops the commitment protocol and broadcasts a view-change message. When enough nodes initiate the view-change protocol, the view-change is triggered successfully. To explain the design of the view-change protocol in more depth we need to explain some terms first.

QC and AQC. We define a QC as a set of votes from Q distinct nodes for the same block within the same phase. For instance, the Q prepare messages for block b collectively form the prepare QC for b. Although QCs generated in different phases may vary, in this paper, we refer specifically to the QC formed during the second phase of the commitment protocol (the block is proposed in the first phase). QCs come in two formats: a set of voting messages or a single message created by combining multiple votes using a threshold or aggregated signature. The latter is referred to as AQC which can greatly reduce protocol complexity when forwarded.

Locking mechanism. The locking mechanism activates during the commitment protocol. When a node receives sufficient QC, it locks onto the current consensus block b. After a view change, correct nodes only propose or vote for their locked block b unless QC for a conflicting block b' is received, at which point they unlock and vote for b'. If a node decided b in the previous view, more than Q nodes are locked on b. The view-change messages ensure that the new primary can compute a cross-view consistent block, but if the faulty primary proposes a conflicting block b', the locking mechanism prevents it from forming QC.

View-change message. The view-change message contains the consensus state of the current view, with key content aligning with the four possible cases outlined in Table 2. The new primary must gather sufficient view-change messages to compute a cross-view safe proposal block.

New-view message. In the new view, the primary node uses a new-view message to make a proposal. For protocols without locking mechanisms, the new-view message must include all view-change messages received by the new primary, allowing replicas to compute a cross-view safe proposal and preventing Byzantine primaries from altering the correct proposal. For protocols with locking mechanisms, the new-view message only needs to include the QC of the new proposed block, as nodes use their local locked values to assist in verification. This approach reduces the authenticator complexity of the new-view phase.

Fixed primary or rotating primary. Additionally, based on the frequency of view changes, the protocol can be categorized into fixed primary and rotating primary types. In fixed primary consensus protocols, the term of a primary has no limit, allowing for multiple rounds within one view, unless the view-change protocol is triggered due to the primary's misbehavior or network issues, as seen in PBFT [31], MinBFT [67], and Zyzzyva [56]. In contrast, rotating primary protocols change the primary node after each round of the commitment protocol, exemplified by Tendermint [32], Hotstuff [42], Spinning [68], Prime [69], and Aardvark [70]. The methods for primary changing include round-robin scheduling and weighted round-robin scheduling [32]. Protocols with locking mechanisms typically employ a rotating primary approach due to the lower cost of view-change. Conversely, protocols without locking mechanisms usually rely on a fixed primary node, initiating the view-change protocol only at certain times, as the cost of view-change is higher.

We classify the design of view-change protocols in current consensus protocols based on two dimensions: whether a locking mechanism is used and what content is included in the view-change messages, as shown in Table 2. From the perspective of what the view-change message includes as proof of the consensus state in the old view, some protocols include only QC/AQC as a proof, some include votes, others include both QC/AQC and votes (Mixed), and some protocols have no view-change protocol at all. From the perspective of using a locking mechanism, current protocols are divided into two categories: those with and those without a locking mechanism.

3.3.1 *QC/AQC in view-change message*

Since a QC/AQC formed for block b in a given view implies that no conflicting block b' can form a QC/AQC in the same view, QCs/AQCs are exclusive within each view. Additionally, QC/AQC carries signatures from different nodes, ensuring authenticity and preventing forgery. To ensure cross-view safety for blocks committed by some nodes, it is sufficient for at least one correct node to send the QC/AQC it got to the primary of the new view. Thus, a replica count of $3f + 1$ is required.

Conversely, it is precisely because such protocols have only $3f + 1$ replicas that, in the worst case, only one correct node can ensure the consensus state of the old view is sent to the new view's primary through a view-change message. As a result, these protocols must incorporate more than three commitment phases, requiring nodes to first receive the QC/AQC before deciding. This is because the QC/AQC, as proof of the consensus state in the old view, is sufficient to assure other nodes.

No lock mechanism. In protocols without a lock mechanism like PBFT, when a view-change is triggered and a node in the old view decides a block b, it must have received commit votes from Q nodes (set A), while the new view's primary received view-change messages from Q nodes (set B). Sets A and B share at least $2Q - n$ nodes, up to $2Q - n - f$ of which are correct. Since nodes that send view-change messages cease participating in the old view's commitment protocol, any overlapping nodes must have sent commit messages prior to their view-change messages. The correct nodes will include the prepared QC for block b in these view-change messages. The new view's correct primary will compute and re-propose b, incorporating all view-change messages into the new-view message. Upon receiving this message, replicas independently verify b as cross-view safe. If the proposal aligns with their computed value, replicas cast prepare votes, initiating the new view's cycle.

With QC included, the number of authenticators for a view-change message is roughly n times that of a message in the commitment phase (i.e., pre-prepare/prepare/commit). Since the new-view message includes multiple view-change messages, its authenticator count is roughly n times that of a view-change message, and n^2 times that of a message in the commitment phase. The message complexities for the view-change and new-view phases are $O(n^2)$ and $O(n)$, respectively, resulting in a total authenticator complexity of $O(n^3)$ for both phases. Thus, the overhead of view-change may become a bottleneck in the protocol.

Fast-Hotstuff [44] is like a linearization of PBFT, using an aggregation signature mechanism to combine voting messages into an AQC, reducing the authenticator complexity of the new-view phase (similar to PBFT's view-change phase) to $O(n)$. During view-change, the primary collects AQCs from Q nodes to form an AQC set (AggQC, replacing the view-change set in PBFT) for broadcast, reducing the authenticator complexity of the prepare phase (similar to PBFT's new-view phase) to $O(n^2)$. Fast-Hotstuff functions correctly even without a locking mechanism, despite incorporating one; that is why we classify it into this category.

Protocols with a similar view-change strategy also include RBFT [43] and Prime [69].

With lock mechanism. The impact of the locking mechanism on the view-change protocol is that the primary does not need to forward the set of view-change messages or AQCs it received during the new-view phase, as replicas do not rely on this set to determine the correctness of the primary's new proposal. Instead, they assess it based on their local locked values. Replicas will vote for the new proposal in the following cases.

(1) The newly proposed block is equal to the locally locked block.

(2) The newly proposed block contains a QC/AQC, which is generated in a newer view than the corresponding view of the locally locked value. For consensus in a chain structure, this rule is modified to check whether the parent block of the new proposed block has a QC/AQC from the newest view. This condition is referred to as the unlocking mechanism, designed to prevent the locking mechanism from compromising liveness.

A representative of this type of consensus protocol is Hotstuff. Hotstuff employs a locking mechanism and a rotating primary mechanism. The primary of the new view calculates a block with the latest AQC based on the set of received new-view messages (corresponding to the view-change set in PBFT). It extends the chain based on this block and broadcasts a prepare message containing the new proposed block and its parent block's AQC. Backups will vote if the AQC is valid and the new block either extends from the backup's locked block or if the parent block is newer than the locally locked block.

Hotstuff recognizes that the simplification of the view-change protocol through the locking mechanism can lead to liveness issues. It constructs a scenario based on linearized Tendermint where consecutive correct primarys cannot make progess: If only one node in the old view has locked on block b when the view-change protocol is triggered, and it has received Q prepare votes from node set A, then there will be at least one correct node in the intersection of set A and the Q nodes that initiated the view-change. This node can only forward its prepare vote in the view-change message, but it is not reasonable for the primary to select it, as the new primary may observe conflicting prepare votes (if the old primary has equivocation behavior). This situation prevents the new primary from determining

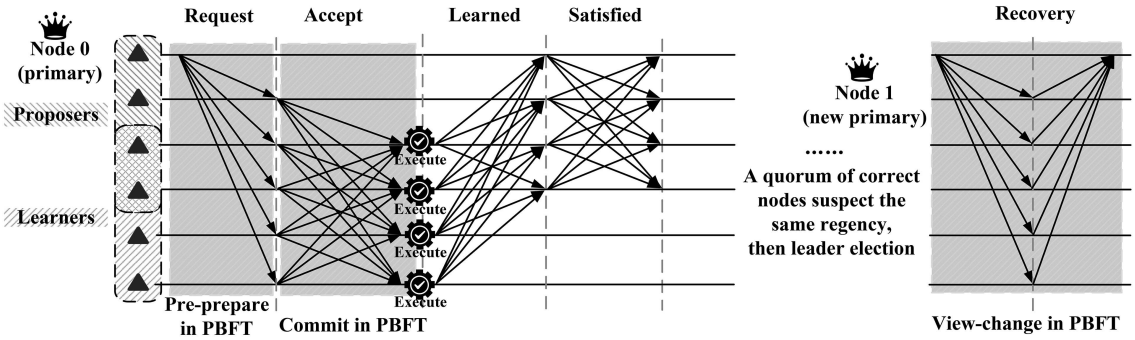

Figure 6 The commitment phases and view-change phase of FaB [57].

the locked block b, and consequently, the nodes locked on b in the old view cannot vote in the new view. This cycle could potentially lead to a protocol liveness issue.

Hotstuff adds an extra phase to the commitment protocol to address the liveness issue, ensuring that once a block is locked, the AQC for the block will be carried into the new view. Referring back to Figure 3, if a node is locked on block b, it indicates that the node received Q votes for b from node set A during the pre-commit phase, and each vote comes from a node that has got the AQC. If the view-change mechanism is triggered at this point, there are Q view-change messages from nodes set B. In the intersection of sets A and B, there will be at least one correct node that forwards its AQC to the new primary, allowing it to propose block b again.

Protocols based on Hutstuff include LibraBFT [71] and Narwhal and Tusk [72].

3.3.2 *Vote in view-change message*

If the view-change messages do not include QC/AQC but only the vote, a sufficient number of votes is required to prove that a value is valid and exclusive once it has been committed in the old view. As mentioned in Subsection 3.2.3, this requirement can be met by increasing the number of replicas. Since there is no need to collect QC, such schemes often simplify the commitment protocol to two phases. However, increasing the number of replicas can also lead to a decrease in fault tolerance.

No lock mechanism. In this type of protocol, FaB is a representative without a lock mechanism. FaB differs significantly in its processes from other protocols; however, the underlying principles remain consistent, as illustrated in Figure 6. FaB incorporates role differentiation, where nodes in the system are divided into proposers and learners, with the primary elected from the proposers. During favorable network conditions, learners can receive proposals through a multicast phase from a primary node. Following this, a voting phase occurs where learners broadcast their votes to one another; once sufficient votes are gathered, the proposal can be decided.

To ensure the liveness of proposers, FaB runs an additional two phases among the proposers after the aforementioned stages. If any proposers do not receive enough messages, a view-change is triggered, electing a new primary among the proposers. During the view-change phase, the new primary generates inquiries to all nodes, which include their voting values in the replies sent back to the primary. The primary then aggregates these replies into a progress certificate, from which it calculates a cross-view safe new proposal. This new proposal, along with the progress certificate, is broadcast to initiate a new round of consensus.

Protocols with a similar view-change strategy also include Zyzzyva5 [56].

With lock mechanism. Sui et al. [21] proposed a three-phase commitment protocol, namely BG[1,1,2] with DP1, which operates with $5f + 1$ replicas and incorporates a lock mechanism. Similar to the $5f + 1$ version of Fast-Hotstuff, the protocol still addresses the issue of the number of correct nodes in two intersections. However, it has the same commitment phases number as Fast-Hotstuff. This raises the question of why it does not conform to the previously proposed principle of "Phase reduction through redundancy". As mentioned in Subsection 3.3.1, in locking mechanisms, the new primary does not need to broadcast the set of view-change messages it received. While this change reduces the overhead of the view-change protocol, it may also introduce liveness issues. To address this, Hotstuff adds an extra phase to ensure that if a node is locked on block b, the QC for block b will be delivered to the next primary during the view-change phase.

It can be said that BG[1,1,2] with DP1 effectively reduces one phase by increasing the number of replicas. However, because it employs a locking mechanism to simplify the new-view messages, an additional phase is introduced to address potential liveness issues caused by the locking mechanism. Ultimately, the result is that the number of phases remains unchanged, while the number of replicas increases, leading to a reduction in the message complexity

of the view-change phase. From this perspective, it can also be interpreted as increasing the number of replicas to lower the message complexity of the view-change protocol.

3.3.3 *Mixed: both QC/AQC and vote in view-change message*

BG[1,1,2] with DP2 in [21] features a three-phase commitment protocol, operates with $4f+1$ replicas, and includes a locking mechanism. Firstly, this protocol maintains three phases, and the view-change messages contain both voting messages and QC/AQC. As a result, during a view change, only one correct node needs to forward the QC to the new primary for the protocol's security, which is analogous to PBFT. Additionally, the protocol utilizes the votes from the second round of the old view to assist the new primary in calculating blocks that may be locked but are not yet decided. If the view-change messages indicate that a unique block b has received more than $f+1$ consistent votes, the new primary will re-propose that block. However, if there are multiple conflicting blocks in the view-change messages, each with more than $f+1$ consistent votes, it suggests that the old primary was a Byzantine node, and the new primary will not adopt any of those blocks. This design aims to mitigate liveness issues introduced by the locking mechanism. However, its effectiveness in improving liveness is not as significant as bringing the locked block exclusively into the new view, as seen in BG[1,1,2] with DP1 (which uses $5f+1$ replicas); it may still compromise optimistic responsiveness.

3.3.4 *No view-change*

Many protocols currently exist that do not include a view-change protocol. The view-change protocol is crucial for ensuring cross-view safety. Although protocols like Tendermint employ a locking mechanism combined with a rotating primary mechanism, thus omitting the view-change protocol entirely, this design has been shown to potentially lead to a liveness problem. Recently, some protocols have emerged in academia that eliminate the need for a view-change protocol. Like Vaap, where every node acts as a primary. While this approach presents an innovative perspective, it also introduces new challenges.

No lock mechanism. In classical designs, protocols without both a view-change and a locking mechanism may seem impossible. However, some unconventional approaches meet this condition. For example, multi-leader protocols like Sphinx [63] and Vaap [64] allow each node to act as a primary, running separate consensus instances. Each instance, in turn, employs a standard consensus protocol, such as PBFT-like consensus for Sphinx and linear PBFT for Vaap. The multi-leader design is not a typical RFCP but rather an ASOM, as it does not follow the logic of interaction between the three basic processes. However, for the sake of the paper's completeness, we mentioned it here, and later also referenced it in Subsection 4.5.2.

The Vaap protocol without a pipeline, as shown in Figure 7, operates as follows: Different clients send transactions to different consensus nodes. Each node runs a consensus instance and acts as the leader of its own instance. Each instance runs a single-leader consensus algorithm, such as Vaap running a linearized PBFT, whose commitment protocol only has two phases because Vaap does not require a view change protocol, and therefore does not need a lock mechanism to simplify the view change protocal, which avoids the need to add extra phases to the commitment protocol to address the liveness issue associated with lock mechanisms. The reason for not requiring a view change protocol is that the multi-leader design ensures that malicious behavior from a leader only affects its own consensus instance, and there is no incentive for the leader to misbehave. Nevertheless, nodes may still act maliciously due to external attacks; thus, Vaap incorporates a rollback mechanism to restore liveness for repaired nodes. This parallel consensus design significantly enhances protocol performance while introducing new challenges, including transaction partitioning and transaction switching.

With lock mechanism. Tendermint lacks a view-change protocol and employs a rotating primary mechanism with a locking mechanism. It has $3f+1$ replicas and a 3-phase commitment protocol, which is very similar to that of PBFT (namely propose, prevote, and pre-commit phases). It may take multiple rounds for a block to achieve consensus, but nodes cast nil votes for invalid blocks, expediting progress. To mitigate frequent view changes due to network delays, Tendermint's last two phases (prevote and pre-commit) include an asynchronous waiting period: replicas initially await Q consensus messages before activating a timer to await Q consistent consensus messages.

From the preceding discussion, we can summarize that two conditions must be satisfied to ensure cross-view security.

(1) The intersection of the node set that triggers a decision and the set that triggers a view-change must ensure that the QC/AQC is relayed to the new primary.

(2) The new view's replicas can verify the correctness of the new proposed block.

The locking mechanism simplifies the condition (2) by allowing the new view's primary to avoid forwarding the view-change set it received, as nodes can rely on their local locked values for judgment. Hotstuff employs the

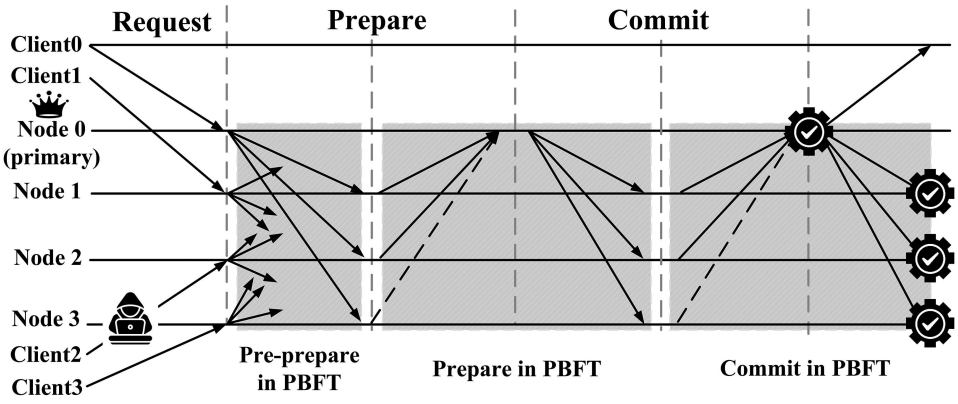

Figure 7 Vaap [64] without pipeline.

locking mechanism to streamline the new-view message but recognizes the potential liveliness issues it introduces, thus adding one more phase. In contrast, Tendermint directly omits the view-change protocol, breaking the condition (1); thus the new primary is unable to get potentially committed or even locked values from view-change messages. Tendermint does not guarantee that the new primary can propose the block decided in the last view, even if a correct new primary may propose a conflicting block, which will not obtain a QC until the primary proposes the decided block again. Furthermore, Tendermint does not ensure that the new primary can propose a block locked in the last view, potentially leading to nodes in the system locking onto different values, resulting in liveliness issues.

Though liveliness issues are theoretically possible, Tendermint's success in the industry shows these issues are rare in real-world conditions.

3.3.5 *Summary*

Subsections 3.1 and 3.2 briefly discuss how view-change message content influences commitment phase number and replica number. In reality, these three are interdependent design elements.

Protocols where view-change messages include a QC/AQC, generally align with a 3-phase commitment protocol and require $3f + 1$ replicas. Those containing only votes tend to have a 2-phase commitment protocol with $5f + 1$ replicas. Following this pattern, adding a locking mechanism allows the new view's primary to avoid forwarding the view-change message set, though it may introduce liveness issues, which may require an additional phase to resolve. Protocols with locking mechanisms can adopt a rotating primary due to the low overhead of view-change. Protocols that incorporate both individual votes and QC/AQC in view-change messages are less common, with BG[1,1,2] with DP2 being an example.

In summary, the view-change protocol is critical for BFT fault tolerance, as omitting it can undermine the protocol's optimistic responsiveness, like Tendermint. Innovative parallel chain approaches, such as Vaap, also introduce new challenges.

4 Addition of supplementary optimization mechanisms

Beyond the fundamental consensus processes design in Section 3, there are extra supplementary optimizations that can be categorized into 5 types, which are largely compatible and can be combined.

Trusted hardware differs from the other five optimizations as it fundamentally affects the protocol's consensus processes. It aligns with the focus of Section 3, where it is frequently mentioned; however, since Section 3 primarily covers process design without auxiliary equipment or mechanisms, we provide additional details on how trusted hardware supports the consensus protocol here.

4.1 Trusted hardware-based consensus

Leveraging trusted hardware for optimizing consensus protocols has emerged as a key research focus. Researchers have developed consensus protocols based on trusted hardware like trusted platform module (TPM) [73], trusted execution environment (TEE) [74], etc. Converting unverifiable resources into verifiable ones within the trusted hardware can enhance the fault tolerance rate of BFT consensus to 1/2.

Proof-of-X type consensus algorithms optimized using trusted hardware include [75, 76]. For blockchain BFT consensus, a key malicious behavior of Byzantine nodes is sending conflicting messages to different nodes, known as equivocation. Many studies address this issue using atomic broadcast via trusted hardware. TTCB [65] implements a trusted atomic multicast service. The primary node first transmits the message hash through a trusted channel, which is synchronized, shared and secure, before broadcasting the complete message over a regular channel. Receiving nodes verify the message by comparing it to the hash from the trusted channel, thus preventing the Byzantine primary from sending conflicting messages to different secondary nodes. A2M-PBFT [66] and HotStuff-M [77] maintain a counter in append-only memory, mapping its incremented value to a message digest to generate a log. This log, along with the message's signature, is broadcast, allowing recipients to verify counter increments. Damysus [78] requires each replica to have two trusted components: an accumulator and a checker. The accumulator helps the primary process messages and create certificates summarizing the last consensus round, while the checker generates sequence numbers using a monotonic counter and logs past transactions. However, these protocols risk reduced responsiveness due to their streamlined design, which limits concurrency [62,79]. MinBFT [67] uses a unique, monotonically increasing counter in a TPM, binding each consensus message to a counter value with a TEE signature to prevent duplicate values. CheapBFT [59] similarly implements this trusted counter in an FPGA-based trusted subsystem. It optimizes for the failure-free case by limiting active replication to $f + 1$ replicas. In the event of a failure, however, CheapBFT requires participation from all $2f + 1$ replicas in the consensus process. Teegraph [80] optimizes DAG consensus by ensuring each event has a single parent event in a TEE, preventing Byzantine nodes from creating forks. Hybster [58] utilizes SGX to virtualize multiple trusted modules, allowing parallel execution of several consensus instances. Recent studies reveal that designing multi-primary protocols is challenging, as up to f Byzantine primaries can collude to compromise liveness [81, 82]. Many researchers integrate secret key sharing and aggregation (or threshold signature) techniques, with consensus protocols to enhance fault tolerant rate and reduce message complexity from $O(n^2)$ to $O(n)$. FastBFT [47] and TBFT [48] utilize TEE for secret key sharing and aggregation, effectively converting Byzantine faults into crash faults. Before executing the consensus protocol, the administrator generates multiple keys in the TEE, binds them to identifiers, splits each key into shards, and distributes them alongside the keys' hash values to the TEEs of replicas, after which the administrator's TEE destroys the original keys. During the consensus processes, the primary assigns a proposal number that indirectly binds the key to a block. When voting, replicas reveal their key shards to the primary, who reconstructs the key once enough shards are collected and broadcasts it to the replicas. The replicas then verify the key's authenticity using the hash value, confirming a successful vote if verification succeeds. RaftTEE [83] and Engraft [84] applied trust hardware on the CFT protocol to achieve Byzantine fault tolerant.

In summary, incorporating trusted hardware is expected to enhance the fault tolerance while reducing its complexity. The use of trusted hardware in consensus protocols involves two trade-offs. The first is the scope of deployment: while early studies required all nodes to use trusted hardware, FlexiBFT [19] argued that only the primary node needs it. The second is the functionality provided: earlier designs focused on simple message mapping to prevent equivocation, whereas TBFT and Damysus expanded trusted hardware capabilities to process messages and create certificates summarizing the last consensus round, reducing message forwarding and protocol complexity. This method also eliminates the need for forwarding view-change messages or AQCs, addressing a challenge that threshold or aggregated signatures cannot effectively solve.

4.2 Better communication topology

Communication topology influences protocol complexity and is classified into five types in current protocols.

(1) Fully connected topology: A topology where all (or a subset of) replicas communicate directly, resulting in quadratic message complexity, such as PBFT.

(2) Gossip topology: Nodes randomly select neighboring nodes to send messages, enabling rapid broadcast. Protocols using this topology include Tendermint, GBC [85], and CCG [86].

(3) Star topology: Communication occurs strictly from a designated replica (e.g., primary) to all other replicas and vice versa, resulting in linear message complexity. Protocols such as Zyzzyva [56], HotStuff [42], Fast-HotStuff [44], and Marlin [87] use this topology.

(4) Tree topology: Replicas are organized in a tree, with the primary at the root, communicating with child or parent replicas each phase. This structure yields logarithmic message complexity, as in Kauri [62], FastBFT [47], ByzCoin [88], MOTOR [89], and X-layer PBFT [90].

(5) Chain topology: Replicas form a pipeline, each communicating only with neighboring replicas, as seen in [91].

Tree and chain topologies are special cases of star topology. In a star topology, the leader fans out to n nodes, while a tree topology fans out to $logn$, and a chain topology fans out to 1. These topologies face a dual dilemma

of leader fan-out and protocol resilience. Star topologies are vulnerable to leader bottlenecks, potentially extending phase durations and negatively impacting overall system performance. Tree and chain topologies impede correct construction in the case of $\frac{n-1}{3}$ resilience, often degrading to star topologies when robust structures are not found. Thus, moving from a fully connected topology to a star topology involves trading off time complexity for message complexity, as the sequential nature of message delivery in star topologies increases broadcast times. Lower fan-out degrees in the leader lead to more balanced loads and reduced single-point bottlenecks, but increase the number of message delivery rounds and overall time complexity.

4.3 Committee-based consensus

The number of replicas participating consensus directly affects the complexity of the consensus protocols. Researchers aim to select a subset of nodes as a committee to participate in consensus and disseminate results, enhancing system scalability. Protocols of this type are classified by Fu et al. [13] as the "Committee + Voting" model.

Committee election strategies vary widely. Some protocols, inspired by PoW [1] and PoS [92], elect committee members based on "proof of some capabilities". DBFT [93] and Tendermint use PoS, linking participation with collateral to address native PoS's Nothing At Stake Attacks. Elastico [94] and ByzCoinX [95] elect leaders via PoW, separating leader election from transaction validation. Permacoin [96] and Spacecoin [97] elect leaders according to participants' available disk space.

We also proposed a committee-based consensus named PoPT [98] in our preliminary work. PoPT ranks nodes by participation in public blockchains for committee elections and leverages PBFT for voting processes. To further improve the committee election fairness and voting efficiency of PoPT, we consider adding nodes' reputation as a ranking factor and replacing PBFT by more efficient voting based on BFT consensus (e.g., Hotstuff). The ranking method incorporates various factors, including the number of times a user serves as a leader, fees paid, and missed commitments. After each consensus round, the leader is removed from the committee, a new member is added based on the rankings, and the second-ranked user becomes the new leader. The newly upgraded protocol helps incentivize honest behavior while punishing malicious actions.

Verifiable random function (VRF) [99] is often used for committee selection. A VRF is a cryptographic scheme that maps inputs to verifiable random outputs, ensuring properties like verifiability, uniqueness, and randomness. Established blockchain projects like Dfinity [45] and Ontology's VBFT [100] utilize VRFs as general pseudorandom functions, where the same input yields the same output. They use the hash of the previous block as input to determine the set of nodes participating in the current consensus round, ensuring the unpredictability of committee members.

Committee selection lowers consensus costs and enhances performance but compromises blockchain decentralization, introducing security risks. Some protocols use anonymous elections to reduce the likelihood of malicious attacks on committee members. This can also be achieved with VRFs, where nodes input agreed-upon data and their private keys to generate a hash and proof. This allows committees selected via VRF to vote anonymously, preventing early exposure to attacks. Algorand proposes BA* [101] using a binomial distribution-based VRF for committee selection. However, the number of elected committees is uncertain, limiting broad applicability to BFT protocols. Feng et al. [102] proposed S2BFT using TEE to elect a fixed number of committees. Additionally, other cryptographic methods, such as the RBFT protocol, utilize ring signatures for anonymous leader elections.

4.4 Optimistic mechanisms

Given that nodes rarely fail when the network is good, an effective optimization strategy is to use optimistic protocols under favorable conditions for rapid consensus, reverting to Byzantine fault tolerant protocols only in case of failures.

4.4.1 *Optimistic phase reduction*

In a linear BFT protocol, assuming no faults in replicas, the function optimistically eliminates two linear phases, equivalent to a single secondary preparation phase, as demonstrated in SBFT [41]. The leader collects signatures from all $3f + 1$ replicas during the second phase, combines them, and sends a signed message to all replicas. Each replica then verifies that all non-faulty replicas have received and agreed on the request, allowing direct submission and skipping the third phase. If the leader fails to receive $3f + 1$ messages within a set timeframe, the protocol reverts to its slower path and executes the third ordering phase. Zyzzyva [56] and its TEE-enhanced version, MinZyzzyva [67], utilize the client as the primary node.

4.4.2 *Optimistic replica reduction*

This class of protocols engages only a subset of replicas in the consensus protocol, while others passively update their states, participating actively only upon consensus failure. Notable examples include CheapBFT [59], FastBFT [47], REBFT [103], and OBFT [104]. These approaches reduce the required replicas from $3f+1$ to $2f+1$, assuming all $2f+1$ replicas are correct. Each phase necessitates a quorum of $2f+1$ matching messages. Once this quorum is achieved, consensus can be reached without the remaining f replicas. The passive replicas update the consensus results but are activated if any active replicas fail. For example, CheapBFT degrades to MinBFT on node failure, while REBFT reverts to PBFT.

4.4.3 *Optimistic conflict-free*

Two main approaches provide Byzantine fault-tolerant consensus: communication-based methods such as BFT and quorum-based methods such as Q/U. The latter assumes requests are conflict-free and that all replicas are fault-free, enabling clients to directly contact replicas for optimistic execution. However, communication-based methods incur unnecessary costs without conflicts, while quorum-based methods struggle under contention. Hybrid protocols [105, 106] have been proposed that leverage a lightweight quorum-based approach when there are no conflicts and switch to BFT to resolve contention when needed.

4.5 Parallel ordering

We classify the strictness of transaction ordering into two categories: totally ordered and partially ordered.

(1) In totally ordered systems, all transactions have a defined sequence. For instance, in traditional blockchain architectures, all blocks (transactions) must be arranged in a strict linear order.

(2) In partially ordered systems, not all transactions have a fixed order; some transactions have a defined sequence while others do not. For instance, consensus protocols based on parallel chains allow transactions to achieve consensus concurrently, establishing order only among transactions on each chain. Directed acyclic graphs (DAGs) are also examples of partially ordered structures, but their consensus mechanisms are more like PoW rather than BFT.

Parallel ordering protocols always use multiple primary nodes running distinct instances of classic primary-replica consensus protocols, where a single node may act as a primary in one instance and a replica in others. These instances operate in parallel, benefiting from the trend of multi-core processors. Based on transaction ordering, the parallel ordering protocol can also be classified into two types: totally ordered and partially ordered.

4.5.1 *Totally ordered*

Totally ordered protocols still require total ordering of all transactions, which introduces non-parallelizable components within the protocol.

Some protocols employ parallelism to enhance fairness or robustness. Prime [69] introduces a preordering stage where replicas exchange client requests and periodically share a vector of received requests, expecting the primary to order following those vectors. RBFT [43] runs multiple PBFT protocol instances in parallel on multicore machines, executing only the master instance's results. Other instances are monitored for performance. If the master underperforms, its primary is considered faulty, initiating a replacement process.

Another group of protocols adopts parallelism to enhance protocol performance. This design typically involves statically partitioning sequence numbers and transactions among nodes, allowing each primary node to independently bind its assigned sequence number set with transactions for consensus in its instance. RCC [81] employs a multi-primary approach to assign sequence numbers to transactions, using rounds to facilitate global execution order. Hybster [58] utilizes multiple virtual trusted execution environments to enable various trusted consensus instances. MirBFT [107] and ISS [82] map requests into different nodes through hashing, with globally assigned sequence numbers statically allocated to nodes, while multiple primary nodes handle transaction-to-sequence number mapping. In Dispel [108], each replica generates a new consensus instance based on its available local resources for resource pipelining. However, slower replicas may hinder system performance. To address this, DQBFT [109] distinguishes between two types of instances: multiple D-instances for parallel independent consensus, achieving local transaction ordering, and one O-instance for global ordering of D-instance consensus results. This strategy avoids waiting for slower nodes, thus preventing them from becoming performance bottlenecks. PPoV [110–112] is a parallel extension of proof of vote (PoV) [113, 114], where multiple butlers generate and broadcast blocks concurrently in each consensus round, while a leader aggregates votes.

4.5.2 *Partially ordered*

The advantages of multiple primaries are more pronounced in partially ordered protocols. If there are a few conflicts between transactions from different primary nodes, these protocols can maximize the benefits of parallelism by executing consensus concurrently. This approach prevents increased client latency due to a minority of slower nodes in the system.

ezBFT [115] optimizes client latency caused by geographic issues through a multi-primary approach. It implements dependency-based ordering, where replicas submit commands after exchanging dependency metadata and execute them in a deterministic order that satisfies these dependencies. Commands with conflicting operations are totally ordered, while others are partially ordered. This dynamic ordering minimizes the overhead of ordering non-conflicting commands. However, when there are many conflicting commands, such protocols can incur significant overhead, reducing performance.

Sphinx [63] defines a weak consensus that relaxes the guarantee of strict consistency. Multiple primary nodes propose different blocks, and each node only needs to ensure the relative order of blocks from the same primary node on its local chain, regardless of how many other blocks from different nodes are inserted in between. However, this interdependence among multiple chains can lead to performance degradation. Vaap [64] introduces a parallel chain data structure where each node serves as the primary for different parallel chains, assuming no conflicts between transactions on different chains. This allows for fully parallel consensus on each chain, thereby avoiding performance bottlenecks associated with single nodes.

It is evident that parallel ordering typically involves the design of multiple primary nodes, with partially ordered protocols achieving higher parallelism than totally ordered protocols.

4.6 Pipelining

To reduce message complexity, a new trend assigns the primary the task of collecting votes from replicas and aggregating them using a threshold or aggregated signature mechanism. This optimization replaces an all-to-all broadcast phase with a single receive-send phase, standardizing the commitment protocol phase. Thus the primary can send the current block along with the AQC of the last block proposed in the previous phase, enabling pipelined parallelism. Protocols utilizing this approach include HotStuff [42], Fast-HotStuff [44], Vaap [64], and Casper FFG [116].

Most protocols utilizing pipelining optimizations have parallelism that matches the number of commitment phases. To enhance parallelism, Crackle [55] modifies the HotStuff3 into $2k+1$ phases, simplifying each sub-phase to achieve improved pipelined parallelism.

4.7 Summary

This section summarizes six higher-level optimizations of consensus protocols. Unlike the design of the fundamental protocol processes, the mechanisms discussed here are independent and can be easily combined.

The current communication topology of consensus protocols is categorized into five types, where tree and chain topologies are special cases of star topologies. Optimizing communication topology can achieve load balancing among nodes, preventing any single node from becoming a bottleneck. However, each topology has its pros and cons, requiring careful selection based on specific needs.

The committee election approach sacrifices some decentralization to enhance protocol efficiency but may reduce security, making it susceptible to attacks such as Sybil attacks, eclipse attacks, and denial-of-service attacks. However, increasing the frequency of committee rotation and incorporating technologies like VRF can help mitigate these risks by introducing anonymity.

Optimistic protocols can significantly enhance performance under good network conditions.

The use of pipelining necessitates combining threshold signatures or key sharing and aggregation techniques to standardize protocol processes. The degree of parallelism is proportional to the number of phases in the protocol; however, an excessive number of phases may increase latency, creating a trade-off.

Integrating trusted hardware fundamentally protects vulnerable parts of consensus protocols from manipulation by attackers, eliminating malicious behavior by nodes and improving the fault tolerance rate to 1/2.

Parallel ordering protocols typically feature multiple primaries. When there are few conflicting transactions within the system, parallel sorting can greatly improve protocol efficiency.

Table 3 Optimizations affecting the fundamental consensus processes. Category I includes four subtypes based on two factors: the using scope of trusted hardware and the functionality implemented in it. In terms of using scope, trusted hardware is either used only (a) on the primary (e.g., FlexiBFT) or (b) on all nodes (e.g., MinBFT). (a) requires $3f+1$ nodes to ensure that at least one correct node brings the committed block into the next view during view change. (b) only requires $2f+1$ nodes, as trusted hardware prevents all nodes from equivocation. Regarding functionality, it is either (i) limited to a trusted counter to resolve node equivocation (e.g., MinBFT) or (ii) extended to allow the primary node to securely compute a new-view message from view-change messages (e.g., TBFT). (i) resolves the primary's equivocation, ensuring no conflicting blocks for Byzantine replicas to vote on. Thus, the view-change message does not need a QC, reducing its authenticator complexity to O(1). (ii) addresses the primary's equivocation and derives the secure new-view message from the set of view-change messages, eliminating the need to broadcast the set, and reducing the authenticator complexity of the new-view phase to O(n). These combinations provide four usage scenarios, balancing optimization benefits with added costs. Category III has two subtypes. Optimization III[2] uses threshold signatures or aggregated signatures to efficiently aggregate QC into AQC, reducing protocol complexity. Optimization III[1] requires threshold signatures for linearization. III[1] and III[2] aggregate QC into AQC, reducing the authenticator complexity of the view-change message to O(1). Building on this, III[1] reduces the authenticator complexity of all phases except new-view to O(n), as the new-view phase still requires broadcasting the view-change message set. VC is an abbreviation for view-change in the table, and NV stands for new-view.

Category		Optimization option	Phases (steps)	Replicas	Authenticator complexity and view-change protocol
I	Primary use trusted hardware	[1] Only fix equivocation (e.g., FlexiBFT)	-1	$3f+1$	VC message to $O(1)$: without QC
		[2] Fix equivocation and secure NV message computing	-1	$3f+1$	VC message to $O(1)$: without QC; NV phase to $O(n)$: no VC messages forwarding
	All nodes use trusted hardware	[3] Only fix equivocation (e.g., MinBFT)	-1	$2f+1$	VC message to $O(1)$: without QC
		[4] Fix equivocation and secure NV message computing (e.g., TBFT)	-1	$2f+1$	VC message to $O(1)$: without QC; NV phase to $O(n)$: no VC messages forwarding
II		[1] Phase reduction through redundancy	-1	$+2f$	VC message to $O(1)$: without QC
III		[1] Votes aggregation and linearization	–	–	VC message to $O(1)$: aggregating QC into AQC; All phases to $O(n)$ except NV phase
		[2] Votes aggregation	–	–	VC message to $O(1)$: aggregating QC into AQC
IV		[1] Lock mechanism	$+1$	–	NV phase to $O(n)$: no VC messages forwarding
V $\otimes$		[1] Non-responsive lock mechanism	–	–	No view-change protocol
		[2] Shifting complexity from commitment to view-change protocol	2	–	Adding one phase to VC protocol
		[3] Resilience	–	$+k(2f)$	–

5 A framework for rapid analysis and design of blockchain BFT protocols

This section proposes a framework for rapid analysis and design of blockchain BFT consensus protocols. Using PBFT as a baseline, it classifies optimizations affecting fundamental consensus processes into 5 categories. From these, 26 protocols are derived: 22 secure, 9 capture existing protocols, and 17 novel and optimal. Additionally, we list 5 options that do not impact fundamental consensus processes, which can generate hundreds of variants when combined with the secure protocols above. This framework can help researchers in designing or selecting blockchain consensus protocols.

We categorize common optimizations into two types based on whether they impact on the protocol's fundamental consensus processes. Table 3 lists optimizations that affect the fundamental consensus processes: Categories I–IV are secure and optimal, while Category V includes methods that may affect correctness or are suboptimal but used in existing protocols.

To explore the various possibilities of the fundamental consensus processes, we propose a method for designing fundamental BFT variants called BD, which combines the optimizations in Table 3 (excluding problematic options in Category V) using four-dimensional indices. Each index corresponds to an optimization option from Categories I–IV, with values indicating the applicable subcategories; a value of 0 means the optimization is not applicable. For instance, the first index has a range of 0–4. Optimization options within the same category cannot be combined (e.g., I[1] and I[2]). We will then eliminate unreasonable combinations from these tuples.

The final results are presented in Table 4, showing the optimal fundamental consensus processes metrics achievable with the corresponding optimization options. Some combinations capture existing protocols, while others represent new protocols identified by this theory. Users can choose specific combinations based on their scenario characteristics and derive optimal consensus metrics for particular combinations. Additionally, protocols with correctness issues or suboptimal solutions are listed at the end of the table.

Table 4 Blockchain BFT protocols generated using combinations of optimizations in Table 3. The number in parentheses under the second column indicates the steps number in the commitment protocol. We list only the overall complexity for the commitment protocol, while providing the complexity of each phase for the view-change protocol to help readers understand the details. In the final column, the equal sign "=" denotes a variant that exactlly captures the existing protocol. Variants marked with symbol "•" represent new protocols captured by this framework. The symbol "≈" introduces potentially similar existing protocols, followed by an explanation of how they are not fully optimized compared to the generated protocol.

Protocol	Phase	Replicas	Message complexity		Authenticator complexity		Captured protocols or similar protocols
			Commitment	View-change	Commitment	View-change	
BD[0,0,0,0]	3	$3f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^3)/O(n^3)$	= PBFT
BD[1,0,0,0]	2	$3f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n^2)$	= FlexiBFT
BD[2,0,0,0]	2	$3f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n)$	•
BD[3,0,0,0]	2	$2f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n^2)$	•
BD[4,0,0,0]	2	$2f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n)$	•
BD[0,1,0,0]	2	$5f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n^2)$	= FaB
BD[0,0,1,0]	3(5)	$3f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n^2)$	• ≈ Fast-Hotstuff without locking mechanism
BD[0,0,2,0]	3	$3f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n^2)$	•
BD[0,0,0,1]	4	$3f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^3)/O(n)$	• ≈ Tendermint with view-change protocol and more commitment phases
BD[1,0,1,0]	2(3)	$3f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n^2)$	• ≈ FlexiBFT with linearization
BD[2,0,1,0]	2(3)	$3f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n)$	•
BD[3,0,1,0]	2(3)	$2f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n^2)$	•
BD[4,0,1,0]	2(3)	$2f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n)$	• ≈ TBFT with fewer phases of commitment and view-change protocols
BD[1,0,0,1]	3	$3f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n)$	•
BD[3,0,0,1]	3	$2f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n)$	•
BD[0,1,1,0]	2(3)	$5f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n^2)$	• ≈ Zyzzyva5 (slow path)
BD[0,1,0,1]	3	$5f+1$	$O(n^2)$	$O(n^2)/O(n)$	$O(n^2)$	$O(n^2)/O(n)$	• ≈ BG[1,1,2] with DP1 without linearization
BD[0,0,1,1]	4(7)	$3f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n)$	= Hotstuff
BD[0,0,2,1]	4	$3f+1$	$O(n^2)$	$O(n)/O(n)$	$O(n^2)$	$O(n)/O(n)$	• ≈ The Hotstuff without linearization
BD[1,0,1,1]	3(5)	$3f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n)$	•
BD[3,0,1,1]	3(5)	$2f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n)$	•
BD[0,1,1,1]	3(5)	$5f+1$	$O(n)$	$O(n)/O(n)$	$O(n)$	$O(n)/O(n)$	= BG[1,1,2] with DP1
The following are some protocols that have correctness defects or are not optimal solutions							
V[1]	3	$3f+1$	$O(n^2)$	–	$O(n^2)$	—	= Tendermint
III[1]+V[2]	2(3)	$3f+1$	$O(n)$	$O(n^2)/O(n^2)$ $/O(n)$	$O(n)$	$O(n^2)/O(n^3)$ $/O(n^3)$	= Zyzzyva (slow path), incomplete linearization
II[1]+V[3]	2	$7f+1$	$O(n^2)$	$O(n^2)/O(n^2)$	$O(n^2)$	$O(n^2)/O(n^2)$	= Bosco
III[1]+ V[2]+V[3]	2(3)	$5f+1$	$O(n)$	$O(n^2)/O(n^2)$ $/O(n)$	$O(n)$	$O(n^2)/O(n^3)$ $/O(n^3)$	= Zyzzyva5 (slow path), incomplete linearization

To elaborate on the process of generating Table 4, we first provide several definitions and theorems.

Given two optimization mechanisms A and B,

Definition 1. If A performs better than B in terms of dimension x, let $A >_{(x)} B$. Specifically, with respect to B's optimization, if $x = phases$, A reduces the number of phases more; if $x = replicas$, A reduces the number of nodes more; if $x = view\text{-}change$, A simplifies the view-change phase more.

Definition 2. If optimization mechanism A performs better than or is at least as good as B in terms of dimension x, let $A \geqslant_{(x)} B$. If $A >_{(x)} B$, then it is guaranteed that $A \geqslant_{(x)} B$.

Definition 3. If $\forall x \in \{phases, replicas, view\text{-}change\}$, we have $A \geqslant_{(x)} B$; then we say $A \geqslant B$.

Definition 4. If $A \geqslant B$ and $\exists x \in \{phases, replicas, view\text{-}change\}$ such that $A >_{(x)} B$, then we say $A > B$. If $A > B$, it is guaranteed that $A \geqslant B$.

Let $A + B$ represent the combined optimization mechanism of A and B,

Theorem 1. If $A \geqslant B$, then adding optimization B on top of optimization A is redundant. Using the combined

optimization mechanism $A+B$ is less effective than using only optimization A. Therefore, we exclude the combined optimization mechanism $A+B$.

Theorem 2. If A is excluded, then $A+B$ is also excluded.

Based on the above definitions and theorems, we can exclude unreasonable algorithms from the permutations and combinations, The specific steps are as follows.

(1) For $\forall i \in \{1,2,3,4\}$, since I[$i$] $\geqslant_{(phases)}$ II[1], I[i] $>_{(replicas)}$ II[1], and I[i] $\geqslant_{(view\text{-}change)}$ II[1], it follows that I[i] > II[1]. Based on Theorem 1, the combined optimization I[i]+II[1], i.e., BD[1, 1, 0, 0], BD[2, 1, 0, 0], BD[3, 1, 0, 0], and BD[4, 1, 0, 0], is excluded.

(2) For $\forall i \in \{1,2,3,4\}$, since I[$i$] $>_{(phases)}$ III[2], I[i] $\geqslant_{(replicas)}$ III[2], and I[i] $\geqslant_{(view\text{-}change)}$ III[2], it follows that I[i] > III[2]. Based on Theorem 1, the combined optimization I[i] + III[2], i.e., BD[1, 0, 2, 0], BD[2, 0, 2, 0], BD[3, 0, 2, 0], and BD[4, 0, 2, 0], is excluded.

(3) For $\forall i \in \{2,4\}$, since I[i] $>_{(phases)}$ IV[1], I[i] $\geqslant_{(replicas)}$ IV[1], and I[i] $>_{(view\text{-}change)}$ IV[1], it follows that I[i] > IV[1]. Based on Theorem 1, the combined optimization I[i] + IV[1], i.e., BD[2, 0, 0, 1] and BD[4, 0, 0, 1], is excluded. (Note: I[i] $>_{(view\text{-}change)}$ IV[1] holds because IV[1] only reduces the authenticator complexity of the new-view phase to $O(n)$, whereas I[2] or I[4] further reduces the authenticator complexity of view-change messages to $O(1)$.)

(4) II[1] and III[2] are not directly comparable, but it is easy to see that II[1] $\geqslant$ II[1] + III[2]. Based on Theorem 1, the combined optimization II[1] + III[2], i.e., BD[0, 1, 2, 0], is excluded.

(5) According to Theorem 2, the following cases are eliminated: BD[1,1,0,1], BD[1,1,1,0], BD[1,1,2,0], BD[1,1,1,1], and BD[1,1,2,1] due to the exclusion of BD[1,1,0,0]; BD[2,1,0,1], BD[2,1,1,0], BD[2,1,2,0], BD[2,1,1,1], and BD[2,1,2,1] due to the exclusion of BD[2,1,0,0]; BD[3,1,0,1], BD[3,1,1,0], BD[3,1,2,0], BD[3,1,1,1], and BD[3,1,2,1] due to the exclusion of BD[3,1,0,0]; BD[4,1,0,1], BD[4,1,1,0], BD[4,1,2,0], BD[4,1,1,1], and BD[4,1,2,1] due to the exclusion of BD[4,1,0,0] (see step (1)). Similarly, BD[1,0,2,1] is eliminated due to the exclusion of BD[1,0,2,0]; BD[2,0,2,1] due to BD[2,0,2,0]; BD[3,0,2,1] due to BD[3,0,2,0]; and BD[4,0,2,1] due to BD[4,0,2,0] (see step (2)). Furthermore, BD[2,0,1,1] is eliminated due to BD[2,0,0,1], and BD[4,0,1,1] due to BD[4,0,0,1] (see step (3)). Finally, BD[0,1,2,1] is eliminated due to BD[0,1,2,0] (see step (4)). Consequently, among the algorithms incorporating three or more optimization mechanisms, only BD[0,1,1,1], BD[1,0,1,1], and BD[3,0,1,1] remain.

According to the above rules, Table 4 has been modified as follows: BD[1, 0, 0, 1], BD[3, 0, 0, 1], and BD[0, 1, 1, 1] are moved forward, while BD[1, 0, 1, 1] and BD[3, 0, 1, 1] are added. Additionally, a more detailed explanation is provided for the algorithms in the last column marked with "≈", highlighting the differences between the generated protocols and existing studies.

BD[0, 0, 1, 0] is similar to Fast-HotStuff but lacks its lock mechanism, which is somewhat redundant in Fast-HotStuff as it does not simplify the prepare phase (similar to PBFT's new-view phase). Instead, it reduces the prepare phase's authenticator complexity to $O(n^2)$ using aggregated signatures.

BD[0, 0, 1, 0] is similar to Fast-HotStuff but lacks its lock mechanism, which is somewhat redundant in Fast-HotStuff as it does not simplify the prepare phase (similar to PBFT's new-view phase). Instead, it reduces the prepare phase's authenticator complexity to $O(n^2)$ using aggregated signatures.

BD[1, 0, 1, 0] is similar to FlexiBFT with linearization.

BD[4, 0, 1, 0] is akin to TBFT, but with simpler commitment and view-change protocols.

BD[2, 0, 1, 0] and BD[4, 0, 1, 0] adopt only the linearization design from III[1], without utilizing vote aggregation. This is because trusted hardware can replace vote aggregation by tallying votes and generating next phase messages (extending the logic of I[2] and I[4] to other phases). From this perspective, III[1] could be further divided to separate linearization as an independent design choice. However, since linearization alone does not optimize algorithm metrics, we did not list it separately to keep the framework simple.

BD[1, 0, 0, 1] and BD[3, 0, 0, 1] reduce only the new-view authenticator complexity but not the view-change complexity, keeping the overall authenticator complexity of the view change protocol at $O(n^2)$. From this perspective, BD[1, 0, 0, 1] is no better than [1, 0, 0, 0], and BD[3, 0, 0, 1] is no better than [3, 0, 0, 0].

BD[0, 1, 1, 0] resembles the slow path of Zyzzyva5. BD[0, 1, 0, 1] is like "BG[1, 1, 2] with DP1" without linearization. BD[0, 0, 2, 1] is similar to HotStuff without linearization.

There are five optimizations that do not affect the fundamental consensus processes of the protocol, as shown in Table 5.

Building on the 22 secure optimal combinations in Table 4, incorporating the options from Table 5 can yield over 500 variants for user selection. This theory has the following applications.

(1) By analyzing the optimization measures on a consensus protocol, one can assess whether it has been fully optimized or if there are correctness concerns.

Table 5 Optimizations not affecting the fundamental consensus processes.

ASOM	Features	Compatibility
Better communication topology	Transitioning from a fully connected topology to a star topology optimizes protocol complexity with linearization. Moving to tree and chain topologies further reduces node fan-out and balances load but may increase time complexity and complicate the search for robust structures.	Compatible with all entries in Table 4
Committee-based consensus	Selecting a subset of nodes as a committee can accelerate the consensus process, though it may introduce security risks.	Compatible with all entries in Table 4
Optimistic mechanisms	In favorable network conditions, a lightweight consensus protocol can be used to speed up the process, with fallback to the base protocol ensuring correctness under network or malicious node issues.	Compatible with all entries in Table 4
Pipelining	Standardizing linearized consensus protocols enable new blocks to be proposed during the voting phase of the last block, improving parallelism and speeding up consensus.	Compatible with the 10 entries containing III[1] optimization in Table 4
Parallel ordering	Running multiple consensus instances enhance protocol robustness or performance.	Compatible with all entries in Table 4

(2) When provided with the target performance metrics of a consensus protocol, users can identify the necessary optimization combinations to achieve the goal.

(3) Users with the ability to implement certain optimizations can examine the optimal results of their protocol.

6 Conclusion

This paper analyzes the design principles of partially synchronous BFT consensus protocols for blockchain applications, using PBFT as a baseline. It first organizes existing protocols from the perspective of fundamental consensus processes, identifying how interactions between process stages influence one another and explaining common design patterns. Five stackable optimization mechanisms are then identified. Finally, a framework for designing consensus protocols is proposed, categorizing optimizations that affect fundamental consensus processes into five types and quantifying their impact. Combining these optimizations yields 17 secure consensus protocols, with an additional five optimizations producing hundreds of potential variants. This framework supports users in customizing consensus protocols for specific application needs.

Acknowledgements This work was supported by National Key R&D Program of China (Grant No. 2022ZD0115302), National Natural Science Foundation of China (Grant Nos. 62202479, 61772030, 62472432), Major Program of Xiangjiang Laboratory (Grant No. 22XJ01004), and Major Project of Technology Innovation of Hunan Province (Grant No. 2021SK1060-1).

References

1 Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf
2 Drosatos G, Kaldoudi E. Blockchain applications in the biomedical domain: a scoping review. Comput Struct Biotechnol J, 2019, 17: 229–240
3 Kshetri N. 1 Blockchain's roles in meeting key supply chain management objectives. Int J Inf Manage, 2018, 39: 80–89
4 Min H. Blockchain technology for enhancing supply chain resilience. Business Horizons, 2019, 62: 35–45
5 Macrinici D, Cartofeanu C, Gao S. Smart contract applications within blockchain technology: a systematic mapping study. Telematics Inf, 2018, 35: 2337–2354
6 Al-Jaroodi J, Mohamed N. Blockchain in industries: a survey. IEEE Access, 2019, 7: 36500–36515
7 Casino F, Dasaklis T K, Patsakis C. A systematic literature review of blockchain-based applications: current status, classification and open issues. Telemat Inf, 2019, 36: 55–81
8 Pilkington M. Blockchain Technology: Principles and Applications. Cheltenham: Edward Elgar Publishing, 2016. 225–253
9 Korpela K, Hallikas J, Dahlberg T. Digital supply chain transformation toward blockchain integration. In: Proceedings of the Hawaii International Conference on System Sciences, 2017. 4182–4191
10 Oki B M, Liskov B H. Viewstamped replication: a new primary copy method to support highly-available distributed systems. In: Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, 1988. 8–17
11 Lamport L. Paxos made simple. ACM SIGACT News, 2001, 32: 51–58
12 Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In: Proceedings of the 2014 USENIX Annual Technical Conference, 2014. 305–319
13 Fu X, Wang H M, Shi P C. A survey of Blockchain consensus algorithms: mechanism, design and applications. Sci China Inf Sci, 2021, 64: 121101
14 Berger C, Reiser H P. Scaling Byzantine consensus: a broad analysis. In: Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, 2018. 13–18
15 Correia M, Veronese G S, Neves N F, et al. Byzantine consensus in asynchronous message-passing systems: a survey. Int J Crit Comput-Based Syst, 2011, 2: 141–161

16 Platania M, Obenshain D, Tantillo T, et al. On choosing server- or client-side solutions for BFT. ACM Comput Surv, 2016, 48: 1–30
17 Distler T. Byzantine fault-tolerant state-machine replication from a systems perspective. ACM Comput Surv, 2022, 54: 1–38
18 Zhang G, Pan F, Mao Y, et al. Reaching consensus in the Byzantine empire: a comprehensive review of BFT consensus algorithms. ACM Comput Surv, 2024, 56: 1–41
19 Gupta S, Rahnama S, Pandey S, et al. Dissecting BFT consensus: in trusted components we trust! In: Proceedings of the Eighteenth European Conference on Computer Systems, 2023. 521–539
20 Amiri M J, Wu C, Agrawal D, et al. The bedrock of Byzantine fault tolerance: a unified platform for BFT protocols analysis, implementation, and experimentation. In: Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation, 2024. 371–400
21 Sui X, Duan S, Zhang H. BG: a modular treatment of BFT consensus toward a unified theory of BFT replication. IEEE TransInform-Forensic Secur, 2023, 19: 44–58
22 Xiang Z, Malkhi D, Nayak K, et al. Strengthened fault tolerance in Byzantine fault tolerant replication. In: Proceedings of the 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), 2021. 205–215
23 Schneider F B. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput Surv, 1990, 22: 299–319
24 Cui H, Gu R, Liu C, et al. Paxos made transparent. In: Proceedings of the 25th Symposium on Operating Systems Principles, 2015. 105–120
25 Attiya H, Welch J. Distributed Computing: Fundamentals, Simulations, and Advanced Topics. Hoboken: John Wiley & Sons, 2004
26 Schneider F B. Byzantine generals in action: implementing fail-stop processors. ACM Trans Comput Syst, 1984, 2: 145–154
27 Lamport L, Shostak R, Pease M. The Byzantine generals problem. In: Concurrency: the Works of Leslie Lamport. New York: ACM Books, 2019. 203–226
28 Dwork C, Lynch N, Stockmeyer L. Consensus in the presence of partial synchrony. J ACM, 1988, 35: 288–323
29 Fischer M J, Lynch N A, Paterson M S. Impossibility of distributed consensus with one faulty process. J ACM, 1985, 32: 374–382
30 Dolev D, Dwork C, Stockmeyer L. On the minimal synchronism needed for distributed consensus. J ACM, 1987, 34: 77–97
31 Castro M, Liskov B. Practical Byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, 1999. 173–186
32 Buchman E, Kwon J, Milosevic Z. The latest gossip on BFT consensus. 2028. ArXiv:1807.04938
33 Floyd R W. Assigning meanings to programs. In: Program Verification. Studies in Cognitive Systems. Dordrecht: Springer, 1993. 65–81
34 David B, Magri B, Matt C, et al. Gearbox: optimal-size shard committees by leveraging the safety-liveness dichotomy. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022. 683–696
35 Boneh D, Lynn B, Shacham H. Short signatures from the Weil pairing. J Cryptol, 2004, 17: 297–319
36 Cachin C, Kursawe K, Shoup V. Random oracles in constantipole: practical asynchronous Byzantine agreement using cryptography. In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, 2000. 123–132
37 Shoup V. Practical threshold signatures. In: Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques, 2000. 207–220
38 Boneh D, Gentry C, Lynn B, et al. Aggregate and verifiably encrypted signatures from bilinear maps. In: Proceedings of the 22nd International Conference on Theory and Applications of Cryptographic Techniques, 2003. 416–432
39 Boneh D, Drijvers M, Neven G. Compact multi-signatures for smaller blockchains. In: Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, 2018. 435–464
40 Shamir A. How to share a secret. Commun ACM, 1979, 22: 612–613
41 Gueta G G, Abraham I, Grossman S, et al. SBFT: a scalable and decentralized trust infrastructure. In: Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2019. 568–580
42 Yin M, Malkhi D, Reiter M K, et al. Hotstuff: BFT consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, 2019. 347–356
43 Sun H, Zhang W, Wang X, et al. A robust Byzantine fault-tolerant consensus algorithm against adaptive attack based on ring signature and threshold signature. Acta Autom Sin, 2021, 49: 1471–1482
44 Jalalzai M M, Niu J, Feng C, et al. Fast-HotStuff: a fast and robust BFT protocol for blockchains. IEEE Trans Dependable Secure Comput, 2023, 21: 2478–2493
45 Hanke T, Movahedi M, Williams D. Dfinity technology overview series, consensus system. 2018. ArXiv:1805.04548
46 Tang F, Peng J, Wang P, et al. Improved dynamic Byzantine fault tolerant consensus mechanism. Comput Commun, 2024, 226–227: 107922
47 Liu J, Li W, Karame G O, et al. Scalable Byzantine consensus via hardware-assisted secret sharing. IEEE Trans Comput, 2018, 68: 139–151
48 Zhang J, Gao J, Wang K, et al. TBFT: efficient Byzantine fault tolerance using trusted execution environment. In: Proceedings of the IEEE International Conference on Communications, 2022. 1004–1009
49 Miller A, Xia Y, Croman K, et al. The honey badger of BFT protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016. 31–42
50 Guo B, Lu Z, Tang Q, et al. Dumbo: faster asynchronous BFT protocols. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020. 803–818
51 Guo B, Lu Y, Lu Z, et al. Speeding dumbo: pushing asynchronous BFT closer to practice. Cryptology ePrint Archive, 2022. https://eprint.iacr.org/2022/027
52 Basu S, Tomescu A, Abraham I, et al. Efficient verifiable secret sharing with share recovery in BFT protocols. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019. 2387–2402
53 Bebel J, Ojha D. Ferveo: threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, 2022. https://eprint.iacr.org/2022/898
54 Abraham I, Nayak K, Ren L, et al. Good-case latency of Byzantine broadcast: a complete categorization. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, 2021. 331–341
55 Xu H, Liu X, Zhang C, et al. Crackle: a fast sector-based BFT consensus with sublinear communication complexity. In: Proceedings of

the IEEE Conference on Computer Communications, 2024. 1–10
56 Kotla R, Alvisi L, Dahlin M, et al. Zyzzyva: speculative Byzantine fault tolerance. ACM Trans Comput Syst, 2009, 27: 1–39
57 Martin J P, Alvisi L. Fast Byzantine consensus. IEEE Trans Dependable Secure Comput, 2006, 3: 202–215
58 Behl J, Distler T, Kapitza R. Hybrids on steroids: SGX-based high performance BFT. In: Proceedings of the Twelfth European Conference on Computer Systems, 2017. 222–237
59 Kapitza R, Behl J, Cachin C, et al. CheapBFT: resource-efficient Byzantine fault tolerance. In: Proceedings of the 7th ACM European Conference on Computer Systems, 2012. 295–308
60 Pass R, Shi E. Thunderella: blockchains with optimistic instant confirmation. In: Proceedings of the 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2018. 3–33
61 Kelkar M, Deb S, Long S, et al. Themis: fast, strong order-fairness in Byzantine consensus. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, 2023. 475–489
62 Neiheiser R, Matos M, Rodrigues L. Kauri: scalable BFT consensus with pipelined tree-based dissemination and aggregation. In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, 2021. 35–48
63 Wang Q, Li R. A weak consensus algorithm and its application to high-performance blockchain. In: Proceedings of the IEEE Conference on Computer Communications, 2021. 1–10
64 Fu X, Wang H, Shi P. Votes-as-a-Proof (VaaP): permissioned blockchain consensus protocol made simple. IEEE Trans Parallel Distrib Syst, 2022, 33: 4964–4973
65 Correia M, Veríssimo P, Neves N F. The design of a cots real-time distributed security kernel. In: Proceedings of the European Dependable Computing Conference, 2002. 234–252
66 Chun B G, Maniatis P, Shenker S, et al. Attested append-only memory: making adversaries stick to their word. SIGOPS Oper Syst Rev, 2007, 41: 189–204
67 Veronese G S, Correia M, Bessani A N, et al. Efficient Byzantine fault-tolerance. IEEE Trans Comput, 2011, 62: 16–30
68 Veronese G S, Correia M, Bessani A N, et al. Spin one's wheels? Byzantine fault tolerance with a spinning primary. In: Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems, 2009. 135–144
69 Amir Y, Coan B, Kirsch J, et al. Prime: Byzantine replication under attack. IEEE Trans Dependable Secure Comput, 2010, 8: 564–577
70 Clement A, Wong E, Alvisi L, et al. Making Byzantine fault tolerant systems tolerate Byzantine faults. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, 2009
71 Niu J, Gai F, Jalalzai M M, et al. On the performance of pipelined hotstuff. In: Proceedings of the IEEE Conference on Computer Communications, 2021. 1–10
72 Danezis G, Kokoris-Kogias L, Sonnino A, et al. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In: Proceedings of the Seventeenth European Conference on Computer Systems, 2022. 34–50
73 Moghimi D, Sunar B, Eisenbarth T, et al. TPM-FAIL: TPM meets timing and lattice attacks. In: Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), 2020. 2057–2073
74 Zhao S, Zhang Q, Qin Y, et al. SecTEE: a software-based approach to secure enclave architecture using TEE. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019. 1723–1740
75 Chen L, Xu L, Shah N, et al. On security analysis of proof-of-elapsed-time (POET). In: Proceedings of the 19th International Symposium, 2017. 282–297
76 Milutinovic M, He W, Wu H, et al. Proof of luck: an efficient blockchain consensus protocol. In: Proceedings of the 1st Workshop on System Software for Trusted Execution, 2016. 1–6
77 Yandamuri S, Abraham I, Nayak K, et al. Brief announcement: communication-efficient BFT using small trusted hardware to tolerate minority corruption. In: Proceedings of the 35th International Symposium on Distributed Computing (DISC 2021), 2021. 62:1–62:4
78 Decouchant J, Kozhaya D, Rahli V, et al. Damysus: streamlined BFT consensus leveraging trusted components. In: Proceedings of the Seventeenth European Conference on Computer Systems, 2022. 1–16
79 Gupta S, Rahnama S, Hellings J, et al. ResilientDB: global scale resilient blockchain fabric. 2020. ArXiv:2002.00160
80 Fu X, Wang H M, Shi P C, et al. Teegraph: trusted execution environment and directed acyclic graph-based consensus algorithm for IoT blockchains. Sci China Inf Sci, 2022, 65: 139104
81 Gupta S, Hellings J, Sadoghi M. RCC: resilient concurrent consensus for high-throughput secure transaction processing. In: Proceedings of the 2021 IEEE 37th International Conference on Data Engineering (ICDE), 2021. 1392–1403
82 Stathakopoulou C, Pavlovic M, Vukolić M. State machine replication scalability made simple. In: Proceedings of the Seventeenth European Conference on Computer Systems, 2022. 17–33
83 Huawei. Blockchain service (BCS). 2024. https://www.huaweicloud.com/intl/en-us/product/bcs.html
84 Wang W, Deng S, Niu J, et al. Engraft: enclave-guarded raft on Byzantine faulty nodes. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022. 2841–2855
85 Zhang S, Chai J, Chen Z, et al. Byzantine consensus algorithm based on gossip protocol. Comput Sci, 2018, 45: 20–24
86 Zhang Q, Wang Z, Zhang Y. Research on trust collection consensus algorithm based on gossip protocol. Comput Sci, 2020, 47: 391–394
87 Sui X, Duan S, Zhang H. Marlin: two-phase BFT with linearity. In: Proceedings of the 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2022. 54–66
88 Kogias E K, Jovanovic P, Gailly N, et al. Enhancing bitcoin security and performance with strong consistency via collective signing. In: Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), 2016. 279–296
89 Kokoris-Kogias E. Robust and scalable consensus for sharded distributed ledgers. Cryptology ePrint Archive, 2019. https://eprint.iacr.org/2019/676
90 Li W, Feng C, Zhang L, et al. A scalable multi-layer PBFT consensus for blockchain. IEEE Trans Parallel Distrib Syst, 2020, 32: 1146–1160
91 Aublin P L, Guerraoui R, Knežević N, et al. The next 700 BFT protocols. ACM Trans Comput Syst, 2015, 32: 1–45
92 Kiayias A, Russell A, David B, et al. Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Proceedings of the Annual International Cryptology Conference, 2017. 357–388
93 Wang Q, Yu J, Peng Z, et al. Security analysis on DBFT protocol of NEO. In: Proceedings of the International Conference on Financial

Cryptography and Data Security, 2020. 20–31
94 Luu L, Narayanan V, Zheng C, et al. A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016. 17–30
95 Kokoris-Kogias E, Jovanovic P, Gasser L, et al. OmniLedger: a secure, scale-out, decentralized ledger via sharding. In: Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), 2018. 583–598
96 Miller A, Juels A, Shi E, et al. Permacoin: repurposing bitcoin work for data preservation. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy, 2014. 475–490
97 Park S, Pietrzak K, Alwen J, et al. Spacecoin: a cryptocurrency based on proofs of space. Cryptology ePrint Archive, 2015. https://www.allcryptowhitepapers.com/wp-content/uploads/2018/05/Spacecoin.pdf
98 Xiang F, Huaimin W, Peichang S. Proof of previous transactions (PoPT): an efficient approach to consensus for JCLedger. IEEE Trans Syst Man Cybern Syst, 2021, 51: 2415–2424
99 Micali S, Rabin M, Vadhan S. Verifiable random functions. In: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999. 120–130
100 Team T O. Ontology launches VBFT, a next-generation consensus mechanism, becoming one of the first VRF-based public chains. 2018. https://medium.com/ontologynetwork/ontology-launches-vbft-a-next-generation-consensus-mechanism-becoming-one-of-the-first-vrf-based-91f782308db4
101 Gilad Y, Hemo R, Micali S, et al. Algorand: scaling Byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, 2017. 51–68
102 Feng L, Ding Y, Tan Y, et al. Trusted-committee-based secure and scalable BFT consensus for consortium blockchain. In: Proceedings of the 2022 18th International Conference on Mobility, Sensing and Networking (MSN), 2022. 363–370
103 Distler T, Cachin C, Kapitza R. Resource-efficient Byzantine fault tolerance. IEEE Trans Comput, 2015, 65: 2807–2819
104 Wang R, Zhang L, Xu Q, et al. Byzantine fault-tolerant consensus algorithm that can be applied to the alliance chain. Appl Res Comput, 2020, 37: 3382–3386
105 Abd-El-Malek M, Ganger G R, Goodson G R, et al. Fault-scalable Byzantine fault-tolerant services. SIGOPS Oper Syst Rev, 2005, 39: 59–74
106 Cowling J, Myers D, Liskov B, et al. Hq replication: a hybrid quorum protocol for Byzantine fault tolerance. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, 2006. 177–190
107 Stathakopoulou C, David T, Pavlovic M, et al. Mir-BFT: high-throughput robust BFT for decentralized networks. 2019. ArXiv:1906.05552
108 Voron G, Gramoli V. Dispel: Byzantine SMR with distributed pipelining. 2019. ArXiv:1912.10367
109 Arun B, Ravindran B. Scalable Byzantine fault tolerance via partial decentralization. 2022. ArXiv:2202.13408
110 Bai Y, Zhi Y, Li H, et al. On parallel mechanism of consortium blockchain: take POV as an example. In: Proceedings of the 2021 3rd International Conference on Blockchain Technology, 2021. 147–154
111 Wang Z, Li H, Wang H, et al. A data lightweight scheme for parallel proof of vote consensus. In: Proceedings of the 2021 IEEE International Conference on Big Data, 2021. 3656–3662
112 Xiao Z, Li H, Wang H, et al. Optimizing parallel proof of vote consensus based on mimic security in consortium blockchains. In: Proceedings of the 2022 IEEE International Conference on Big Data, 2022. 3215–3224
113 Li K, Li H, Hou H, et al. Proof of vote: a high-performance consensus protocol based on vote mechanism & consortium blockchain. In: Proceedings of the 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2017. 466–473
114 Li K, Li H, Wang H, et al. PoV: an efficient voting-based consensus algorithm for consortium blockchains. Front Blockchain, 2020, 3: 11
115 Arun B, Peluso S, Ravindran B. ezBFT: decentralizing Byzantine fault-tolerant state machine replication. In: Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019. 565–577
116 Neu J, Tas E N, Tse D. Ebb-and-flow protocols: a resolution of the availability-finality dilemma. In: Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), 2021. 446–465