Proceedings of the 3rd International Conference on Mathematical Physics and Computational Simulation
DOI: 10.54254/2753-8818/2025.25474

An Improved PBFT Consensus Algorithm Based on the Raft Voting Mechanism and DAG Ledger Structure

Wei Zhang

School of Computer Science and Artificial Intelligence, North China University of Technology, Beijing, China
2283170387@qq.com

Abstract. To address the issues of high communication overhead, low throughput, and arbitrary primary node selection in the traditional PBFT consensus algorithm, this paper proposes an improved PBFT consensus algorithm based on the Raft voting mechanism and the DAG ledger structure. By introducing a two-layer architecture composed of proxy nodes and candidate nodes, the system nodes are reorganized. Within each proxy domain, a proxy primary node is elected using the Raft voting mechanism, thereby enhancing the stability and efficiency of primary node transitions. During the consensus process, leveraging the DAG ledger structure enables parallel block generation, which is divided into two phases: data block consensus and address block consensus. Digital signatures and hash commitment mechanisms are introduced in each phase to ensure message integrity and the verifiability of consensus data. Simulation results demonstrate that the improved algorithm achieves lower transaction latency and higher throughput compared to the original PBFT algorithm.

Keywords: Blockchain, Byzantine Fault Tolerance, Raft, DAG, Parallel Block Generation

1. Introduction

In recent years, with the continuous development of blockchain technology, its characteristics—such as decentralization, traceability, and immutability—have demonstrated broad application prospects in various fields, including financial transactions, the Internet of Things, and digital identity authentication. However, the underlying consensus mechanism of a blockchain system directly determines its performance, fault tolerance, and scalability. Therefore, designing efficient, secure, and reliable consensus algorithms has become one of the core issues in current research. The Practical Byzantine Fault Tolerance (PBFT) algorithm, as a classical Byzantine fault-tolerant consensus protocol, has been widely adopted due to its advantages of not relying on proof of work, low computational overhead, and fast transaction confirmation.

However, PBFT still faces several limitations in practical applications, primarily including: 1)As the number of consensus nodes increases, the communication complexity grows to $O(n^2)$, severely restricting system scalability; 2)The primary node becomes a performance bottleneck. When it fails or acts maliciously, the system must perform a view change, resulting in significant delays; 3)The algorithm lacks sufficient security enhancement mechanisms, making it vulnerable to message forgery or collusion attacks by malicious nodes [1]. To address these issues, researchers have

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2753-8818/2025.25474

proposed various optimization schemes. For example, Yin et al. introduced the HotStuff protocol, which compresses the original three-phase consensus process into a single pipelined channel through a chained structure. This design reduces message complexity and enables rapid view changes, significantly improving the efficiency of primary node transitions [2]. Bessani et al. developed the BFT-SMaRt framework, which incorporates asynchronous execution and a modular architecture to mitigate performance bottlenecks caused by primary node overload [3]. To further reduce communication overhead, Ren et al. introduced an aggregate signature mechanism that merges and compresses responses from multiple nodes, significantly reducing network bandwidth consumption [4]. Additionally, Gueta et al. proposed the SBFT protocol, which integrates threshold signatures with pipelined consensus to enhance processing capacity in large-scale node deployments [5]. Although the above optimization approaches have achieved notable progress, there remains room for improvement in balancing consistency and performance.

After a comparative analysis of existing improved PBFT algorithms, this paper proposes a two-phase proxy PBFT consensus algorithm—RDPBFT—that integrates the Raft voting mechanism, digital signatures, and hash commitment techniques. While preserving the security properties of PBFT, this algorithm enhances system scalability through proxy-based structural partitioning and strengthens the non-repudiation and tamper-resistance of consensus messages via cryptographic mechanisms, thereby significantly improving system performance while ensuring consistency.

2. Related background

2.1. Introduction to the PBFT algorithm

The PBFT algorithm is designed to solve the Byzantine Generals Problem and ensures the consistency and availability of a system with $n = 3f + 1$ nodes in the presence of up to f Byzantine nodes (i.e., malicious or faulty nodes). PBFT is based on the state machine replication model, in which the primary node and replica nodes collaboratively exchange multiple rounds of messages to filter out interference from malicious nodes and ensure that all non-malicious nodes execute client requests in the same order, thereby maintaining global state consistency [6,7]. The core components of the PBFT algorithm include the consistency protocol, the view change protocol, and the checkpoint protocol.

2.1.1. Consistency protocol

The figure below illustrates the PBFT consistency protocol, also known as the three-phase consensus process. It involves a client node C and four consensus nodes, where node P is the primary node selected through a rotation and election mechanism, and nodes 1, 2, and 3 are replica nodes (also referred to as backup nodes).

DOI: 10.54254/2753-8818/2025.25474

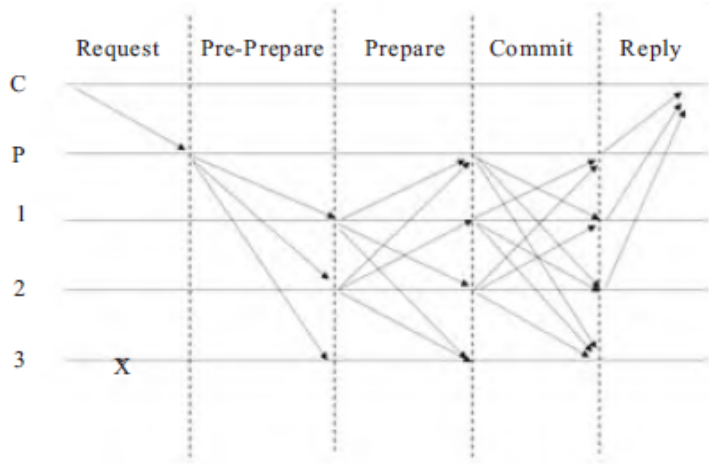


Figure 1. PBFT algorithm consistency protocol

2.1.2. View change protocol

The view change protocol in PBFT is used to replace the primary node in the event of failure or delay, thereby maintaining system availability. If a replica node does not receive periodic heartbeat or pre-prepare messages from the current primary node, it triggers an election timeout mechanism, switches its state to candidate, increments the current view number by one, and broadcasts a VIEW-CHANGE message to all nodes. This message contains digests of the prepare and commit logs it has received. Once a node receives at least $2f+1$ VIEW-CHANGE messages, it becomes the new primary node if it satisfies the rotation policy. It then constructs a NEW-VIEW message that aggregates all VIEW-CHANGE messages and includes any pre-prepare or prepare messages that need to be replayed in the new view. After verifying the validity of the NEW-VIEW message, other nodes switch to the new view and resume the consistency protocol, thereby ensuring the consensus process can continue even in the presence of up to f Byzantine nodes [8].

2.1.3. Checkpoint protocol

The checkpoint protocol in PBFT is designed to limit the length of logs and accelerate node state synchronization. After a node finishes executing a request whose sequence number is a multiple of a predefined interval, it computes a digest of the current system state and packages it together with the view number and corresponding sequence number into a checkpoint message, which is then broadcast to all other nodes. Upon verifying the legitimacy of the message source, receiving nodes record the checkpoint in their local logs. If a node collects checkpoint messages from at least $2f+1$ distinct nodes containing the same view number, sequence number, and state digest, the checkpoint is considered stable. Once a checkpoint becomes stable, all request logs with sequence numbers less than or equal to that checkpoint can be pruned, retaining only the corresponding state digest. This significantly reduces storage and communication overhead. For nodes newly joining the network or recovering from failure, it is sufficient to synchronize directly from the latest stable checkpoint's state snapshot, without replaying the full transaction history, thus greatly improving restart and state recovery efficiency.

DOI: 10.54254/2753-8818/2025.25474

2.2. Introduction to the raft voting mechanism

The Raft voting mechanism employs a term-based democratic election to ensure the uniqueness of a strong leader. When a follower does not receive a heartbeat from the leader within the election timeout period, it increments its term, transitions to a candidate state, and initiates a vote request [9]. Each node can cast only one vote per term, following a first-come, first-served principle [10]. A candidate must obtain votes from a majority of the cluster to become the leader. This design effectively prevents split-brain scenarios through a quorum mechanism. If the candidate fails to receive a majority before the election timeout, a randomized timeout triggers a retry, significantly reducing the probability of vote failure caused by competition among multiple candidates [11]. Once elected, the leader maintains authority by continuously sending heartbeats (with intervals shorter than the minimum election timeout). In the event of a network partition, the minority partition cannot achieve a majority vote and will repeatedly fail to elect a leader, thereby ensuring data consistency under partition scenarios [12].

2.3. Research on consensus algorithms based on DAG ledger structure

In traditional blockchain systems, transaction data is organized in a linear chain structure where each block can reference only a single predecessor. This requires the consensus process to strictly confirm transactions in chronological order, resulting in bottlenecks in throughput and concurrency performance. To address these issues, researchers have proposed DAG-based ledgers that enable parallel linking of transactions or blocks, achieving high throughput and low latency. In 2016, the Popov team developed the IOTA Tangle, which constructs a blockless DAG architecture through referencing relationships between transaction units, achieving a throughput of thousands of TPS, but relies on a bounded delay assumption to prevent double-spending risks [13]. In 2017, LeMahieu designed the Block-lattice structure in Nano, combining a DPoS voting mechanism to improve efficiency; however, the use of representative nodes causes partial centralization [14]. In 2018, Long Fan and colleagues proposed the Conflux Tree-Graph consensus, which guarantees global consistency in multi-chain parallel scenarios through main chain ordering, reaching a throughput of 6,400 TPS [15]. In 2020, Gao Zhengfeng's team pointed out in systematic research that topological sorting delay is a common bottleneck in DAG consensus [16]. In 2023, IOTA 2.0 introduced a fast probabilistic consensus to strengthen security, capable of resisting up to 89% malicious attacks [17]. However, Mayank Raikwar and other scholars revealed in 2024 that DAG protocols still face challenges in cross-epoch data verifiability and require dynamic weight mechanisms to defend against large-weight attacks [17]. Currently, academia is focusing on indexing learning techniques (such as the Lever mechanism) to optimize data query efficiency, aiming to balance decentralization and high throughput demands [17].

3. RDPBFT consensus algorithm scheme

The RDPBFT consensus scheme consists of three parts: consensus system initialization, the Raft voting mechanism, and the DPBFT consensus algorithm.

3.1. Consensus system initialization

All nodes are first divided into proxy domains and candidate domains, where R1 represents the proxy domain and R2 represents the candidate domain.

DOI: 10.54254/2753-8818/2025.25474

Define within R1: $R_1 \geq 3f+1$, where R1 is the total number of nodes in the proxy domain, and f represents the number of faulty nodes in the proxy domain.

Define within R2: $R_2 \geq 2f+1$, where R2 is the total number of nodes in the candidate domain, and f represents the number of faulty nodes in the candidate domain. Nodes in R2 do not participate in consensus.

3.2. Raft voting mechanism

In this scheme, the Raft voting mechanism is employed to dynamically add or remove candidate domain nodes when selecting the proxy primary node. The proxy primary node is elected within the candidate domain using Raft's leader election method.

3.2.1. Proxy primary node selection

R2 maintains a persistent connection with the proxy primary node within R1 through a heartbeat mechanism. When the proxy primary node in R1 switches, a heartbeat timeout causes the connection with R2 to be interrupted. At this time, R2 proactively synchronizes its pre-maintained candidate primary node information to R1, thereby avoiding the cumbersome process of re-initiating the primary node election in traditional algorithms. This effectively reduces the latency caused by primary node switching and improves the system's consensus efficiency. The primary node replacement process is illustrated in the figure below.

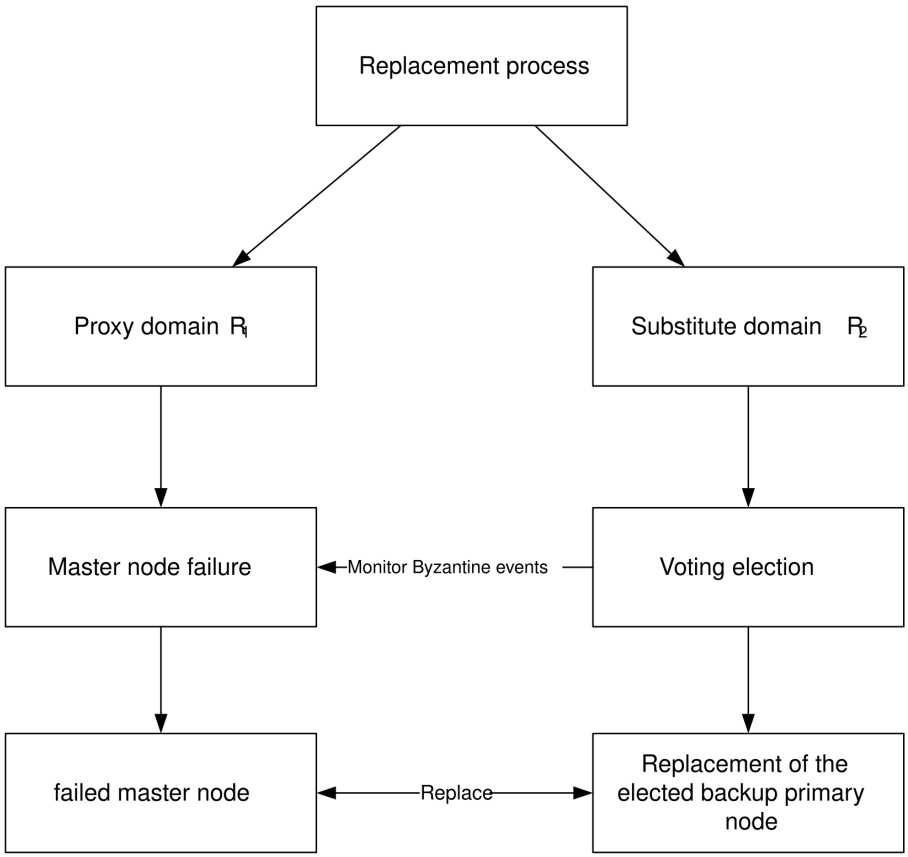


Figure 2.

DOI: 10.54254/2753-8818/2025.25474

3.2.2. Election method

The election method refers to the Raft algorithm. Nodes in the cluster maintain the leader's authority through heartbeats. If a follower fails to receive a heartbeat within the timeout period, it initiates a new candidate leader election. The candidate sends vote requests to other followers and increments its term number. The node that obtains the majority of votes becomes the new leader. The primary node election process is illustrated in the figure below.

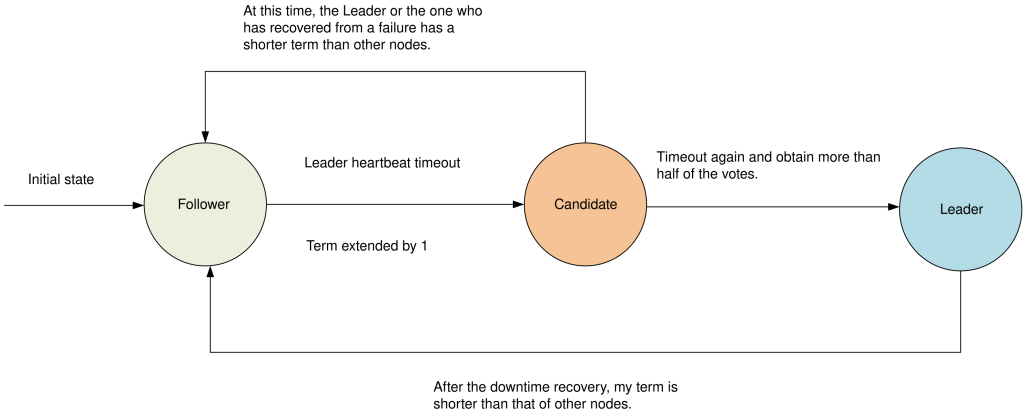


Figure 3. Primary node election process

3.3. DPBFT consensus algorithm

3.3.1. Dag-based ledger structure

Unlike linear blockchain structures, DAG ledger technology does not have a global chain order, allowing multiple transactions to be processed in parallel. Based on this idea, this scheme proposes the DPBFT consensus algorithm, which supports multi-node parallel block generation, thereby improving consensus efficiency. The ledger structure adopts a dual-module design, consisting of data blocks and address blocks. The ledger structure is illustrated in the figure below.

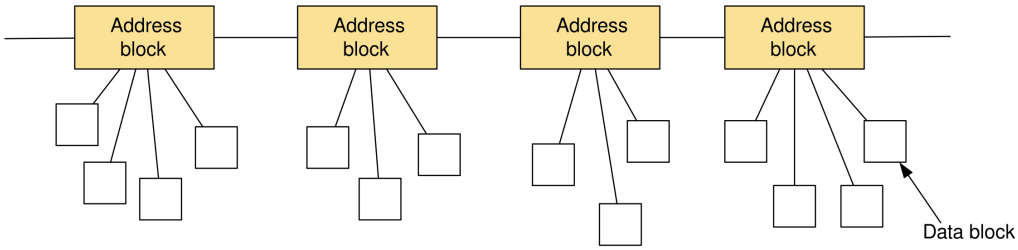


Figure 4. DPBFT algorithm ledger structure

3.3.2. Consensus process of the DPBFT algorithm

The consensus process of the DPBFT algorithm is divided into five phases: pre-propose, propose, pre-prepare, prepare, and commit. Throughout these five phases, digital signatures and hash commitments are employed. On one hand, digital signatures ensure message authenticity and integrity; on the other hand, hash commitments conceal the data block digest during the pre-propose phase and are only revealed in the propose phase through verification commitments actively

DOI: 10.54254/2753-8818/2025.25474

provided by nodes. This approach protects data privacy while ensuring data integrity. The consensus process of the DPBFT algorithm is illustrated in the figure below.

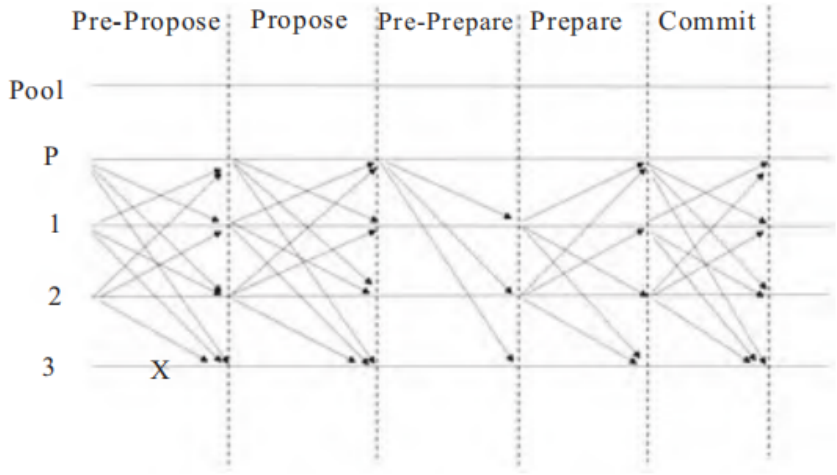


Figure 5. Consensus process of the DPBFT algorithm

1) Pre-propose Phase: Each node i packages its pending transaction set $_i$ into a data block DB_i and computes its digest.

$$d_i = H(DB_i),$$

Here, $\mathrm{H}(\bullet)$ denotes a secure hash function. Each node then randomly selects a commitment random number $\mathrm{r_i} \in \mathrm{Z_q}$ to generate a hash commitment

$$C_i = \mathsf{Com}(d_i; r_i) = g^{d_i} h^{r_i} \mathrm{mod} p,$$

to conceal the data block digest. Next, node i constructs the pre-propose message

$$m_i^{\mathrm{PP}} = \langle \mathrm{PRE\text{-}PROPOSE}, r, v, C_i, i \rangle$$

and computes a digital signature using its private key $\mathrm{sk_i}$

$$\sigma_i^{\mathrm{PP}} = \mathsf{Sign}_{\backslash \mathrm{sk}_i}(r \parallel v \parallel C_i \parallel i).$$

Finally, node i broadcasts the signed message pair

$$M_i^{\mathrm{PP}} = \langle \mathrm{PRE\text{-}PROPOSE}, r, v, C_i, i, \sigma_i^{\mathrm{PP}} \rangle.$$

After receiving it, other nodes M_i^{PP} verify it by

$$\mathrm{Verify}_{\backslash \mathrm{pk}_i}\left(\sigma_i^{\mathrm{PP}}, r \parallel v \parallel C_i \parallel i\right) = \mathrm{true}$$

DOI: 10.54254/2753-8818/2025.25474

verifying the validity of the signature. If the verification succeeds, the pair (C_i, i) is stored in the local log. When a node accumulates at least $2f+1$ pre-propose messages with valid signatures and identical (r, C_i), the corresponding data block is marked as "Proposed."

2) Propose Phase: Nodes verify the signatures and consistency of commitments for all previously received pre-propose messages $\left\{M_j^{\mathrm{PP}}\right\}$ one by one. If node j locally holds the opening commitment random number r_j and the original data block DB_j, it then verifies

$$\mathsf{Com}\left(H(\mathsf{DB}_j); r_j\right) \stackrel{?}{=} C_j, \;\; H\left(\mathsf{DB}_j\right) \stackrel{?}{=} d_j,$$

and verifies the digital signature

$$\mathrm{Verify}_{\backslash \mathrm{pk}_j}\left(\sigma_j^{\mathrm{PP}}, r \parallel v \parallel C_j \parallel j\right) = \mathrm{true}.$$

For each verified node i, the node constructs a propose message

$$m_i^{\mathrm{P}} = \left\langle \mathrm{PROPOSE}, r, v, C_i, i \right\rangle,$$

and computes its signature.

$$\sigma_i^{\mathrm{P}} = \mathsf{Sign}_{\backslash \mathrm{sk}_i}\left(r \parallel v \parallel C_i \parallel i\right).$$

The node then broadcasts the signed message pair

$$M_i^{\mathrm{P}} = \left\langle \mathrm{PROPOSE}, r, v, C_i, i, \sigma_i^{\mathrm{P}} \right\rangle,$$

The receiver verifies the message

$$\mathrm{Verify}_{\backslash \mathrm{pk}_i}\left(\sigma_i^{\mathrm{P}}, r \parallel v \parallel C_i \parallel i\right) = \mathrm{true}$$

and then records (C_i, i). If the node accumulates at least $2f+1$ propose messages with identical (r, C_i), the corresponding data block is marked as “Proposed,” preparing for the next phase.

3) Pre-prepare Phase: The proxy primary node L, elected through the voting mechanism in Section 3.2, collects all commitments $\{C_{i_1}, C_{i_2}, \ldots, C_{i_m}\}(\text{where } m \geq 2f+1)$ submitted by nodes marked as “Proposed” in the current round. The proxy primary node then sequentially requests these nodes to provide their data blocks DB_{i_k} and commitment random numbers r_{i_k}, and verifies

$$\mathsf{Com}\left(H(\mathsf{DB}_{i_k}); r_{i_k}\right) \stackrel{?}{=} C_{i_k}, \;\; H\left(\mathsf{DB}_{i_k}\right) \stackrel{?}{=} d_{i_k}.$$

After confirming their correctness, the proxy primary node constructs a Merkle tree using all d_{i_k} as leaf nodes and computes the Merkle root

$$\mathrm{root}_M = \mathrm{MerkleRoot}\left(\{d_{i_k}\}_{k=1}^{m}\right),$$

DOI: 10.54254/2753-8818/2025.25474

and defines the content of the address block as follows

$$\text{IndexBlock} = \{\text{root}_M, r, v, L\}, d_v = H\left(\text{IndexBlock}\right).$$

The proxy primary node then constructs the pre-prepare message

$$m_L^{\text{PP}^*} = \left\langle \text{PRE-PREPARE}, r, v, d_v, L \right\rangle$$

and signs

$$\sigma_L^{\text{PP}^*} = \text{Sign}_{\backslash \text{sk}_L}\left(r \parallel v \parallel d_v \parallel L\right),$$

and finally broadcasts it.

$$M_L^{\text{PP}^*} = \left\langle \text{PRE} - \text{PREPARE}, r, v, d_v, L, \sigma_L^{\text{PP}^*} \right\rangle.$$

Upon receiving the message, each replica node $\text{M}_\text{L}^{\text{PP}^*}$ first verifies the signature

$$\text{Verify}_{\backslash \text{pk}_L}\left(\sigma_L^{\text{PP}^*}, r \parallel v \parallel d_v \parallel L\right) = \text{true},$$

Then, using the known $\{\text{C}_{\text{i}_\text{k}}\}$ along with the revealed $\{\text{d}_{\text{i}_\text{k}}, \text{r}_{\text{i}_\text{k}}\}$, it recomputes the Merkle root to verify

$$\text{root}_M \overset{?}{=} \text{MerkleRoot}\left(\{d_{i_k}\}\right).$$

If the verification passes, the replica node writes $(\text{d}_\text{v}, \text{root}_\text{M})$ into the local log and proceeds to the prepare phase.

4) Prepare Phase: Each replica node j constructs a prepare message based on the verified pre-prepare message $\text{M}_\text{L}^{\text{PP}^*}$

$$m_j^{\text{PR}} = \left\langle \text{PREPARE}, r, v, d_v, j \right\rangle$$

and signs

$$\sigma_j^{\text{PR}} = \text{Sign}_{\backslash \text{sk}_j}\left(r \parallel v \parallel d_v \parallel j\right).$$

and finally broadcasts it.

$$M_j^{\text{PR}} = \left\langle \text{PREPARE}, r, v, d_v, j, \sigma_j^{\text{PR}} \right\rangle.$$

Other nodes, upon receiving $\text{M}_\text{j}^{\text{PR}}$, verify the digital signature.

DOI: 10.54254/2753-8818/2025.25474

$$\mathsf{Verify}_{\backslash \mathrm{pk}_i}\left(\sigma_j^{\mathrm{PR}}, r \parallel v \parallel d_v \parallel j\right) = \mathsf{true},$$

They then count the valid prepare messages with the same $(\mathrm{r}, \mathrm{d_v})$; once a node accumulates at least $2\mathrm{f}+1$ prepare messages with valid signatures and identical $(\mathrm{r}, \mathrm{d_v})$, it confirms that the address block has been approved by the majority of nodes and proceeds to the commit phase.

5) Commit Phase: Each node k that meets the prepare threshold constructs a commit message

$$m_k^{\mathrm{C}} = \langle \mathrm{COMMIT}, r, v, d_v, k \rangle$$

and signs

$$\sigma_k^{\mathrm{C}} = \mathsf{Sign}_{\backslash \mathrm{sk}_k}(r \parallel v \parallel d_v \parallel k).$$

Node k broadcasts the message

$$M_k^{\mathrm{C}} = \langle \mathrm{COMMIT}, r, v, d_v, k, \sigma_k^{\mathrm{C}} \rangle.$$

Other nodes verify the signature upon receipt.

$$\mathsf{Verify}_{\backslash \mathrm{pk}_k}\left(\sigma_k^{\mathrm{C}}, r \parallel v \parallel d_v \parallel k\right) = \mathsf{true},$$

They then count the commit messages with the same (r, d_v) ; once at least $2\mathrm{f}+1$ commit messages with valid signatures and identical (r, d_v) are received, it is concluded that the address block has reached final consensus in the network. The address block is then written to the ledger, triggering node state updates and transaction execution.

4. RDPBFT consensus algorithm process

Assuming that R consensus nodes are deployed in the network, with node identifiers $\{0,1,\ldots,\mathrm{R}-1\}$. The algorithm process is illustrated in the figure below:

DOI: 10.54254/2753-8818/2025.25474

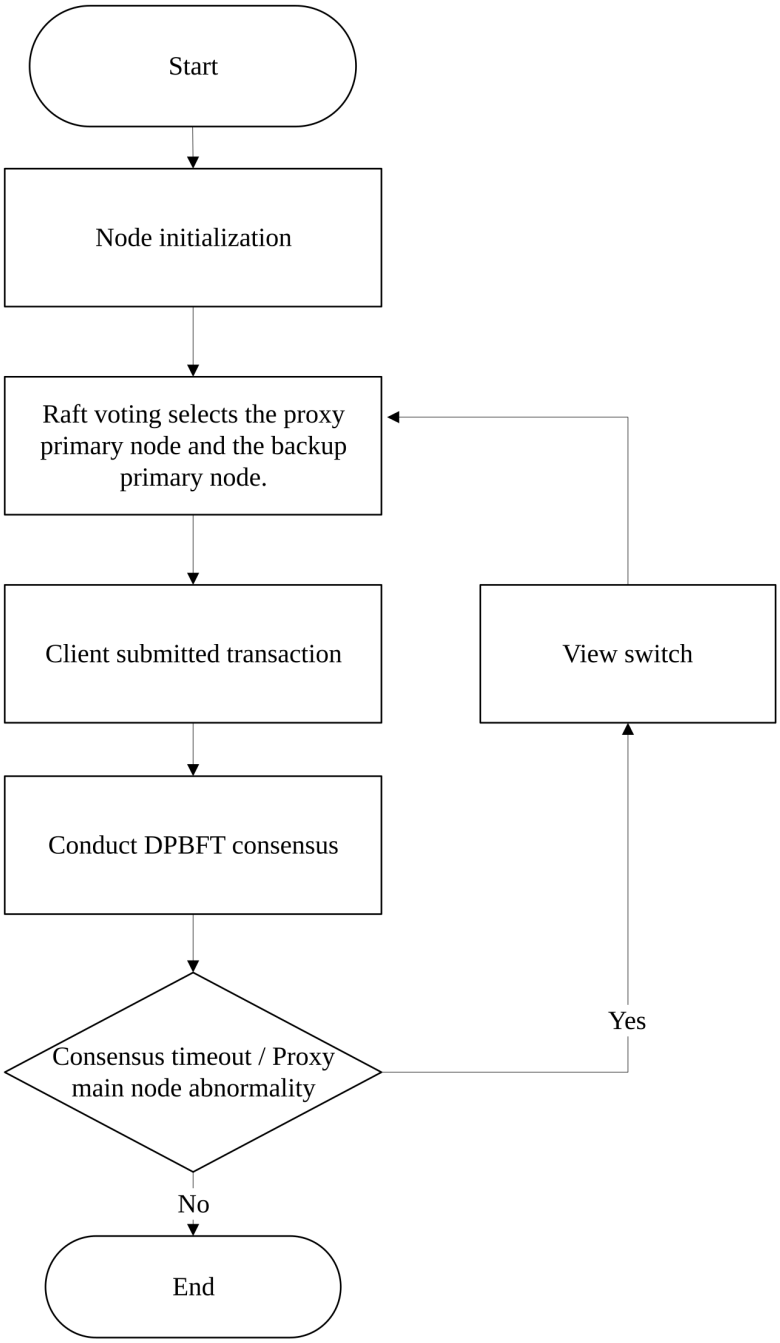


Figure 6. RDPBFT consensus algorithm process

1) Node Initialization: All R nodes in the system are assigned roles based on their functions and divided into the proxy domain R1 and the candidate domain R2. Nodes in the proxy domain participate in the consensus execution of the current round, while nodes in the candidate domain serve as backup resources for consensus nodes to enhance fault tolerance.

2) Within the proxy domain, the Raft voting mechanism is used to elect the proxy primary node for the current round; simultaneously, a candidate primary node is elected within the candidate domain to enable rapid switching and replacement in case of primary node failure.

3) Clients submit transactions.

DOI: 10.54254/2753-8818/2025.25474

4) Consensus nodes within the proxy domain enter the DPBFT consensus process and sequentially execute the following five phases: pre-propose $M_i^{\mathrm{PP}} = \langle \mathrm{PRE\text{-}PROPOSE}, r, v, C_i, i, \sigma_i^{\mathrm{PP}} \rangle$, propose $M_i^{\mathrm{P}} = \langle \mathrm{PROPOSE}, r, v, C_i, i, \sigma_i^{\mathrm{P}} \rangle$, pre-prepare $M_L^{\mathrm{PP}^*} = \left\langle \mathrm{PRE-PREPARE}, r, v, d_v, L, \sigma_L^{\mathrm{PP}^*} \right\rangle$, prepare $M_j^{\mathrm{PR}} = \left\langle \mathrm{PREPARE}, r, v, d_v, j, \sigma_j^{\mathrm{PR}} \right\rangle$, and commit $M_k^{\mathrm{C}} = \langle \mathrm{COMMIT}, r, v, d_v, k, \sigma_k^{\mathrm{C}} \rangle$, ultimately reaching consensus.

5) If consensus is not reached within the specified time window, or if the primary node is detected to be malfunctioning or acting maliciously, the view change protocol is immediately triggered. The system will automatically replace the failed primary node with a pre-selected candidate primary node, thereby ensuring the continuity and robustness of the consensus process.

5. Simulation experiments and analysis

The experiments were conducted on a computer configured with an Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz processor, 8GB of memory, a 256GB solid-state drive, and Windows 10 operating system. The PBFT algorithm was implemented in the Go programming language, and the RDPBFT algorithm was developed based on it. Multiple nodes were locally launched to simulate the consensus processes of the PBFT and RDPBFT algorithms, recording the algorithms' throughput and consensus latency for comparison.

5.1. Throughput test

Transaction throughput is an important metric for measuring the performance of consensus algorithms. In blockchain systems, it refers to the number of transactions reaching consensus per second, expressed as TPS (transactions per second).

$$TPS = \frac{Transactions}{\Delta t}$$

Here, TPS represents throughput, Transactions denotes the number of transactions, and Δt indicates the time used. Throughput tests were conducted on both algorithms by sending different transaction volumes within the same block interval for 10 trials, with the average taken as the result. As shown in the figure below, throughput increases initially and then declines later; however, the TPS of the RDPBFT algorithm consistently remains higher than that of the original algorithm.

DOI: 10.54254/2753-8818/2025.25474

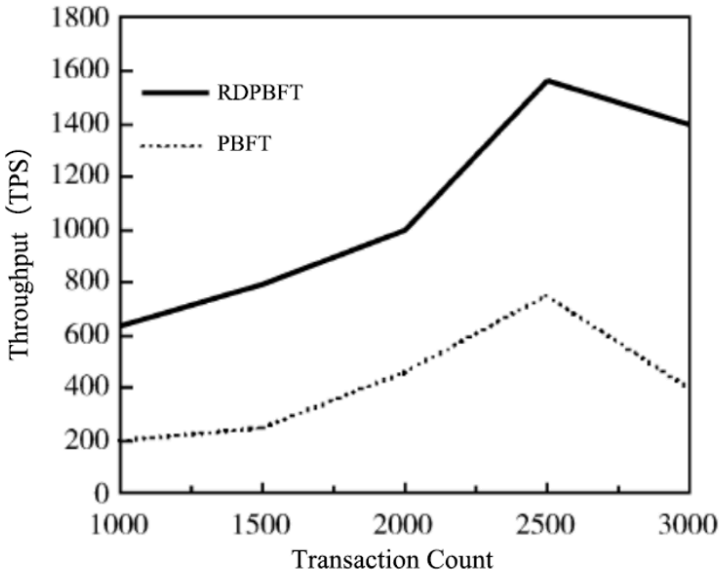


Figure 7. Throughput comparison test

5.2. Consensus latency test

Consensus latency refers to the time from when the client sends a request to a node until it receives f +1 identical response messages, denoted by Δt.

$$T = T_{\text{complete}} - T_{\text{submip}}$$

Here, $T_{complete}$ denotes the time when the transaction is finally completed, and T_{submip} denotes the time when the transaction is initiated. A comparison test between the two algorithms was conducted with 7 consensus nodes, where the system sent 200, 400, 600, 800, and 1000 transactions respectively. The experimental results are shown in the figure below, with the average values from multiple trials recorded. As the transaction volume increases, the latency curve of the PBFT algorithm rises rapidly, while that of the RDPBFT algorithm grows more slowly. Ultimately, the latency decreases by approximately 30%, indicating that the RDPBFT algorithm has lower latency.

DOI: 10.54254/2753-8818/2025.25474

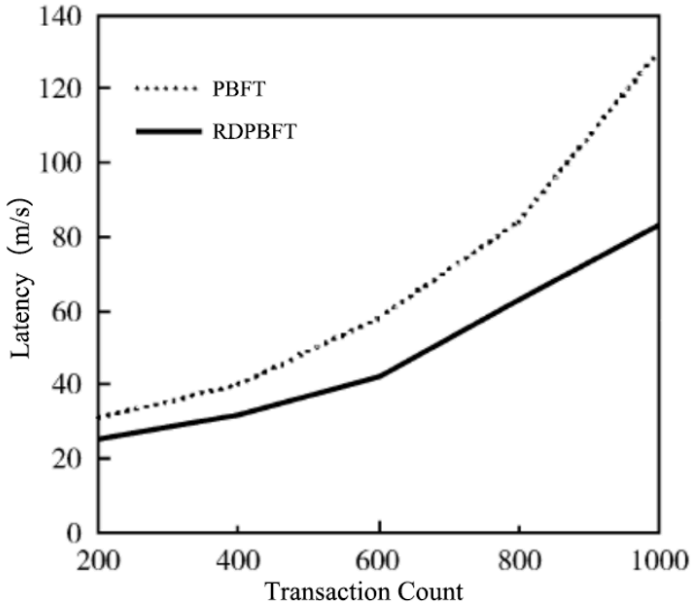


Figure 8. Consensus latency comparison experiment

6. Conclusion

This paper addresses the limitations of Byzantine fault-tolerant consensus algorithms regarding high communication overhead, low throughput, and uncertainty in primary node selection, and proposes the RDPBFT algorithm, which integrates the Raft election mechanism with a DAG ledger structure. The algorithm introduces the Raft mechanism to achieve orderly election and dynamic switching of the primary node, while leveraging the DAG structure to enable parallel block production. In terms of security design, it combines digital signatures and hash commitment techniques to enhance the verifiability and tamper resistance of consensus messages. Experimental results demonstrate that the algorithm significantly reduces network latency and improves throughput performance while maintaining security. Future research may further explore the algorithm's adaptability and scalability in cross-domain chain collaboration and resource-constrained environments.

References

[1] Ren, L., Zhang, S., & Wang, Y. (2021). A survey on Byzantine consensus and fault tolerance in blockchain. ACM Computing Surveys, 54(6), 1–36. https: //doi.org/10.1145/3457606

[2] Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., & Abraham, I. (2019). HotStuff: BFT consensus with linearity and responsiveness. In Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC) (pp. 347–356). https: //doi.org/10.1145/3293611.3331591

[3] Bessani, A., Sousa, J., & Alchieri, E. A. P. (2014). State machine replication for the masses with BFT-SMaRt. In 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (pp. 355–362). https: //doi.org/10.1109/DSN.2014.43

[4] Ren, L., Zhang, X., & Xie, Y. (2021). A communication-efficient PBFT consensus algorithm based on aggregate signature. Journal of Systems Architecture, 117, 102128. https: //doi.org/10.1016/j.sysarc.2021.102128

[5] Gueta, G. G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M. K., & Tamir, O. (2019). SBFT: A scalable and decentralized trust infrastructure. In 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (pp. 568–580). https: //doi.org/10.1109/DSN.2019.00062

DOI: 10.54254/2753-8818/2025.25474

[6] Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals problem. ACM Transactions on Programming Languages and Systems, 4(3), 382–401. https: //doi.org/10.1145/357172.357176
[7] Castro, M., & Liskov, B. (2002). Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems, 20(4), 398–461. https: //doi.org/10.1145/571637.571640
[8] Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys, 22(4), 299–319. https: //doi.org/10.1145/98163.98167
[9] Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC'14) (pp. 305–319). USENIX Association.
[10] Howard, H., Malkhi, D., Naor, M., et al. (2015). Raft refloated: Do we have consensus? ACM SIGOPS Operating Systems Review, 49(1), 30–36. https: //doi.org/10.1145/2723872.2723882
[11] Alvaro, P., Hellerstein, J. M., & Marczak, W. R. (2020). Leadership in protocol Raft. Distributed Computing, 33(1), 1–19. https: //doi.org/10.1007/s00446-019-00355-1
[12] Butt, A. R., Ahmed, A., Shah, M. A., et al. (2023). Performance characterization of Raft consensus. IEEE Transactions on Parallel and Distributed Systems, 34(2), 408–422. https: //doi.org/10.1109/TPDS.2022.3196875
[13] Popov, S. (2016). The Tangle [White paper]. IOTA. Retrieved June 12, 2025, from https: //iota.org/Tangle_illustration.pdf
[14] LeMahieu, C. (2017). Nano: A feeless distributed cryptocurrency network [White paper]. Retrieved June 12, 2025, from https: //nano.org/en/whitepaper
[15] Li, Z., Li, C., Zhou, P., et al. (2019). Conflux: A scalable and decentralized blockchain. In Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP) (pp. 538–555). ACM.
[16] Gao, Z., Jiang, C., & Wang, Y. (2020). Research on distributed ledger consensus mechanism based on DAG. Journal of Software, 31(10), 2974–2994. https: //doi.org/10.13328/j.cnki.jos.006134
[17] Raikwar, M., Ghosal, D., Conti, M., et al. (2024). SoK: DAG-based consensus protocols [Preprint]. arXiv. https: //arxiv.org/abs/2411.10026