BiniBFT v2 - December 3, 2024

BiniBFT

- A Secure, Scalable BFT for Hyperledger Fabric

Anasuya Threse Innocent .A[#], Raju Owk*

[#]*Founder & Director, BiniWorld Innovations Private Limited, Manakkarai West, Villukuri Post, Kanyakumari District, India, www.biniworld.com*

Mentor, Linux Foundation Hyperledger Collaborative Learning Program, 2024, binibft@biniworld.com

**Mentee, Linux Foundation Hyperledger Collaborative Learning Program - An Optimized BFT on Fabric, 2024, rajuowk37@gmail.com*

Abstract

Blockchain technology has a wide range of day-to-day applications ranging from electronic voting to supply chain management to financial transactions to healthcare management, energy management etc in a distributed and a decentralized environment. The enterprise blockchain provides transparency and immutability in a premissioned setting. The consensus protocols play a crucial role in establishing trustworthiness within these systems, facilitating secure and reliable operations across various sectors. The seamless integration of Byzantine Fault Tolerance (BFT) further fortifies the trustworthiness of enterprise blockchain. To achieve high security and scalability this paper presents the BiniBFT consensus for Hyperledger Fabric. This paper highlights the pivotal role of blockchain and BiniBFT in revolutionizing the reliability and security of day-to-day operations across industries, offering a glimpse into the transformative potential of decentralized technologies in shaping the future of sustainable digital ecosystems.

I. Introduction

Blockchain is a digital ledger system that stores transactions in a chronological order of linked blocks. One distinguishing feature of blockchain is its transparency, which allows all network participants to view transaction data while providing immutability of the data. Blockchain is a distributed network that relies on nodes, or users, to validate transactions and maintain ledger integrity. Decentralization removes a single point of failure from the system, improving overall system security and reliability. The redundancy created by replicating the ledger across all nodes reduces the risk of data loss. Nodes equipped with authorized hardware play an important role in the network's long-term viability.

Each node in the network has a unique identifier, making it easier to distinguish across nodes. These nodes, which are critical to preserving decentralization, work together to support the consensus mechanism, ensuring transaction legitimacy. Control in a decentralized blockchain network is not centralized in specific organizations or authorities.

Blockchain can broadly be classified into Permissioned or Private Blockchain, Public Blockchain, Hybrid Blockchain and Consortium Blockchain. The public chains are mainly used in the crypto world, ensuring the anonymity of the individuals involved. The private chains are useful within a closed organization, whereas for enterprise applications we need permissioned public chains which can be used for day-to-day applications.

The fundamental function of a blockchain node is to check the legitimacy of succeeding blocks of network transactions verified using a consensus mechanism that allows the nodes to agree on a single state. This validation process ensures that established blockchain network rules and regulations are followed, contributing to overall security and integrity. The nature of the consensus mechanism used varies depending on the type of blockchain and the requirements.

The next Section talks about the different Byzantine Fault Tolerance (BFT) consensus and provides the comparative study of them focusing on the enterprise blockchain, Hyperledger Fabric. In Section III the BiniBFT Consensus algorithm is proposed with its detailed design and pseudo-code, and the Section IV concludes.

II. BFT Consensus

Byzantine Fault Tolerance (BFT) is like a superhero power for computer systems, especially when lots of computers need to work together. Imagine you and your friends trying to decide on what game to play. Everyone suggests something, and we want to make sure that even if one friend gives a silly or wrong idea (like playing in the middle of the street), we still make a good choice.

Now, think of these friends as computers in a big network, working together to solve important tasks instead of just picking a game. The challenge is that sometimes, a computer might act weird, maybe due to a problem or even because someone is trying to mess things up on purpose.

BFT is the solution to this challenge. It's like having special rules that ensure the computers can still make the right decision, even if some of them are giving wrong information or acting strangely. It's like having a clever plan to outsmart any friend who might want to ruin the game.

So, why do we need BFT? Well, when computers work together, we want to be sure they can trust each other. Imagine if your computer had to make important decisions, like sending money or running a crucial program. We wouldn't want one mischievous computer to mess everything up. BFT is like having a trusty team where even if a few members try to misbehave, the majority can still make sure everything goes smoothly. In simple terms, BFT is like having a group of friends who always figure out the best game to play, no matter if some friends are being silly. It

helps computers in a network make reliable decisions, even when some of them are acting strange or being a bit naughty.

Features of BFT

Agreement on Shared State: In a distributed system, multiple participants need to agree on the state of the blockchain. A consensus mechanism ensures that all honest nodes in the network agree on a single source of truth, even if some nodes are faulty or malicious.

Security: Consensus mechanisms are designed to withstand various types of failures and attacks.

Fault Tolerance: Systems need to be able to handle faults or malicious behavior from a subset of participants. BFT mechanisms specifically are designed to handle a certain threshold of Byzantine faults.

Coordination and Synchronization: In any distributed system, nodes may need to operate concurrently and often asynchronously. Consensus mechanisms help in coordinating system actions and synchronizing state across all nodes.

Robustness: Nodes may join or leave the network, or they might become temporarily unavailable. A robust consensus mechanism can adapt to changes in the network's composition, maintaining the system's overall integrity and availability.

Existing BFT Mechanisms

This section provides a brief of available consensus mechanisms used in permissioned blockchains like Hyperledger Fabric.

pBFT

Practical Byzantine Fault Tolerance (pBFT) [1] stands as a prominent and influential consensus algorithm in distributed systems, owing to its effective Byzantine fault tolerance, operational efficiency, and relative simplicity. Among various Byzantine Fault Tolerance (BFT) algorithms, pBFT distinguishes itself with its practicality and efficiency. The algorithm employs a three-phase message exchange protocol, which includes the *Pre-prepare, Prepare*, and *Commit phases*.

In the *Pre-prepare phase*, the primary node disseminates a proposal to all other nodes in the network. Subsequently, in the *Prepare phase*, upon receiving the pre-prepare message, other nodes cast their votes on the proposal and transmit their prepared messages back to the primary node. Finally, in the *Commit phase*, if the primary node accumulates enough prepared messages, surpassing two-thirds of the total nodes, it broadcasts a commit message, thereby finalizing the proposal.

Despite its effectiveness, the standard pBFT algorithm has limits when applied to networks with many nodes, often more than 100. This constraint limits scalability, limiting its applicability in large networks. To address this problem, a multi-layer pBFT algorithm was developed to improve network scalability and reduce the obstacles posed by an increased number of consensus nodes. A multi-center pBFT algorithm, on the other hand, has been presented, with the goal of improving fault-tolerant capabilities to assure the system's smooth functioning even in increasingly demanding conditions.

Scalability of the pBFT method has been examined primarily through the lenses of communication complexity and scalability. Scalability appears to be a substantial obstacle, necessitating the development of a number of techniques to address it. Strategies such as network sharding have been implemented to increase the network's capability for expansion. However, it is critical to note that, while these approaches improve network scalability, they usually weaken system security by ignoring the consensus network's fault tolerance.

The pBFT algorithm, lauded for breaching the original Proof of Work (POW) algorithm's performance constraint, achieves improved throughput and reduced transaction confirmation wait. The algorithm, however, confronts problems such as high communication complexity, poor scalability, and limited fault tolerance. These difficulties have an impact on the performance of blockchain-related projects, making it is impossible for them to meet the strict standards of real-world business scenarios.

While pBFT remains a strong consensus algorithm, its limits in scalability and fault tolerance have prompted the development of alternate and improved versions, such as multi-layer and multi-center pBFT algorithms. The ongoing investigation and development in this sector strives to achieve a balance between scalability and security, guaranteeing that distributed systems, particularly in blockchain applications, can fulfill the demands of real-world business requirements.

Mir-BFT

Mir-BFT protocol [2], Mir-BFT protocol aims to improve network communication reliability, particularly in decentralized systems such as blockchain. Unlike prior protocols, Mir-BFT enables many leaders to suggest batches of requests simultaneously, eliminating malicious attacks that attempt to duplicate requests. Spreading the burden across multiple leaders minimizes the likelihood of one leader creating a bottleneck while increasing the network's capacity to process transactions.

One of its main advantages is how it handles client verification, which is sometimes a bottleneck in wide-area networks. Mir-BFT accelerates this process using a technique known as client signature verification sharding, which increases its efficiency even further.

Mir-BFT outperformed current protocols in tests, handling over 60,000 transactions per second, each of which is equal in size to Bitcoin transactions, across a huge network of 100 nodes connected by a 1 Gbps WAN. Processing these transactions typically takes only a few seconds,

which is impressive for such a large network.

Mir is a protocol based on pBFT (Practical Byzantine Fault Tolerance), with a few major differences that improve its efficiency and security. Its key features are:

- **Authenticate client requests**: Unlike pBFT, Mir uses signatures instead of MACs to authenticate client requests. This update eliminates some sorts of attacks connected with MACs, however it may cause a CPU bottleneck owing to the additional burden in verifying client signatures.

- **Batching and Watermarks**: Mir uses batch processing, similar to pBFT, to boost throughput. It also keeps the request and batch watermarks used by pBFT to improve efficiency. Unlike some modern BFT protocols, Mir continues to employ watermarks to allow different leaders to submit batches at the same time.

- **Protocol Round Structure**: Mir uses epochs that correspond to views in pBFT. Each epoch has an epoch primary and an epoch leader, which allows several nodes to offer batches without conflict. Epoch transitions occur in response to suspected node failures or after a predetermined amount of batches are committed.

- **Epoch Leader Selection**: The primary node selects epoch leaders and broadcasts them to all nodes. Strategies for picking leaders can differ depending on characteristics such as execution history, fault patterns, and even blockchain stake. To prevent request duplication and censorship assaults, Mir divides the hash space into buckets and assigns each one to a specific leader. This avoids duplication assaults and guarantees that requests are not ignored by leaders.

- **Mitigating Request Duplication and Censorship Attacks**: Mir separates the request hash space into buckets and gives each bucket to a specific leader. This avoids duplication assaults and guarantees that requests are not ignored by leaders.

- **Request Partitioning**: Mir divides the request hash space into buckets, each with its own leader. Bucket redistribution occurs at regular intervals to prevent request censorship and ensure load balancing.

- **Parallelism**: The Mir implementation is highly parallelized, with each worker thread handling a single batch. It also uses several gRPC (Remote Procedure Cal) connections between nodes, which is critical for enhancing throughput, particularly in wide-area networks with a small number of nodes.

In essence, Mir-BFT preserves the safety elements of PBFT while making adjustments to improve its efficiency, particularly when dealing with several leaders proposing transactions at the same time.

BFT-SMaRt

BFT-SMaRt (Byzantine Fault Tolerant State Machine Replication) [3, 4, 5] is a robust protocol designed to ensure the integrity and fault tolerance of distributed systems, particularly in scenarios where nodes may behave maliciously or exhibit faults. One critical aspect of BFT-SMaRt lies in its handling of transactions within the consensus mechanism, the validation process, and the secure signing of blocks during the commit phase.

The components are:

- Orderer node: Assembler assembles transactions into blocks
- Peer receives a Header - metadata stream from a randomly selected orderer
- Consensus consists of 3 phases: *Pre-prepare, Prepare* and *Commit*
- Client sends transactions to all the nodes in the network

In the context of block handling, BFT-SMaRt employs a systematic approach. Transactions, rather than being routed through an intermediary, are directly submitted to the BFT library. This strategy ensures that the BFT system actively monitors and prevents transaction censorship, enhancing the overall integrity and openness of the network.

Trust and validation mechanisms play a pivotal role in ensuring the correctness of transactions within the BFT-SMaRt protocol. Unlike conventional systems where followers might solely rely on the leader's decisions, BFT-SMaRt emphasizes an additional layer of security. Followers not only receive transaction proposals from the leader but also conduct transaction revalidation within the block proposal. This necessitated the introduction of specific APIs in the BFT-SMaRt library, enabling followers to verify transactions during the consensus process. This rigorous validation process significantly contributes to the protocol's resilience against potential faults or malicious behaviors by ensuring that transactions are correct and valid before finalization.

Block signing, a crucial step in ensuring the immutability and authenticity of the ledger is handled meticulously within the BFT-SMaRt consensus protocol. Unlike traditional systems where block signing might be performed by a single node, BFT-SMaRt elevates security by distributing this responsibility across multiple nodes. Specifically, the block is signed during the commit phase, and not by a single node, but by a predefined subset of nodes denoted as Q nodes. This multi-node signing approach enhances the trustworthiness and tamper-resistance of the ledger, as a malicious actor would need to compromise multiple nodes simultaneously to tamper with the committed block.

The commitment to sign blocks during the commit phase and involve multiple nodes in the signing process reinforces the security and reliability of BFT-SMaRt. This distributed approach to block signing mitigates the risks associated with single-point vulnerabilities, offering a robust solution to ensure the authenticity and integrity of the ledger.

In summary, BFT-SMaRt's transaction handling, validation mechanisms, and block signing strategies collectively contribute to its robustness and resilience against Byzantine faults, offering a highly secure and reliable framework for distributed systems.

Comparison of Existing BFT Protocols

Protocol	Latency	Security	Throughput
pBFT	Higher latency	High	High
BFT-SmaRt	Acceptable for most usecases	Compliant with Fabric standards, MSP integration	Low to Moderate
Mir-BFT	Low latency, designed for efficient throughput	Robust to Byzantine Faults	High, especially in WANs

Fig.1. Comparison of BFT Protocols

A comparative study of the candidates chosen were conducted considering the parameters *Latency, Security,* and *Throughput*, and is depicted in Fig 1. The study clearly shows that there is a need for a BFT consensus which is highly secure and scalable. To address this we present the BiniBFT Protocol in the next Section.

III. BiniBFT Consensus

The **BiniBFT Consensus Protocol** is designed to improve the scalability, fault tolerance, and throughput of blockchain networks. The design of the BiniBFT protocol has been matured than the one mentioned in version 1 [6]. Unlike traditional consensus protocols, which involve all nodes in the network, this protocol divides the network into smaller, independent **shards**. Each shard processes transactions in parallel, significantly reducing the load on individual nodes and allowing the system to scale linearly as more shards are added. The protocol uses a **Accumulator Pool**, where client requests are accumulated, validated, and batched before being sent to a **shard leader** for processing. The leader then manages the consensus process within the shard.

The consensus protocol operates in three phases: **Pre-Prepare**, **Prepare**, and **Commit**. In the **Pre-Prepare** phase, the shard leader validates the batch and broadcasts it to the followers and shard leaders where shard leaders will forward to the shard members of their shard. In the **Prepare** phase, same as **Pre-Prepare** the shards and followers validate the batch and share their votes. Finally, in the **Commit** phase, once a quorum is reached, the batch is added to the ledger. To ensure the system remains fault-tolerant, the protocol incorporates a **heartbeat mechanism** to monitor the health of shard leaders. If a leader fails or times out, a **view change**

is triggered, and a new leader is elected to ensure uninterrupted consensus. The flow follows as given below Fig 2.

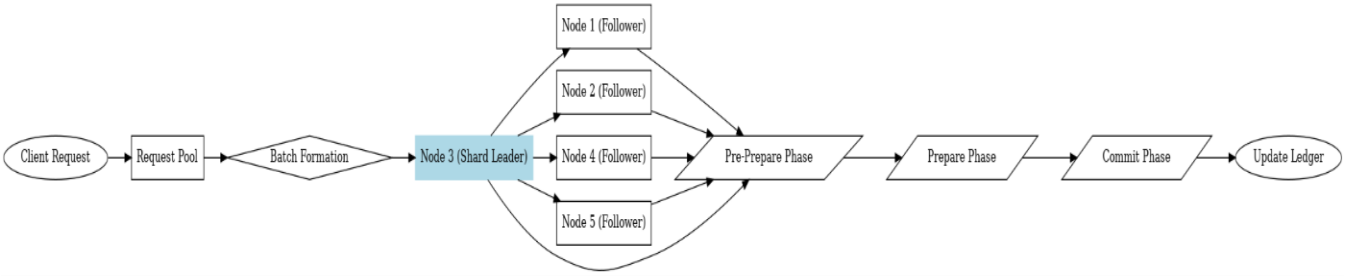

Fig.2. Flow of BiniBFT Protocol

Additionally, the protocol enhances throughput by batching client requests before they are sent to the leader, reducing the number of messages exchanged and minimizing latency. This makes the system highly efficient, especially in environments with high transaction volumes. Overall, the **BiniBFT Consensus Protocol** offers a decentralized, scalable, and resilient solution for blockchain networks. Its ability to handle faults dynamically, coupled with its ability to scale efficiently, makes it an ideal fit for large, permissioned blockchain frameworks like **Hyperledger Fabric**, where high performance and fault tolerance are critical.

Here we can see the Pseudo code for the reference, how the consensus selects the leader and follows the BiniBFT approach.

Firstly, let's see how the shards are divided.

Leader Selection from Shards

```
Input:
  - Nodes in the network `Nodes[]`.
  - Desired number of shards `NumShards`
  - n=>no of Nodes in the network.
Output:
  - `Shards[]` with elected leaders.


Algorithm:
1. Shuffle the nodes: `ShuffledNodes ← Shuffle(Nodes)`.
2. Divide the nodes into shards:
```

```
    a. For each node `n` in `ShuffledNodes`:
        i. Assign `n` to a shard: `ShardIndex ← Index(n) % NumShards`.
        ii. Add `n` to `Shards[ShardIndex]`.

3. Elect a leader for each shard:
    a. For each `Shard` in `Shards`:
        i. Select a Leader: `Shard.Leader ← ElectLeader(Shard.Nodes)`.

4. Output the shards with their leaders.
```

Lets see how the request flows from client to the commit phase

1. Request Handling and Consensus Flow

```
Input:
  - Client Requests: Requests submitted by external clients.
      Here r=>Request comes from the client
           sig=>Signature of the client
Output:
  - Batch of requests forwarded to a shard leader.

Algorithm:
1. Initialize an `AccumulatorPool`.

2. Upon receiving a client request `REQUEST(r, sig)`:
    a. Verify the request signature: `SigVer(r, sig)`.
    b. Add the request `r` to the `AccumulatorPool`.
```

3. Periodically check the `AccumulatorPool`:
    a. If `size(AccumulatorPool) >= BatchSize` OR `Timeout reached`:
        i. Create a batch `Batch ← ExtractRequests(AccumulatorPool, BatchSize)`.
        ii. Forward the batch to a randomly selected leader:
            - `Leader ← RandomSelect(ShardLeaders)`.
            - `send(Batch) to Leader`.

2. Consensus Protocol

Phase 1: Pre-Prepare

Input:
- Batch received by a shard leader `Batch`.

Output:
- `PRE-PREPARE` message sent to shard followers and other leaders.

Algorithm:
1. Upon receiving `Batch` at a leader:
    a. Validate the batch origin: `IsValidOrigin(Batch)`.
    b. Verify all requests in the batch:
        i. For each request `r` in `Batch`:
            - Validate signature: `SigVer(r, σc)`.

2. If valid:
    a. Broadcast `PRE-PREPARE(Batch)` to:
        i. Shard followers.

ii. Leaders of other shards.

Phase 2: Prepare

```
Input:
  - `PRE-PREPARE(Batch)` received by followers and other shard leaders.
Output:
  - `PREPARE` message sent to shard followers and other leaders.
Algorithm:
1. Upon receiving `PRE-PREPARE(Batch)`:
    a. Validate the batch:
        i. Ensure it has not been processed before.
        ii. Confirm the sequence number matches the shard's state.

2. If valid:
    a. Broadcast `PREPARE(Batch)` to:
        i. Shard followers.
        ii. Other shard leaders.
```

Phase 3: Commit

```
Input:
  - `PREPARE(Batch)` messages received by followers and other leaders.
Output:
  - `COMMIT(Batch)` message sent to shard followers and other leaders.

Algorithm:
1. Upon receiving sufficient `PREPARE` messages:
    a. Validate quorum for the batch.
    b. Broadcast `COMMIT(Batch)` to:
```

```
        i. Shard followers.
        ii. Other shard leaders.

2. Upon receiving sufficient `COMMIT` messages:
  a. Add the batch to the ledger: `CommitToLedger(Batch)`.
```

Lets see how the heartbeat between leader and followers happens and view change if leader fails to send the heart beat.

1. Leader Heartbeat Broadcast

```
Input:
  - Shard `Shard`.
  - Heartbeat interval `HeartbeatInterval`.

Algorithm:
1. Initialize:
    - `LastHeartbeatSent ← currentTime()`.
    - `Shard.Followers ← Nodes - {Leader}`.

2. Periodically (every `HeartbeatInterval`):
    a. For each follower `f` in `Shard.Followers`:
        i. Send `HEARTBEAT(LeaderID, Timestamp)` to `f`.
    b. Update `LastHeartbeatSent ← currentTime()`.

3. Log the successful transmission.
```

2. Follower Heartbeat Monitoring

```
Input:
```

- Received heartbeats `HEARTBEAT(LeaderID, Timestamp)`.
- Heartbeat timeout threshold `HeartbeatTimeout`.

Algorithm:

1. Initialize:
   - `LastHeartbeatReceived ← currentTime()`.

2. Upon receiving a `HEARTBEAT(LeaderID, Timestamp)`:
   a. Validate the heartbeat:
      i. Ensure `LeaderID` matches the expected leader.
      ii. Ensure `Timestamp` is within the expected range:
         - `currentTime() - Timestamp ≤ HeartbeatInterval`.
   b. If valid:
      i. Update `LastHeartbeatReceived ← currentTime()`.

3. Periodically (every `HeartbeatInterval`):
   a. If `currentTime() - LastHeartbeatReceived > HeartbeatTimeout`:
      i. Mark leader as failed:
         - `LeaderStatus ← FAILED`.
      ii. Trigger a view change:
         - `TriggerViewChange()`.

3. View Change Process

Input:

- Detected leader failure.

```
Algorithm:

1. Upon detecting a failed leader:

    a. Broadcast `VIEWCHANGE(OldLeaderID, Timestamp)` to all nodes in the shard.

    b. Collect `VIEWCHANGE` messages from other nodes.


2. If `VIEWCHANGE` quorum is met:

    a. Elect a new leader:

        - `NewLeader ← ElectNewLeader(Shard.Nodes - {OldLeaderID})`.

    b. Notify all followers:

        - `send LEADERCHANGE(NewLeaderID, Timestamp)` to `Shard.Followers`.

    c. Resend pending batches to the new leader:

        - `ResendPendingBatches(NewLeader)`.
```

IV. Conclusion

This work presents the integration of the BiniBFT consensus mechanism into the Hyperledger Fabric (HLF) framework, introducing advanced Byzantine Fault Tolerance (BFT) capabilities. Inspired by the SmartBFT model, our approach leverages critical innovations such as Verifiable Random Function (VRF)-based leader election, dynamic sharding, and optimized random polling for consensus validation. These enhancements are strategically designed to address scalability and fault tolerance challenges in permissioned blockchain environments.

By implementing this sharded BFT-driven consensus model within HLF, we aim to streamline transaction workflows, from batching and leader proposal to network-wide validation and commit phases. This architecture reduces communication overhead and ensures high resilience against adversarial behavior or node failures.

We anticipate that the integration of BiniBFT will lead to a substantial increase in Hyperledger Fabric's throughput, achieving higher transactions per second (TPS) without compromising security or consistency. This advancement positions Hyperledger Fabric as a robust and scalable solution for enterprise-grade blockchain applications requiring high performance and fault tolerance.

References

[1] Miguel Castro and Barbara Liskov Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, 1999 https://pmg.csail.mit.edu/papers/osdi99.pdf

[2] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, Marko Vukolic, Mir-BFT: High-Throughput Robust BFT for Decentralized Networks arXiv:1906.05552v3 [cs.DC] 22 Jan 2021] https://arxiv.org/pdf/1906.05552.pdf

[3] João Sousa, Alysson Bessani, (2014). State Machine Replication for the Masses with BFT – SMaRt

https://www.researchgate.net/publication/286593169_State_machine_replication_for_the_masses_ with_BFT-SMART

[4] Barger, Artem, et al. "A byzantine fault-tolerant consensus library for hyperledger fabric." 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). IEEE, 2021.https://arxiv.org/pdf/2107.06922.pdf

[5] Hyperledger Fabric v3.0.0-preview Release Notes - September 1, 2023 https://github.com/hyperledger/fabric/tree/v3.0.0-preview

[6] BiniBFT Whitepaper v1 - January 15, 2024 – Anasuya Threse Innocent et al, “BiniBFT - A Secure, Scalable BFT for Hyperledger Fabric”, https://github.com/BiniWorld/Hyperledger-BiniBFT/blob/main/BiniBFT%20Whitepaper.pdf