IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 2, 2023, accepted July 17, 2023. Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

SoK: A Taxonomy for Critical Analysis of Consensus Mechanisms in Consortium Blockchain

WEI YAO[1], (Student member, IEEE), FADI P. DEEK[1], RENITA MURIMI[2], (Senior member, IEEE), and Guiling Wang[1], (Fellow, IEEE)

[1]New Jersey Institute of Technology, University Heights, Newark, NJ 07102 USA
[2]University of Dallas, 1845 East Northgate Drive Irving, TX 75062 USA

Corresponding author: Wei Yao (e-mail: wy95@njit.edu)

The research is partially supported by FHWA EAR 693JJ320C000021

ABSTRACT Consensus algorithms are central to blockchain technology and an emerging research area. In this paper, we begin with an overview of the different types and architectures of blockchain networks. Then, with a focus on consortium blockchains, we survey, classify, and assess their principal consensus mechanisms. Furthermore, as consensus mechanisms determine network reliability, enhance performance efficiency, and ensure system security, we conduct a critical analysis of the strengths and weaknesses of consensus algorithms using a taxonomy of three different criteria: reliability, performance, and security. We conclude with insights into current and future research challenges and opportunities in this domain.

INDEX TERMS Consensus, Consortium blockchain, Taxonomy

I. INTRODUCTION

Blockchain, a decentralized, immutable, and transparent distributed ledger, maintains a continuously growing list of transaction records ordered into blocks. At its core lies the so-called consensus algorithm, an agreement to validate the correctness of blockchain transactions. By their nature, public blockchains are resource-intensive technology. For example, in Bitcoin, each node uses the Proof of Work (PoW) algorithm to reach a consensus by competing to solve a puzzle [1]. Being less resource-intensive, a private blockchain functions as a distributed but secure ledger for a particular organizational purpose. A hybrid of both public and private blockchains, a consortium blockchain is an enterprise-level ledger that does not contend with issues of creating a resource-saving global consensus protocol like public chains. This section covers some important fundamental aspects of blockchain and overviews related work.

Nakamoto proposed Bitcoin in 2008 [1], heightening interest in the blockchain technology that forms the foundation for digital currency [2]. The consensus algorithm provides a process for all nodes to seek and reach a common agreement in a distributed, increasingly untrusted environment. Since Bitcoin, many cryptocurrencies have emerged [3]. Among them, Ethereum [4] is noteworthy for introducing the concept of smart contracts, which allow contracts to be coded on the blockchain and use Ethereum as a platform for currency transactions. Ethereum and Bitcoin share the feature of being public and allowing any node to participate in network activities, with similar consensus mechanisms. In 2015, the Linux Foundation initiated Hyperledger Fabric [5] as a solution specifically designed for enterprise-level applications, in contrast to the open nature of Bitcoin and Ethereum without any authentication mechanisms. Unlike Bitcoin's incentive mechanism, Fabric uses permissioned blockchain to increase energy efficiency and performance. With the rapid development of blockchain technology, more enterprise-level users have begun considering blockchain to meet their business needs [6]–[8]. Therefore, exploring effective consensus protocols for use in consortium blockchains has developed into a research problem of emerging significance. The release of Facebook's Libra project in 2019 [9] led to a new round of cryptocurrency interest, which in turn further increased attention from investors and researchers. Among various applications of blockchain, a notable one is that of digital governance. In what is touted as Web 3.0, the prevalent use of blockchain has accelerated the pace of innovation; thus, the requirements for consensus have also risen to a new level.

There are a number of surveys on blockchain and applications [10]–[13]. Similarly, surveys on blockchain consensus protocols have been published in the literature [14], [15] and presented on arXiv [16]–[18]. Nguyen et al. [15] provided a tutorial-style review on distributed consensus protocols that classifies consensus algorithms into proof-based and voting-based on the mechanism of reaching consensus. Important

protocols, such as RBFT, HotStuff, and LibraBFT are not covered. Salimitari et al. [16] reported on consensus algorithms and their applicability in the IoT areas. As noted [15], multiple important protocols, including LibraBFT, are missing. Consensus protocols such as LibraBFT [9] are relevant not only because they are suitable for enterprise scenarios but because they include features of public blockchain consensus protocols, such as incentive mechanisms. The survey of Ferdous et al. [19] also missed multiple important protocols. Recently, Badr et al. [20] provided a comprehensive survey on distributed ledger technologies, but they mainly focused on the network models, not consensus algorithms.

Given that a comprehensive survey covering all the important consensus protocols for consortium blockchains remains missing, as well as considering the importance of consensus mechanisms and the rapid development of enterprise-level blockchains, this paper fills a gap by providing a systematic taxonomy of enterprise-level blockchain consensus protocols and a detailed analysis of each protocol, considering aspects such as reliability, performance, and security. The remaining parts of this paper are organized as follows: Section II overviews blockchain technology and the notion of consensus algorithms. Section III presents our proposed taxonomy with algorithmic workflow for achieving fault tolerance and a comparative analysis of representative categories. Section IV addresses the degree of achieved reliability. Section V focuses on performance, while section VI focuses on security. Finally, Section VII offers concluding remarks and presents research challenges and opportunities to motivate future work.

II. BUILDING BLOCKS

A. BLOCKCHAIN OVERVIEW

The nomenclature of **blockchain** is derived from its architecture: each block is linked cryptographically to the previous block. The **genesis block** is the first block, and each block has a set of transactions. Blockchain has the following characteristics: decentralization, trustlessness, immutability, and anonymity. Decentralization means there's no central trusted third party. Trust is established through consensus. Blockchain is tamper-proof, and it ensures some degree of anonymity and privacy-protection technologies like group signatures, ring signatures, and zero-knowledge proofs [21].

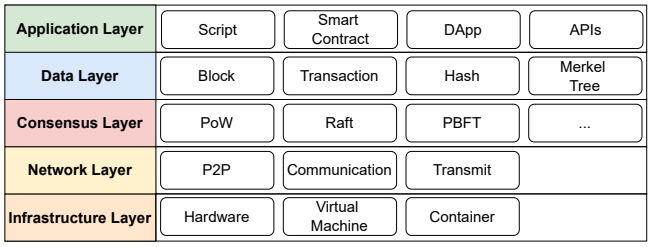

FIGURE 1. Blockchain Architecture

The framework of the blockchain is shown in Figure 1. It comprises the following layers from bottom up: (1) **Infrastructure Layer**: Contains hardware, architecture equipment, and deployment environment for blockchain systems. (2) **Network Layer**: Includes the node organization method, communication mechanisms, and transmit protocol. P2P networks are used with a flat topology, where nodes are equal, distributed, and autonomous [22]. (3) **Consensus Layer**: Forms the core of the consensus protocol used to ensure trust and security in the network and to achieve consistency of nodes participating in the distributed ledger. (4) **Data Layer**: To realize traceability and non-tampering, transaction data is recorded through the blockchain structure and can be tracked through this chain ledger [23]. For example, each data block in Bitcoin comprises a block header and a block body containing a packaged transaction, shown in Figure 2. The block header contains information such as the current system version number, the previous block's hash value, the random number, the root of the Merkel tree of the block transaction, and the timestamp [1]. The block body includes verified transactions and a complete Merkel tree composed of these transactions [24]. The Merkel tree is a binary tree, where the bottom layer corresponds to the content of the leaf node. Each leaf node is the hash value of the corresponding data. Two neighboring leaves unite to perform a hash computation that becomes the content of the upper-level node. Recursive computations form the content of the root node. Blockchain's non-tampering is ensured by Merkle tree's particular data structure since any modification in the leaf will be passed to its parent and propagated to the tree's root [25]. (5) **Application Layer**: Encapsulates various script codes, smart contracts, Decentralized Applications (DApps) and APIs. Scripts are sets of instruction lists attached to transactions. Smart contracts are event-driven, stateful computer programs that run on a shared blockchain data ledger, processing data and managing on-chain smart assets. DApps are decentralized and secure applications that run on a distributed network, utilizing open-source code and storing data and records on the blockchain using cryptographic technologies. APIs are provided for development, allowing DApps and third-party applications to handle smart contracts and retrieve data from blockchain.

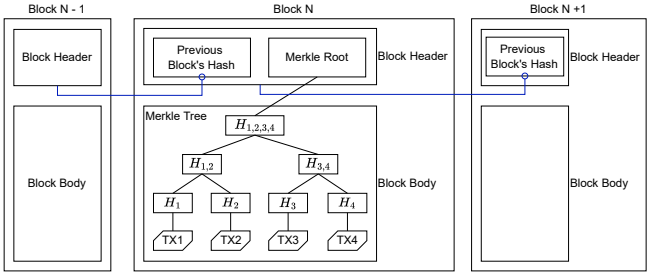

FIGURE 2. An example of block and chain structure

B. TYPES OF BLOCKCHAIN NETWORKS

Blockchain networks can be categorized as public, consortium, or private in order of decreasing degrees of openness available for participation by nodes.

Public Blockchain: Referred to permissionless blockchain, enable any node to enter and exit the network [26]. It is completely decentralized, and each node can participate anonymously without registration, authorization, or authentication [27]. Cryptography-related technologies such as digital signatures, hashing [28], symmetric/asymmetric keys [29], and ECDSA [30] are used to ensure that transactions cannot be tampered with. Economic incentives such as transaction fees and rewards motivate consensus nodes to participate in the consensus process.

Private Blockchain: Known as the permissioned chain, is generally not open to the outside world and is only used by individuals or institutions [11]. Access to read and write on the private blockchain is governed by rules established by private organizations. Private chains prioritize preventing internal and external security attacks on data and providing users with a secure, tamper-proof, and traceable system. They offer a certain degree of centralized control instead of complete decentralization, sacrificing some of the latter's benefits for better performance compared to public chains.

Consortium Blockchain: A hybrid architecture comprising features from public and private blockchains in which participation is limited to a consortium of participating members [14]. Each node may refer to a single organization or institution in the consortium. The number of nodes is determined by the size of the pre-selected participants in the blockchain. For example, a financial blockchain is designed for a consortium of 30 financial institutions and allows 30 nodes in this consortium blockchain. The number of nodes required to reach a consensus depends on which algorithm the consortium blockchain uses. The consortium chain accesses the network through the gateways of member institutions and generally provides members' information authentication, data read and write permission authorization, network transaction monitoring, member management, and other functions. Each member can have permissions assigned by the consortium to access the ledger and validate the generation of blocks. The Hyperledger project is an example of consortium blockchain architecture. Since there are relatively few nodes participating in the consensus process, the consortium blockchain generally does not use the PoW mining mechanism as the consensus algorithm. Consortium blockchain requirements for transaction confirmation time and transaction throughput are, thus, different from those of public blockchains.

C. CONSENSUS ALGORITHMS

A consensus algorithm ensures the integrity of a distributed ledger by facilitating agreement among nodes on transaction content and order. Without it, data inconsistency can occur, leaving the ledger vulnerable to manipulation. Consensus algorithms can address several blockchain problems, including:

The CFT Problem: CFT consensus algorithms only guarantee a blockchain's reliability and resiliency to blockchain node failure [15]. Also known as non-Byzantine errors, node failures can be caused by failed hardware, crashed processes, broken networks, or software bugs. CFT cannot address scenarios involving malicious activities, referred to as Byzantine errors. When nodes intentionally and maliciously violate consensus principles, e.g., tampering with data, a CFT algorithm cannot guarantee the system reliability. Thus, CFT consensus algorithms are mainly used in closed environments such as enterprise blockchains. Current mainstream CFT consensus algorithms include Paxos and Raft. The latter is a derivative of the former and a simplified consensus algorithm designed to be more suitable for industry implementation.

The BFT Problem: Unlike CFT problems that deal with crashes or failures, a Byzantine fault is caused by malicious nodes sending incorrect information to prevent other nodes from reaching consensus. In distributed systems, the Byzantine General's Problem translates into an inability to maintain consistency and correctness under certain conditions. The Byzantine Generals Problem, proposed by Lamport [48], is described as follows. Several Byzantine armies are camping outside an enemy city, and a general commands each army. The generals can only communicate with each other by dispatching a messenger who carries messages back and forth [48]. After assessing the enemy's situation, they must agree on an identical action plan. However, some traitors among these generals may prevent loyal generals from reaching an agreement. The generals require an algorithm to guarantee that all loyal generals reach a consensus, even if a small number of traitors cheat. Let $v(i)$ represent the information

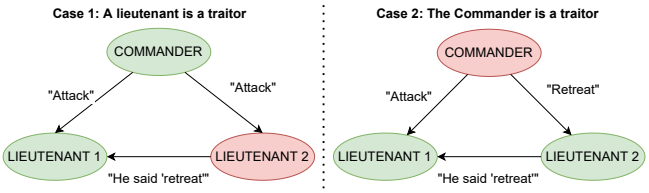

FIGURE 3. Byzantine General Problem

sent by the i-th general. Each general draws up a battle plan based on $v(1)$, $v(2)$, $\cdots$, $v(n)$, where n is the number of generals. The problem can be described in terms of how a commanding general issues orders to lieutenants. Therefore, the problem will be transformed into the following **Byzantine General Problem**: A commander issues an order to his $n-1$ lieutenants such that: (a) IC1. All loyal lieutenants obey the same order. (b) IC2. If the commander is loyal, each loyal lieutenant must obey the order. IC1 and IC2 are conditions for interactive consistency, which is a configuration that includes the number of generals in a final agreement [48].

One case of the Byzantine Generals Problem is shown in Figure 3. Here, the Commander and Lieutenant 1 are loyal, and Lieutenant 2 is a traitor. The Commander issues an attack order to all lieutenants. Lieutenant 2 is a traitor, and

TABLE 1. Mainstream Platforms and Consensus Algorithms

Blockchain Platforms	Consensus Algorithm	Use Cases Areas	Smart Contract	DApps	Open Source	Open APIs
Antchain	HoneybadgerBFT [31]	Multi-Purpose	x	✓	x	✓
Cardano	Ouroboros [32]	Cryptocurrency	✓	✓	✓	✓
Enterprise Ethereum	Customized	Multi-Purpose	✓	✓	✓	✓
FISCO BCOS	Raft [33], PBFT [34]	Multi-Purpose	✓	✓	✓	✓
Google Chubby	Paxos [35]	Distributed System	x	x	✓	✓
Hedera	Hashgraph [36]	Cryptocurrency	✓	✓	x	✓
Hyperledger Besu	QBFT, IBFT [37]	Multi-Purpose	✓	✓	✓	✓
Hyperledger Burrow	Tendermint [38]	Multi-Purpose	✓	✓	✓	✓
Hyperledger Fabric	PFBT, Raft [33], Kafka [39]	Supply chain	✓	✓	✓	✓
Hyperledger Firefly	IBFT [37]	Multi-Purpose	✓	✓	✓	✓
Hyperledger Indy	RBFT [40]	Identity Management	x	✓	✓	✓
Hyperledger Iroha	YAC [41]	Multi-Purpose	x	✓	✓	✓
Hyperledger Sawtooth	PoET [42]	Multi-Purpose	✓	✓	✓	✓
IPFS private	Raft [33]	Data Storage	✓	✓	✓	✓
Libra	LibraBFT [9]	Cryptocurrency	✓	✓	x	✓
MOBI	MOBI Consensus	IoT	✓	✓	x	✓
Neo	dBFT [43]	Cryptocurrency	✓	✓	x	✓
Quorum	IBFT, Kafka [39]	Financial	✓	✓	x	x
R3 Corda	Customized	Multi-Purpose	✓	✓	✓	✓
Ripple	RPCA [44]	Cryptocurrency, Financial	x	x	x	✓
Stellar	SCP [45]	Cryptocurrency, Financial	x	✓	x	✓
Symbiont	BFT-SMART [46]	Fintech	x	x	x	x
Tendermint	Tendermint [38]	Multi-Purpose	✓	✓	x	✓
TradeLens	Raft [33]	Supplychain	✓	✓	x	✓
Trusted IoT Alliance	Customized	IoT	✓	✓	x	✓
VeChain	Proof of Authority [47]	Supplychain	x	✓	x	✓
Zookeeper	Paxos [35], Kafka [39]	Distributed System	x	x	✓	x

he/she deceives Lieutenant 1 by sending a tampered message called "retreat". Since Lieutenant 1 does not know whether the Commander or Lieutenant 2 is a traitor, he/she cannot judge which message includes the correct information and, thus, cannot reach a consensus with the loyal Commander. In another case, shown in Figure 3, the two lieutenants are loyal, and the Commander is a traitor. The Commander issues different orders to the two lieutenants. Lieutenant 2 conscientiously delivered the information of the Commander to Lieutenant 1. Lieutenant 1 cannot judge which information is correct, resulting in loyal lieutenants not reaching a consensus.

If there are f traitors and the total number of generals n is less than $3f + 1$, the Byzantine Generals Problem has no solution. Lamport [48] proposed a solution to solve the Byzantine Generals Problem in exponential time $\mathcal{O}(n^f)$ if the adversary mode is $n = 3f + 1$. This original BFT algorithm is computationally expensive to implement, and a practical BFT algorithm is introduced later.

The **adversary model** represents a malicious entity that aims to prevent non-malicious entities from achieving their goal [49]. An adversary model imposes a specific limit on the percentage of computing power or property that an adversary can hold, generally represented by f for the number of adversaries and n for the total number of nodes. For example, if a BFT algorithm's adversary model is $n = 3f+1$, it implies that if the algorithm can tolerate f faulty replicas, the system requires a minimum number of $n = 3f + 1$ replicas.

D. CONSENSUS ALGORITHM CLASSIFICATION

One way of classifying consensus algorithms is by their approach to making final decisions to reach consensus [15]. The first category is proof-based consensus algorithms, since a node in this category has to compete with other nodes and prove that it is more qualified than others to commit transactions. PoW [1], PoS [50], Proof of Authority (PoA) [51], Proof of Elapsed Time (PoET) [52], and Proof of Space (PoSpace) [53] are algorithms of this category. The other category is that of voting-based algorithms since the commitment depends on which committed result wins the majority of votes. Paxos [35], Raft [33], PBFT [34], RFBT [40], RPCA [44], SCP [45], Tendermint [54], and HotStuff [55] belong to this category. The first group of consensus algorithms is proof-based, while the second group is voting-based. Another way of classifying consensus algorithms is by the design principle of fault tolerance. Nodes can suffer from non-Byzantine errors (also known as crash faults), which is exemplified by situations where the node fails to respond. Alternatively, nodes can forge or tamper with the information and respond maliciously, causing Byzantine Fault. Thus, consensus algorithms may be classified as being designed for Crash Fault Tolerance (CFT) or Byzantine Fault Tolerance (BFT). This classification method only focuses on the original design principle. Most BFT-based consensus algorithms can tolerate either crash fault or Byzantine fault. Since the design principle of algorithms in the previous proof-based family is different from fault tolerance, those proof-based families will

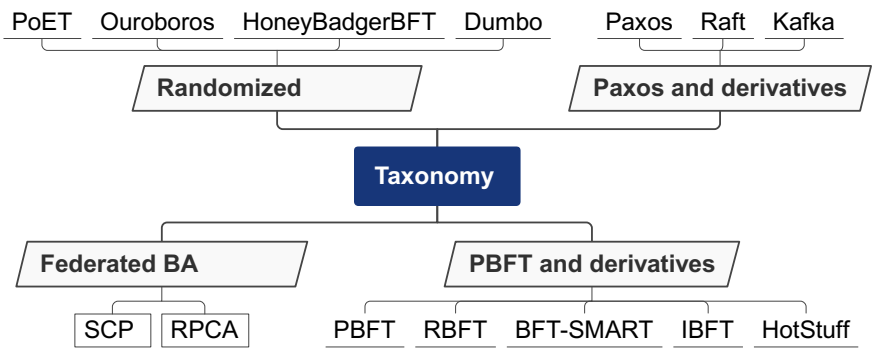

FIGURE 4. Proposed Taxonomy for Assessing Consensus

be excluded from this classification. Paxos [35], Raft [33], and Zab [56] belong to the category of CFT-based consensus algorithms. A number of variants of PBFT [34] algorithms, such as RBFT [40], SBFT [57], BFT-SMART [46], DBFT [43], and HotStuff [55] form a collection in the category of BFT-based consensus algorithms. Another group of consensus algorithms, forming a collection in the same category, uses Byzantine Federated Agreement (BFA) [45] for voting, such as RPCA [44] and SCP [45].

In the next section, we will propose a more comprehensive taxonomy, aiming to better capture the nuances and complexities of the evolving landscape of consensus algorithms.

III. TAXONOMY OF CONSENSUS MECHANISMS

Mainstream consortium blockchains and distributed systems [10]–[20] considered in this paper are shown in Table 1 along with their consensus algorithms, use cases areas, and availabilities for smart contract, DApps, open source, and open APIs. From Table 1, a representative list of consensus algorithms used in mainstream consortium blockchains are selected based on their methodologies of achieving consensus. Figure 4 shows our proposed taxonomy of consensus algorithms.

A. PAXOS AND DERIVATIVES

Paxos [35] is a classical and widely used consensus algorithm in distributed systems. Due to its efficiency and usability, its derivatives are adopted in many blockchain platforms.

1) Paxos

A fault-tolerant consensus algorithm that relies on message passing in a distributed system [35]. It divides nodes into three roles: **proposer**, **acceptor**, and **learner**. Each node can have multiple roles simultaneously. A proposer is responsible for presenting a proposal and awaiting acceptors' responses. An acceptor votes on the proposal. A learner is informed of the proposal's result but does not participate in voting. Paxos ensures the proposal's uniqueness with a proposal number and content with a value. When more than half of the acceptors approve a proposal, it is considered Chosen. Paxos ensures safety and liveness, ensuring the proposed value is chosen and the proposal is completed within a limited time.

Paxos execution is divided into two phases, as shown in figure 5. In the **Prepare** phase, the proposer sends a Prepare request with a proposal number to more than half of the acceptors in the network to determine whether a majority is prepared to accept the proposal. After receiving the proposal, the acceptor stores the largest proposal number it has received and returns a Promise message to the proposer if the proposal number is greater than the saved maximum proposal number. The acceptor promises not to accept any proposal with a number less than the proposal number that has already been received. In the **Accept** phase, the proposer broadcasts an Accept request with the proposal if it receives more than half of the responses as Promise messages. The request contains a proposal number and the value that the node would like to propose. If the response does not contain any proposal, the proposer determines the value. If the response message contains a proposal, the value is replaced by the value in the response with the largest proposal number. After an acceptor receives the Accept request, it accepts the proposal and updates the accepted maximum proposal if the proposal number is not less than the maximum proposal number promised by the acceptor. If a majority of acceptors accept the proposal, the value is Chosen, indicating consensus has been reached.

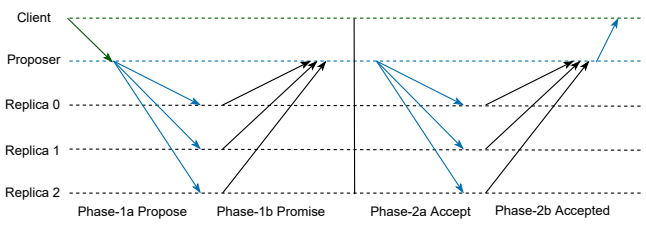

FIGURE 5. Paxos two phases.

2) Raft

Motivated by Paxos, Raft is designed for ease of understandability and implementability for industry applications [33]. Its core premise is that servers start from the same initial state

and execute a series of command operations in the same order. Its goal is to achieve a consistent state, and therefore, it uses the log method for synchronization, a consistent algorithm for managing replicated logs. Raft divides nodes into three mutually convertible roles: **leader**, **follower**, and **candidate**, with only one leader in a cluster of at least five nodes. The leader handles client requests, replication logs, and communication with followers. Initially, all nodes are followers, which passively respond to Remote Procedure Call (RPC) requests from the leader. Followers do not communicate with each other but instead respond to log replication and election requests from the leader and candidate nodes, respectively. If a follower receives a request from a client, it forwards it to the leader. A candidate initiates an election vote when the leader becomes inactive, and one or more nodes may switch from follower to candidate. Once a candidate wins the election, it becomes the new leader and can revert to a candidate if a new leader is elected but then fails.

Raft runs in two phases. The first phase is leader election, where a leader sends heartbeat messages to followers to maintain its authority. If a follower does not receive the heartbeat within an Election timeout period, it switches to a candidate role and starts an election process, signaling that the leader has failed [33]. Then, it increases its term, sends RequestVote RPC to other servers, and waits for results. If a candidate wins the election, it becomes the leader. If another server wins the election, the candidate becomes a follower. If no one wins, the election is reinitiated. The second phase is log replication, where the leader accepts client requests, updates log, and sends a heartbeat to followers to synchronize leader's log.

B. PBFT AND DERIVATIVES

PBFT is the practical BFT algorithm for enterprise-level distributed systems. Numerous variants are proposed to improve its ability to solve more challenges in blockchain.

1) Practical Byzantine Fault Tolerance (PBFT)

It is a consensus algorithm based on state machine replication [34] where services are replicated on different nodes in a distributed system. Each copy of the state machine saves the state of the service and the operations. PBFT reaches consensus through three phrases: **Pre-prepare**, **Prepare**, and **Commit**. In PBFT, there is one primary node out of n nodes, and others are backup. If the primary node fails, the backups nodes initiate a view-change to select a new primary node.

The process of reaching consensus in PBFT is as follows: (a) **Propose**: The client uploads the request message m to the primary node and replicas. (b) **Pre-prepare**: The primary node receives the request message m and generates a Pre-prepare message $\langle \text{Pre-prepare}, \mathsf{H}(m), s, v \rangle$, where $\mathsf{H}(m)$ is a one-way hash function, s is the message sequence number, and v represents the current view. The primary node signs the message with its private key and sends it to replicas. (c) **Prepare**: Upon receiving the Pre-prepare message, replicas verify the message and create a Prepare message $\langle \text{Prepare}, \mathsf{H}(m), s, v \rangle$ and broadcast it to the network. If a replica node receives $2f + 1$ valid Prepare messages, it generates a prepared certificate. (d) **Commit**: If a replica has a prepared certificate, it broadcasts a Commit message $\langle \text{Commit}, s, v \rangle$ and stores the message m in the local log. If a replica receives $2f + 1$ valid Commit messages, it generates a committed certificate, indicating the message has been successfully committed. (e) **Reply**: Once a node receives $2f + 1$ valid Commit messages, it sends the committed certificate and the message m to the client. PBFT includes a view-change protocol in case of primary node failure, where a replica triggers a view change to elect a new primary node if it has not received a response from the current primary after a timeout, ensuring liveness.

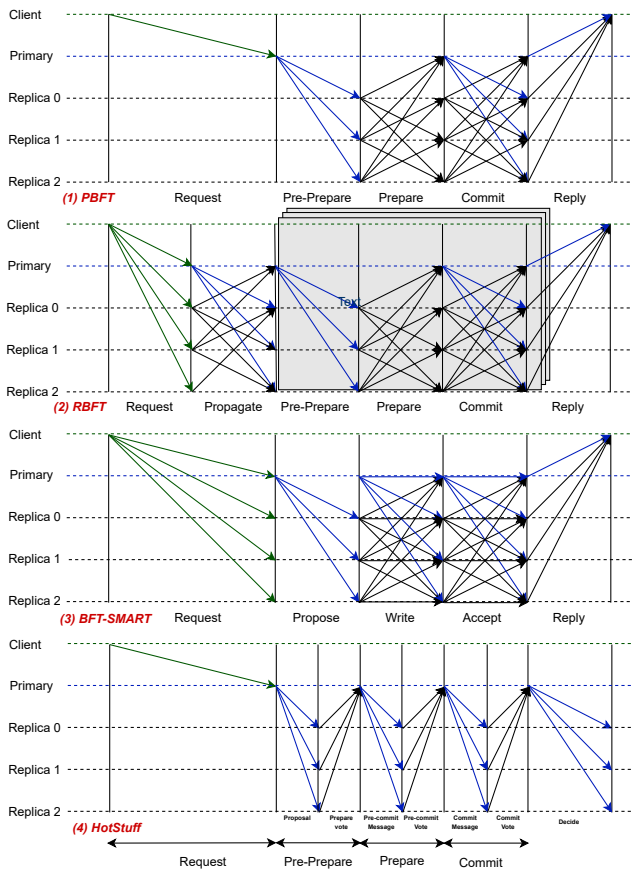

FIGURE 6. Processes of PBFT algorithm and its derivatives

2) Redundant Byzantine Fault Tolerance (RBFT)

A variant of PBFT that improves robustness by using a multi-core architecture [40]. In RBFT, each node runs $f + 1$ PBFT protocol instances in parallel, with only one instance serving as the master and the rest as backup. Each instance has its own n replicas, and in $f + 1$ instances, each node can have at most one primary.

RBFT uses a communication process similar to PBFT in the consensus protocol phase, but adds a propagate phase

before the **Pre-prepare** phase to ensure all correct nodes eventually send the request to the next phase, as shown in Figure 6. $f+1$ PBFT instances must receive the same client request for correctness, which is achieved by forwarding the message to each other. Once a node receives $2f+1$ requests from a client, it sends the request to $f+1$ instances and moves to the next phase, following the 3-phase process similar to PBFT [34], and is represented in steps 3, 4, and 5 in Figure 6. The algorithm is performed by the $f+1$ instances during the consensus protocol, and the result is returned to the client through MAC authentication messages. When the client receives $f+1$ valid and consistent replies, it accepts these replies as the result.

RBFT improves upon PBFT by implementing a monitoring mechanism and a protocol instance change mechanism to promote robustness [40]. Each node runs a monitoring program to monitor the throughput of all $f+1$ instances, and if $2f+1$ nodes find that the performance difference between the master and the best backup instance reaches a certain threshold, then the primary of the master instance is considered as a malicious node. A new primary is then selected, or the primary in the backup instance with the best performance is chosen, and it is elevated to master. To update all instances' primaries, each node maintains a counter to record the change information of each instance. If a node needs to change the primary, it sends an Instance-change message with a MAC authenticator to all nodes. After receiving the incoming Instance-change message, each node verifies the MAC and compares it with its counter. If the counter is larger, it discards the message. Otherwise, the node checks whether it also needs to send the Instance-change message by comparing the performance of the master and backup. If $2f+1$ valid Instance-change messages are received, the counter is incremented by one, and this starts the view-change process to update all instances' primaries, including the master's.

3) BFT-SMART

A state machine replication library written in the Java [46]. It supports state transfer services to repair a failed node, add/remove replicas through a Trusted Third Party client, and access other nodes to obtain the latest replica status. The system ensures stable error recovery from f faulty nodes by storing each node's operation logs on other disks. BFT-SMART also divides nodes into leaders and backups and employs a reconfiguration protocol [58] similar to the view-change protocol in PBFT to handle failure.

The consensus process in Mod-SMaRt [59] is based on a leader-driven algorithm [60] with three phases: **Propose**, **Write**, and **Accept**, as shown in Figure 6. A leader is elected from the network. When a client initiates a request, it sends a Request message containing the client serial number, digital signature, and operation request content to all nodes and waits for a response. In the normal phase, the leader verifies the correctness of the received Request message. After verification, the leader accepts the message, assigns a serial number, and sends the Request message to the replicas. As long as a replica accepts the message and forwards it, other nodes will also receive and send the Write message to all nodes, including itself. When a node receives $2f$ Write messages, the node broadcasts an Access message to all nodes, including itself. When a node receives $2f+1$ Access messages, the request is executed. The protocol stores content of the series of request operations and the encrypted certificate in each node's log and replies Access to client.

In BFT-SMaRt, if a node experiences two timeouts, it enters the synchronization phase and the reconfiguration protocol re-elects the leader node. During first timeout, the consensus and reconfiguration processes can execute simultaneously. The Request message is automatically forwarded to all nodes after the first timeout, and a Stop message is sent to other nodes after the second timeout. Once a node receives more than f Stop messages, it starts the next reconfiguration phase immediately. After the leader election, all nodes send a Stopdata message to the new leader. If the leader accepts at least $n-f$ valid Stopdata messages, it sends a Sync message to all nodes. Replicas start to synchronize if the leader is valid.

4) HotStuff

A partially synchronized network [61] with an adversary model of $n=3f+1$. It uses a parallel pipeline to process a proposal, which is equivalent to combining the preparation and commitment phases of PBFT. The original proposal includes two implementations of HotStuff, Basic HotStuff, and Chained HotStuff. The Basic HotStuff protocol forms the core of HotStuff, which switches between a series of views. The views switch according to a monotonically increasing number sequence. A unique consensus leader exists within each view. Each replica node maintains a tree structure of pending commands in its memory. Uncommitted branches compete, and only one branch in a round will be agreed upon by the nodes. In the HotStuff protocol, branches are committed as the view number grows. Voting in HotStuff uses the cryptographic term Quorum Certificate (QC), where each view is associated with a QC that indicates whether enough replicas have approved the view. If a branch is confirmed by a replica, it signs the branch, creates a partial certificate [61], and sends to the leader. The leader collects $n-f$ partial certificates, which can be combined into a QC. A view with a QC means that it received the majority votes of the replicas. The leader collects signatures from $n-f$ replicas by using threshold signatures [57], [62]. The process of collecting signatures consists of three phases: **Prepare**, **Pre-prepare**, and **Commit**. Moreover, the entire algorithm consists other two phases: **Decide** and **Finally**, as shown in Figure 6.

(1) **Prepare**. The leader denoted by the current highest view designated as highQC, initiates a proposal for highQC, encapsulates it into a Prepare message with content ⟨Prepare, CurProposal, HighQC⟩, and broadcasts it to all replicas. Replicas will decide whether to accept the proposal or not, and then return a vote with partial signature to the leader if the proposal is accepted. (2) **Pre-commit**. When

the leader receives votes from $n-f$ replicas for the current proposal, it combines them into PrepareQC, encapsulates PrepareQC into a Pre-commit message, and broadcasts it to all replicas. The replica votes after receiving the above proposal message and returns the vote to the leader. (3) **Commit**. When the leader receives the Pre-commit votes from $n-f$ replicas, it merges them into PrecommitQC, encapsulates a PrecommitQC into a Commit message, and broadcasts them to all replicas. The replica votes after receiving the proposal message and returns the Commit vote to the leader. To ensure the safety of the proposal, the replica is locked by setting its LockedQC to PrecommitQC. (4) **Decide**. When the leader receives the Commit votes from $n-f$ replicas, it merges them into one CommitQC and then uses the Decide message to broadcast it to all replicas. After receiving this message, the replica confirms and submits the proposal in the CommitQC, executes the command, and returns it to the client. After this, the replica increases the ViewNumber and starts the next view. (5) **Finally**. If the system moves to the next view, each copy sends a message to the next view's leader with the message ⟨New-view, PrepareQC⟩.

The processes in each phase of Basic HotStuff are very similar to each other, as shown in Figure 6. A modified version of HotStuff, called Chained HotStuff, was proposed [55] to optimize and simplify Basic HotStuff. In the Chained HotStuff protocol, the replicas' votes in the **Prepare** phase are collected by the leader, and stored in the state variable GenericQC. Then, GenericQC is forwarded to the leader of the next view, essentially delegating the next phase's (the **Pre-commit** phase) responsibilities to the next view's leader. Thus, instead of starting its new **Prepare** phase alone, the next view's leader actually executes the **Pre-commit** phase simultaneously. Specifically, the **Prepare** phase of view $v+1$ also acts as the **Pre-commit** phase of view v. The **Prepare** phase of view $v+2$ acts as both the **Pre-commit** phase of view $v+1$ and the **Commit** phase of view v.

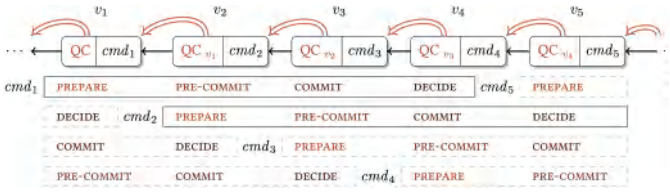

FIGURE 7. Chained HotStuff is a pipelined Basic HotStuff where a QC can serve in different phases simultaneously. [55]

Figure 7 shows that a node can be in different views simultaneously. Through a chained structure, a proposal can reach consensus after three blocks, resembling a Three-Chain shown in figure 8. An internal state converter enables automatic switching of proposals through GenericQC. The chained mechanism in Chained HotStuff reduces cost of communication messages and allows pipelining of processing. In the implementation of Chained HotStuff, if a leader fails in obtaining enough QC, then it may appear that the view numbers of a node are not consecutive. This is solved by adding dummy nodes, as shown in Figure 8, where a dummy node is added to force v_6, itself, and v_8 to form a Three-Chain.

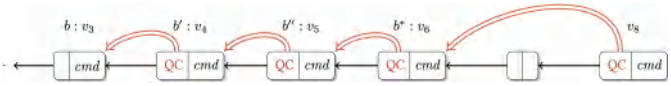

FIGURE 8. The nodes at views v_4, v_5, v_6 form a Three-Chain. The node at view v_8 does not make a valid One-Chain in Chained HotStuff. [55]

HotStuff achieves $\mathcal{O}(n)$ message authentication complexity by improving the distributed consistency algorithm's efficiency using threshold signatures, parallel pipeline processing, and linear view changing. Compared to PBFT, HotStuff can reach consensus pipelining without a complex view-change mechanism and improves consensus efficiency.

5) LibraBFT

As a variant of HotStuff, LibraBFT introduces several changes to meet various business requirements. One of the changes is the introduction of epochs, which allows for consensus node replacement and support for incentive and penalty mechanisms [63]. With the economic incentives and penalties, nodes are encouraged to participate in the voting process and penalized if they violate voting constraints or submit conflicting proposals. Another change is to address the problem of unknown upper bounds on message latency in HotStuff, which only requires partial synchronization [64]. The view-change in HotStuff is not time-bound and relies on the status of the last view, which can result in variable confirmation latencies. To address this, LibraBFT uses pacemaker [63] to ensure that confirmation latency is lower than an upper bound.

C. FEDERATED BYZANTINE AGREEMENT

The algorithms in this category differ from PBFT variants in that a group of nodes can choose one or more nodes as representatives.

1) RPCA

Uses pre-configured validators to vote on transactions for consensus [44]. After several rounds of voting, if a transaction receives a threshold (usually 80%) of votes, it will be recorded in the ledger. Nodes maintain a subset of validators as a list called Unique Node List (UNL). Non-validators, known as tracking servers, forward transaction information and respond to client requests but don't participate in consensus. A validator and tracking server can switch roles, and inactive validators are removed from the UNL.

The process of RPCA is shown in Figure 9. The client initiates a transaction and broadcasts it to the network. Validators receive the transaction data, store it locally, and verify it. Invalid transactions are discarded, while valid transactions are integrated into the candidate set of transactions. Validators periodically send their candidate sets as proposals to other nodes. Once a validator receives a proposal, it checks whether the sender is on the UNL. If not, the proposal is discarded.

Otherwise, the validator stores the proposal locally and compares it with the candidate set. The transaction obtains one vote if it is the same as in the candidate set. If the transaction fails to reach 50% of the votes within a certain period [65], it returns to the candidate set and waits for the next consensus process. If it reaches a threshold denoted by 50% of votes, it enters the next round and is re-sent as a proposal to other nodes, with the threshold raised. As the number of rounds increases, the threshold continues to increase until the transaction reaches 80% or more of the votes, at which point the validator writes it into the ledger.

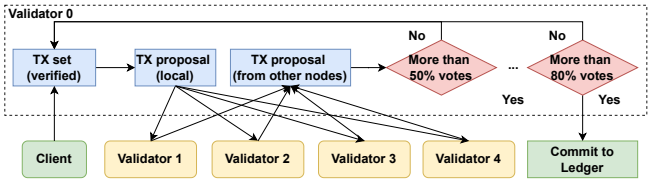

FIGURE 9. Ripple's RPCA Consensus Algorithm

2) Stellar Consensus Protocol (SCP)

A distributed consensus algorithm designed around state machine replication and does not require miners but a distributed server network to run the protocol [45]. SCP is based on the Federated Byzantine Agreement (FBA) and Federated Byzantine Fault Tolerance (FBFT) protocols. The FBA introduces the concept of a quorum slice, which is a subset of nodes that a given node chooses to trust. A quorum is a set, and each non-faulty member of it contains at least one quorum slice. The UNL in RPCA is similar to a quorum slice. In Stellar, the ledger will not update the transaction until 100% of nodes in a quorum slice agree, unlike in Ripple which requires only 80% agreement. There are two mechanisms in the quorum slice model, federated voting and federated leader election. In voting, nodes vote on a statement and use a two-step protocol to confirm it. If each quorum of non-faulty nodes v_1 intersects each quorum of non-faulty nodes v_2 in at least one non-faulty node, then v_1 and v_2 are considered intertwined, and conflicting transactions will not be approved [66]. In leader election, nodes pseudo-randomly select one or a small number of leaders in the quorum slice.

SCP is a global consensus protocol that includes three components: a nomination protocol, a ballot protocol, and a timeout mechanism. The nomination phase proposes new values as candidates for reaching an agreement using the statement **Nominate** x, where x is a valid candidate consensus value. Each node that receives these values votes for a single value from among received ones. The nomination phase generates the same set of candidate values as a deterministic combination of all values on each intact node [66]. After the successful execution of the nomination phase, the nodes enter the ballot phase, which uses federated voting to commit or abort the values. In FBA, as shown in Figure 10, the three-step process involves a node broadcasting a valid statement a and then accepting it if it doesn't conflict with any previously accepted values. If all members of the node's quorum set accept a, it is rebroadcasted. Finally, a is confirmed if each node in the node's quorum accepts it, and the node confirms it as well. However, there may be a **stuck** state since the node cannot conclude whether to abort or commit a value. SCP uses two statements **Prepare** and **Commit**, and a series of numbered ballots, to avoid stuck votes in the federated voting process. A statement **Prepare**$\langle n, x\rangle$ states that no value other than x was or will ever be chosen in any ballot $\leq n$. Another statement **Commit**$\langle n, x\rangle$ states that value x is chosen in ballot n. A node has to confirm the **Prepare**$\langle n, x\rangle$ statement before voting for the **Commit**$\langle n, x\rangle$ statement. Once **Commit** is confirmed, the value x can be output by the node. SCP provides liveness by using these two statements when the node determines a stuck ballot has been committed. The timeout mechanism is a crucial part of SCP. If the current ballot n appears to be stuck, a new round of federated voting begins on a new ballot with a higher counter $n+1$. Unlike the Byzantine agreement, SCP allows participating nodes to determine their quorums, making it a significant difference. SCP utilizing FBA avoids stuck states and enables low latency and flexible trust.

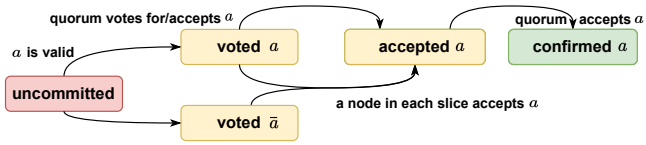

FIGURE 10. Federated voting process

D. RANDOMIZED CONSENSUS MECHANISMS

The algorithms in this category leverage aspects of randomized mythologies to choose nodes that have decisive effects on reaching consensus in the network.

1) PoET

Proof of Elapsed Time [42] is an efficient consensus protocol utilizing a Trusted Execution Environment (TEEs), i.e., Intel SGX-enabled CPUs [67], implemented in Hyperledger Sawtooth Project [52]. Although PoET looks like a competitive consensus algorithm (i.e., PoW) by name, it essentially selects the validator node by a randomized waiting time, in contrast to PoW, where all the nodes compete to solve a cryptographic puzzle and mine the next block. As a result, significantly fewer computational resources are required. In this mechanism, each node waits for a particular random time $\mathcal{T}$ which satisfies a predefined probability distribution function $\mathcal{F}$. The first node to finish the waiting time becomes the primary node. The other nodes verify that the primary node has indeed waited for the particular random time $\mathcal{T}$ in three steps: (a) Proof of waiting time is generated with the assistance of SGX while generating the block on the primary node. (b) The primary node broadcasts the generated proof along with the block to other nodes. The other nodes will produce block and waiting time verification. (c) A probabilistic

statistical verification is applied to check whether the node's waiting time meets the predefined probability distribution $\mathcal{F}$. The PoET consensus algorithm relies on the hardware for trust so that the blockchain system does not need to spend significant computing power and achieves fairness.

2) Ouroboros

Based on the PoS [50], Ouroboros randomly chooses a stakeholder and authorizes the selected stakeholder permission to generate the block in a certain period [32]. Ouroboros divides time into epochs, with each epoch containing multiple slots, and at most, one block will be generated per slot. For each slot, a leader is elected, generating a block and passing it to the next slot leader. The process of the slot leader election is randomized, and it has been proven as an unpredictable process. It isdenoted as $\mathsf{F}(S, p, sl_j) \rightarrow S_i$. Where S is the set of candidate stakeholders, p is the random seed, sl_j denotes the j-th slot in the epoch, and S_i the selected node. The randomized algorithm selects the slot leader from the candidate set S for each slot. During the epoch cycle, S and p remain unchanged until the of the cycle. If a new epoch starts, then a new p is generated along with an updated S. Each node executes the random function F to check who is the slot leader of the current slot based on the seed p of the current epoch. If the slot leader is itself, then the node will combine the transactions and create a new block; otherwise, it waits for the slot leader to create the block, broadcasts it, and verifies the received block. If a node has not received a broadcast for a long time (beyond the time of one slot), it considers that no block is generated for current slot. This process is repeated in current epoch until all slots are finished.

In Ouroboros, the random seed p for each epoch is generated by a secure multi-party computation protocol (MPC), which uses a publicly verifiable secret-sharing algorithm [68] to guarantee the generation of bias-resistant random numbers in the presence of adversaries. Similar to the voting algorithms, the randomly generated leader node reduces resource consumption. However, unlike the voting algorithms, in which the generation of leaders requires network communications between nodes, the randomized algorithm uses a random algorithm to generate leaders without communication, which shortens the running time for reaching consensus, improves efficiency, and enhances scalability.

3) HoneyBadgerBFT (HB-BFT)

HB-BFT [31] achieves distributed consensus consistency in asynchronous networks without requiring any time assumptions, and solves the transaction censorship problem through the use of threshold encryption. In each consensus round, the protocol operates on data of size B, assuming n nodes in the network. The primary process of HB-BFT is as follows: (a) Each node collects transaction data in its own transaction data buffer. Before each round, the node removes the first B/n transactions from the buffer. HB-BFT then divides a block into n copies, each containing different transactions. After that, these copies will be sent to replica nodes. Then, replica nodes will exchange the remaining $n-1$ copies to fully use the bandwidth between replica nodes and ease the broadcast pressure of the primary node. (b) Each replica node randomly selects transactions to verify, sign, and broadcast with threshold encryption to improve transaction throughput. As HB-BFT is an asynchronous consensus protocol, the transactions received by each replica node are asynchronous and random, and their arrival order can vary. After receiving transaction information, each node stores it in a local cache pool and randomly selects several transactions to process. The node then verifies, signs, and broadcasts the selected transactions with threshold encryption using Reliable Broadcast (RBC). The ciphertext data is then generated and broadcasted by the replica. Eventually, each node combines the different subsets of transactions received to form the complete set of transactions contained in the block of that view. (c) The final confirmation of required transactions is achieved through Asynchronous Binary Byzantine agreement (ABA). The primary node forms a transaction set ciphertext from encrypted data received from nodes and initiates ABA consensus. A consensus round determines the final confirmed binary value, which determines which transactions will be approved. Nodes can confirm the Asynchronous Common Subset (ACS) subcomponent after eliminating invalid and duplicate transactions based on the received value V. (d) Transaction data threshold decryption. When the ABA consensus is completed, i.e., the transaction set's ciphertext data is confirmed, the nodes run the threshold decryption algorithm. As long as at least $f+1$ nodes complete the decryption, the plaintext data in the set can be restored, and the transaction is confirmed. There are two advantages to this strategy: first, it prevents adversaries from understanding the transaction selection strategy to interfere or attack; and second, although the order of transactions received by each node may not be consistent, most of them may be in the same order under similar network conditions, and random selection instead of all sequential selection can avoid a large number of duplicates.

4) Dumbo

Proposed as the first practical asynchronous BFT protocol [69] based on HB-BFT and the motivation to address performance bottleneck of HB-BFT due to the number of ABA instances. Because each node has to run N instances of ABA in each round, and each instance has to verify $\mathcal{O}(n^2)$ threshold signatures, HB-BFT consumes a lot of computing resources. To improve the efficiency of asynchronous consensus by reducing the number of ABA instances in each round, Dumbo uses two atomic broadcast protocols: (1) Dumbo1 which reduces the number of instances running ABA from N to k by choosing a committee of k members. By leveraging an added coin-tossing protocol, Dumbo1 can randomly elect a committee of k members, with a high probability that at least one of the k members is honest. (2) Dumbo2, which reduces number of ABA to a constant running time by using the Multi-value Validated Byzantine Agreement (MVBA) [31]. To apply MVBA, a provable reliable broadcast (PRBC) is

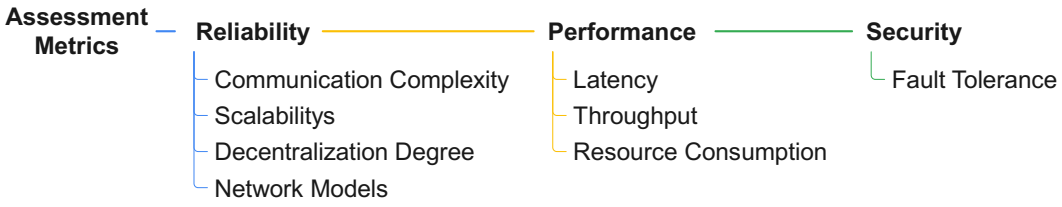

FIGURE 11. Assessment Metrics for Proposed Taxonomy

implemented by levering threshold signature on the RBC index. As a result, only three consecutive ABA instances need to be performed.

E. ASSESSMENT METHODOLOGY

Selected consensus algorithms are compared along the following three dimensions that form the backbone of scalable solutions in blockchain technology: (1) Reliability is measured by communication complexity, scalability, and decentralization. (2) Performance is measured by latency, throughput, and resource consumption. (3) Security is measured by fault tolerance. Figure 11 shows comparison methodologies for proposed taxonomy. The Assessment for selected algorithms will be presented in section IV, V, and VI.

IV. RELIABILITY

This section analyses the network reliability of consensus algorithms along the metrics of communication complexity, scalability, decentralization degree, and network models. Table 2 shows the comparison of above metrics in selected consensus algorithms.

Complexity refers to the number of messages required to reach a round of consensus. The smaller the complexity, the more efficient is the consensus algorithm. The number of messages in a normal case can differ from a leader failure situation. Table 2 presents the communication complexity of different protocols in normal situations and situations in which the leader fails. A message's communication complexity affects the network's scalability to some extent. If the complexity is higher, the nodes need more messages to accomplish consensus in one round. Therefore, algorithms in networks that accommodate more nodes tend to have lower complexity. In most consensus algorithms, the complexity is higher when a network leader fails since all participating nodes may require more rounds to have a new leader. The best case of complexity is linear for both normal and leader failure cases. In the category of PBFT and derivatives, PBFT's message complexity is $\mathcal{O}(n^2)$ when a non-malicious primary node operates without failure. Alternatively, it increases to $\mathcal{O}(n^3)$ if the primary node fails (processing view-change protocol). Since the leader is malicious, the protocol replaces it with a new leader through a view-change that includes at least $2f + 1$ signed messages. A new leader then broadcasts a new-view message containing proof of $2f + 1$ signed view-change messages. Validators will examine the new view-change message and broadcast it if it matches $2f + 1$ view-change messages. Overall, the view-change complexity is $\mathcal{O}(n^3)$ while the leader fails. In this category, HotStuff can reduce the complexity to $\mathcal{O}(n)$ and guarantee responsiveness by using threshold signatures, three rounds of voting, and a chained structure to acknowledge a block [55]. HotStuff can reach consensus pipelining without a complex view-change mechanism and improves efficiency. In the category of FBA, the complexity is determined by number of total nodes n and number of nodes K that formalized the federation. In normal case, complexity is $\mathcal{O}(nK)$. Upper bound is $\mathcal{O}(n^2)$ if and only if K is close to n.

Scalability refers to the number of peering nodes that the algorithm can process and implies an upper bound on the size of the network. If a consensus protocol can support over 100 participants without losing network reliability, we conclude its scalability is **High**. A range of 20 to 100 is considered **Medium**, and anything lesser than 20 is considered to be **Low** suitability for scalability. Regardless of the message size and network environment, the scalability of a consortium blockchain network is also determined by the communication complexity. It can increase by reducing complexity. Therefore, a consortium blockchain atop a consensus algorithm with a lower complexity is always more scalable than a higher one. In the category of Paxos and derivatives, Paxos provides medium scalability, while Raft, its upgraded derivative, can support a relatively large network with high scalability. In the category of PBFT and derivatives, the total number of broadcast messages grows quadratically with the increase in the total number of nodes, leading to rapid super-linear performance degradation. Therefore, PBFT is only suitable for consortium blockchain and private blockchain with low scalability. In the category of FBA, SCP emphasizes maintaining the network's activity and allows any node to join each other's trust list for transactions if it follows the policy. With SCP, the Stellar network can run approximately 100 nodes [70].

Decentralization implies that the existence of a relatively neutral entity functioning as a central node. In a round of reaching consensus, the node which decides the recording of transactions on the distributed ledger is considered as the central node [15]. All other nodes keep the data consistent around it. In order to maintain the distributed state of the system, the role of each node (including central node) is subject to change. Therefore, we compare the degree of de-

TABLE 2. Consensus Algorithm Comparison

Category →	Paxos Derivatives		PBFT Derivatives					FBA		Randomized			
Metrics \ Protocol	Paxos [35]	Raft [33]	PBFT [34]	RBFT [40]	BFT-SMART [46]	SBFT [71]	HotStuff [55]	RPCA [44]	SCP [45]	PoET [42]	Ouroboros [32]	HB-BFT [31]	Dumbo [69]
Complexity Nomal	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(nK)$	$\mathcal{O}(nK)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
Complexity Leader Failed†	-	-	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(nK)$	$\mathcal{O}(nK)$	-	-	-	-
Scalability	◉	●	○	○	◉	◉	●	●	●	●	●	●	●
Decentralization	●	●	○	◉	●	●	●	○	●	●	●	●	●
Network Model	P	P	P	P	P	P	P	P	P	S	S	A	A
Latency	○	○	○	○	○	○	○	◉	◉	○	◉	○	○
Throughput	◉	◉	○	●	●	●	●	◉	○	○	○	●	●
Resource Consumption	○	○	◉	◉	◉	○	○	○	○	○	○	○	○
Crash Fault Tolerance	50%	50%	33%	33%	33%	33%	33%	20%	33%	50%	50%	33%	33%
Byzantine Fault Tolerance	-	-	33%	33%	33%	33%	33%	20%	33%	-	-	33%	33%

● = high; ◉ = medium; ○ = low; S = Synchronous; P = Partially Synchronous; A = Asynchronous;
† Leader Failed case is only suitable for PBFT derivatives and FBA.

centralization of the algorithm according to the recording node's selection rules and the number of selected recording nodes in each round. If all nodes are competitive while participating in consensus reaching and the probability of each node becoming the primary node is equal to each other, the degree of decentralization is defined as High. For instance, in Raft, every node has an equal chance of being elected by submitting a proposal and getting the votes from other nodes. If more than one node is selected as primary nodes or some nodes have higher priorities than other nodes, the degree of decentralization is assumed as Medium. The Low degree of decentralization only exists in cases where the primary nodes are pre-selected. i.e., the verification nodes in RPCA are pre–configured in the entire network. In a consortium blockchain, the degree of decentralization is flexible. It is unnecessary to require all nodes to have equal permission to participate in consensus reaching. To maintain the reliability of a consortium blockchain, some consensus algorithms use a low degree of decentralization as a tradeoff to reduce the number of validator nodes and seek a higher communication efficiency. Alternatively, the degree of decentralization can be decided by committee members as a business strategy.

The **network model** is an important metric that defines the ability of different message latencies to limit the blockchain network reliability. In general, there are three types of network models: (1) **Synchronous**: There is a known upper bound Δ on the latency of message communication between all nodes. The synchronous model offers an ideal communication pattern, which is hardly visible in real life but plays an important role in the theoretical study of distributed systems, and many early distributed consistency algorithms were designed under the assumption of synchronous networks. (2) **Asynchronous**: The above-mentioned upper bounds Δ does not exist, so the asynchronous model is more in line with the realistic Internet environment. Asynchronous is a more general case than synchronous. An algorithm that works for an asynchronous system can also be used for a synchronous system, but the converse does not hold. (3) **Partially Synchronous**: This model offers a communication pattern between the synchronous model and asynchronous model [64]. In the partially synchronous model, it is assumed that there is a globally stable clock GST (Global Stabilization Time), the message arrives in Δ time and the entire system may be in the asynchronous state before GST, but after GST, the whole system can return to a synchronous state. The timing assumption of the partial synchronization model is more in line with the real-world need for consensus algorithms, i.e., consensus can always be reached in a synchronized state; but once the network goes down, consensus may enter a period of blocking until it returns to normal. In addition, based on the relationships of the network models (synchronous $\subseteq$ partially synchronous $\subseteq$ asynchronous), if a consensus fully supports an asynchronous network, it theoretically supports synchronous and partially synchronous networks. Thus, a fully asynchronous consensus protocol provides high reliability.

V. PERFORMANCE

This section provides an assessment of selected consensus algorithms regarding performance efficiency. In a consortium blockchain, domain services and transaction types vary. This creates a need for an efficient backbone for applications with low latency, high throughput, and low resource consumption. Table 2 shows evaluation results.

Latency is defined as the time elapsed from the moment a node submits a transaction to the time that the transaction is confirmed by blockchain. High latency means longer confirmation times for transactions, which can be an issue for applications that require faster processing times. On the other hand, low latency can provide faster confirmation times and

improve the user experience. Thus, it is critical to consider the latency of a consensus algorithm when evaluating performance of blockchain. In this study, latency is classified as High, Medium, or Low [16]. High latency is in the magnitude of minutes, Medium is in seconds, and Low is in milliseconds.

Throughput refers to the block generation rate and the number of Transactions Per Second (TPS) the system can process. Block generation is expressed as time required for the entire process starting from the time when transactions are packaged into blocks up to the time when consensus is reached. TPS is determined by the size of block and block generation speed. TPS is calculated as the number of transactions in the block divided by the length of time required for the generation of the current block. We classify throughput into three categories. A protocol can provide $>$ 2,000 TPS is classified as a High throughput protocol. A TPS between 1,500 to 2,000 indicates Medium throughput, and a TPS $<$ 1,500 indicates a Low throughput. In the category of PBFT and derivatives, a blockchain implementation with BFT-SMART protocol by Symbiont can reach a throughput of 8000 TPS in a 4-node network cluster, which meets the expected performance of the original paper [46]. In the category of FBA, the advantage of RPCA algorithm is its relatively high performance and efficiency. Ripple can generate a block every three seconds with a throughput that reaches 1500 TPS.

Resource consumption refers to the computing power, memory, input and output, and energy resources that each node consumes while reaching a consensus. Communication complexity is a theoretical proxy of resource consumption. Resource consumption is classified as High, Medium, or Low [16]. High exists in some consensus algorithms that exhaust large resources for competing for a leader. This category only exists in public blockchains, for instance, when a node runs PoW it competes for a hash computation to get the transactions to be committed, including using a graphic card to acquire a high computing ability and thus is a drain on resources. Consensus algorithms defined as Medium in terms of their resource consumption usually are not a drain on resources when competing as a leader, but still need some due to a high communication complexity. Low resource consumption algorithms are ones that have an upper bound on communication complexity in any condition and do not require extensive computation to compete for a leader.

VI. SECURITY

This section provides an assessment of consensus algorithms regarding system security elaborated by fault tolerance. In terms of security, public blockchains face a high diversity of attack vectors. For instance, there are various attacks towards PoW, such as 51% attack, eclipse attack, dust attack, empty block attack, selfish mining attack, block withholding attack, etc. [72]–[79].However, the consensus protocols adopted in the consortium blockchains are more fault-tolerant and more controllable since they can detect and tolerate a certain number of malicious nodes. Therefore, consortium blockchains face relatively fewer security attacks and relatively low attack diversity. Therefore, only the impact of fault tolerance will be discussed in this section. The ability of a blockchain system to maintain uninterrupted operations when one or more of its components fail is intrinsic to its nature. Table 2 presents a comparison of selected consensus algorithms.

Fault Tolerance is essential to security, and denotes the capacity of a distributed network to minimize the severity and frequency of network incidents, continue operations under stress, and recover as quickly as possible. We define security as the tolerance of the distributed network to malicious attacks or the amount of Byzantine behavior that the network can resist. Robustness is the ability of the network to maintain its performance in face of failures, or changes in topology or load. We consider two types of failures: Byzantine failures are those where some network participants may exhibit malicious behavior, such as sending conflicting messages, while crash failures are some participants are outage or unreachable.

In the category of Paxos and derivatives, Paxos enables a distributed system to reach consensus when the number of non-failure nodes is greater than half of the total nodes. Raft was inspired by Paxos and its fault tolerance is very similar to Paxos. Neither Paxos nor Raft provide Byzantine fault tolerance. In the category of PBFT and derivatives, PBFT can tolerate both non-Byzantine errors and Byzantine errors, simultaneously, by sending broadcasts to the entire network in each round and allowing each node to participate in electing the primary node. This advanced mechanism ensures that PBFT has the capabilities to maintain consistency, ensure availability, and enable anti-fraud. RBFT was proposed for better resiliency during the process of Byzantine fault tolerance. In earlier BFT algorithms such as PBFT, Prime [80], Aardvark [81], and Spinning [82], if the primary node is malicious, the whole system's performance is degraded. The RBFT model proposes executing multiple PBFT protocol instances in parallel using multi-core machines, which can easily detect malicious nodes and prevent performance degradation. If one or more Byzantine faulty nodes exist in the blockchain network, it has been shown that the maximum performance degradation of RBFT is 3%, which is better than other protocols; for instance, Prime is 80%, Aardvark is 87%, and Spinning is 99% [40]. HotStuff's responsiveness enables nodes to quickly confirm blocks under normal network conditions and can wait longer to confirm under limited network conditions. Overall, the algorithms in this category have the same level of fault tolerance on both crash and Byzantine faults. In the category of FBA, the fault tolerance of SCP is the same lever as PBFT family. However, the fault tolerance of RPCA is lower than that of SCP and PBFT-like algorithms since the verification node is pre-configured, and the fault tolerance is bound to the number of verification nodes. In the category of Randomized, PoET and Ouroboros provide the same ability to tolerate crash faults. HB-BFT and Dumbo have the same level of fault tolerance with PBFT and derivatives on both crash and Byzantine faults.

VII. CONCLUDING REMARKS

Our intent for this paper was to carry out groundwork that informs researchers, developers, and the blockchain community at large of the current landscape of consensus methodologies and technologies and remaining challenges. The choice of a consensus algorithm has an enormous impact on the performance of a blockchain application. Therefore, ongoing research into the design and implementation of consensus algorithms will advance and facilitate the adoption of blockchain for diverse applications. Regardless of the type of blockchain used and its applications, a consensus algorithm is pivotal to the blockchain operation and must therefore be carefully designed. The primary research challenges that need to be addressed in the consensus mechanism domain for consortium blockchain are: (1) **Scalability enhancement**: While the consortium blockchain offers limited membership, the issue of scalability in a consortium blockchain is critical. As we discussed, the size of a network has implications for parameters such as fault tolerance that impact the blockchain's efficiency. As business needs grow, the number of access nodes required by the platform may increase to keep pace with the platform's expansion. Proactive approaches to building consortium blockchains that adapt to changing business and platform expansion needs must be considered to strengthen scalability. (2) **Algorithmic melding**: As applications and platforms evolve, consensus algorithms may require more flexibility in adapting to changing environments. The evolution of applications and platforms may introduce logical, but intricate, requirements for fusion between algorithms. Therefore, integrating different types of consensus mechanism algorithms in the future poses a distinct challenge to interoperability. (3) **Privacy-preservation**: The consortium blockchain needs authentication for participating nodes, which reduces the probability of possible attacks, to a certain extent. Nevertheless, we still need to consider the security and privacy of data on the consortium chain. The use of cryptography to ensure security and privacy of data on the blockchain while still conforming to the central paradigm of blockchain decentralization will be a tradeoff to consider. (4) **Performance improvement**: The potential to further several performance factors, such as increase in throughput, reduction in latency, and reduction in computational requirements for consensus algorithms must be considered. Each of these factors impacts the scalability of the blockchain. Therefore, ensuring increasing performance while reducing the impact on scalability is a challenge. (5) **Searching and storing optimization**: While the original philosophy of blockchain called for implementations to build a distributed ledger, the expectations for blockchain networks have evolved into data retrieval over the years. In this use scenario, a blockchain ledger is more like a distributed database without the capability of deleting and updating operations due to the immutability property of blockchain. Therefore, the consensus mechanisms that are built for blockchain should also consider whether the data storing and searching can be optimized accordingly. These challenges broadly identify the various areas of improvement for consortium blockchain algorithms. However, since these protocols are still under development and the applications leveraging these algorithms are continuously being refined, the scope of challenges for consensus algorithms in consortium blockchain will continue to be a work in progress.

ACKNOWLEDGMENT

The research is partially supported by FHWA EAR 693JJ320C000021.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Cryptography Mailing list at https://metzdowd.com*, Mar. 2009.
[2] M. Swan, *Blockchain: blueprint for a new economy*. O'Reilly, first edition ed., 2015.
[3] M. Dotan, Y.-A. Pignolet, S. Schmid, S. Tochner, and A. Zohar, "Sok: cryptocurrency networking context, state-of-the-art, challenges," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pp. 1–13, 2020.
[4] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.
[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, 2018.
[6] HyperLedger, "Case study: How walmart brought unprecedented transparency to the food supply chain with hyperledger fabric," Mar 2019.
[7] HyperLedger, "Case study: How culedger protects credit unions against fraud with hyperledger indy," 2020.
[8] HyperLedger, "When hyperledger sawtooth met kubernetes - simplifying enterprise blockchain adoption," 2020.
[9] Libra Association Members, "Libra White Paper | Blockchain, Association, Reserve," Apr. 2020.
[10] H. Rathore, A. Mohamed, and M. Guizani, "A survey of blockchain enabled cyber-physical systems," *Sensors*, vol. 20, no. 1, p. 282, 2020.
[11] S. T. Aras and V. Kulkarni, "Blockchain and its applications–a detailed survey," *International Journal of Computer Applications*, vol. 180, no. 3, pp. 29–35, 2017.
[12] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics and informatics*, vol. 36, pp. 55–81, 2019.
[13] Y. Zou, T. Meng, P. Zhang, W. Zhang, and H. Li, "Focus on blockchain: A comprehensive survey on academic and application," *IEEE Access*, vol. 8, pp. 187182–187201, 2020.
[14] O. Dib, K.-L. Brousmiche, A. Durand, E. Thea, and E. B. Hamida, "Consortium blockchains: Overview, applications and challenges," *International Journal On Advances in Telecommunications*, vol. 11, no. 1&2, pp. 51–64, 2018.
[15] G.-T. Nguyen and K. Kim, "A survey about consensus algorithms used in blockchain," *Journal of Information processing systems*, vol. 14, no. 1, pp. 101–128, 2018.
[16] M. Salimitari and M. Chatterjee, "A survey on consensus protocols in blockchain for iot networks," *arXiv preprint arXiv:1809.05613*, 2018.
[17] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.
[18] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," *ArXiv*, vol. abs/1707.01873, 2017.
[19] M. S. Ferdous, M. Chowdhury, M. A. Hoque, and A. Colman, "Blockchain consensus algorithms: A survey," *arXiv: Distributed, Parallel, and Cluster Computing*, 2020.
[20] B. Bellaj, A. Ouaddah, E. Bertin, N. Crespi, and A. Mezrioui, "Sok: a comprehensive survey on distributed ledger technologies," in *ICBC 2022: IEEE International Conference on Blockchain and Cryptocurrency*, pp. 1–16, IEEE, 2022.
[21] J. B. Bernabe, J. L. Canovas, J. L. Hernandez-Ramos, R. T. Moreno, and A. Skarmeta, "Privacy-preserving solutions for blockchain: Review and challenges," *IEEE Access*, vol. 7, pp. 164908–164940, 2019.

[22] Y. Mao, S. Deb, S. B. Venkatakrishnan, S. Kannan, and K. Srinivasan, "Perigee: Efficient peer-to-peer network design for blockchains," in *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pp. 428–437, 2020.

[23] H.-Y. Paik, X. Xu, H. D. Bandara, S. U. Lee, and S. K. Lo, "Analysis of data management in blockchain-based systems: From architecture to governance," *Ieee Access*, vol. 7, pp. 186091–186107, 2019.

[24] M. Szydlo, "Merkle tree traversal in log space and time," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 541–554, Springer, 2004.

[25] H. L. Pham, T. H. Tran, T. D. Phan, V. T. D. Le, D. K. Lam, and Y. Nakashima, "Double sha-256 hardware architecture with compact message expander for bitcoin mining," *IEEE Access*, vol. 8, pp. 139634–139646, 2020.

[26] H. Vranken, "Sustainability of bitcoin and blockchains," *Current opinion in environmental sustainability*, vol. 28, pp. 1–9, 2017.

[27] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," *arXiv preprint arXiv:1906.11078*, 2019.

[28] C. Dods, N. P. Smart, and M. Stam, "Hash based digital signature schemes," in *Cryptography and Coding* (N. P. Smart, ed.), p. 96–115, Springer Berlin Heidelberg, 2005.

[29] S. Chandra, S. Paira, S. S. Alam, and G. Sanyal, "A comparative survey of symmetric and asymmetric key cryptography," in *2014 international conference on electronics, communication and computational engineering (ICECCE)*, pp. 83–93, IEEE, 2014.

[30] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[31] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 31–42, 2016.

[32] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual international cryptology conference*, pp. 357–388, Springer, 2017.

[33] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference*, 2014.

[34] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance," in *OsDI*, vol. 99, pp. 173–186, 1999.

[35] L. Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pp. 51–58, 2001.

[36] L. Baird, B. Gross, and T. Donald, "Hedera consensus service," *Hedera Hashgraph*, 2020.

[37] R. Saltini and D. Hyland-Wood, "Ibft 2.0: A safe and live variation of the ibft blockchain consensus protocol for eventually synchronous networks," *arXiv preprint arXiv:1909.10194*, 2019.

[38] E. Buchman, *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.

[39] J. Kreps, N. Narkhede, J. Rao, *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, pp. 1–7, 2011.

[40] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "Rbft: Redundant byzantine fault tolerance," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pp. 297–306, IEEE, 2013.

[41] F. Muratov, A. Lebedev, N. Iushkevich, B. Nasrulin, and M. Takemiya, "Yac: Bft consensus algorithm for blockchain," *arXiv preprint arXiv:1809.00554*, 2018.

[42] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (poet)," in *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pp. 282–297, Springer, 2017.

[43] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "Dbft: Efficient leaderless byzantine consensus and its application to blockchains," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8, IEEE, 2018.

[44] D. Schwartz, N. Youngs, A. Britto, *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, no. 8, p. 151, 2014.

[45] D. Mazières, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Development Foundation*, vol. 32, pp. 1–45, 2015.

[46] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with bft-smart," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 355–362, IEEE, 2014.

[47] S. D. Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain," in *Italian Conference on Cybersecurity*, 2018.

[48] L. Lamport, R. E. Shostak, and M. C. Pease, "The byzantine generals problem," in *Concurrency: the Works of Leslie Lamport*, pp. 203–226, ACM, 2019.

[49] Wikipedia, "Adversary (cryptography)," Dec 2020. Page Version ID: 995747901.

[50] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, no. 1, 2012.

[51] V. Foundation, "Vechain whitepaper," Dec 2019.

[52] K. Olson, M. Bowman, J. Mitchell, S. Amundson, D. Middleton, and C. Montgomery, "Sawtooth: an introduction," *The Linux Foundation*, 2018.

[53] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Annual Cryptology Conference*, pp. 585–605, Springer, 2015.

[54] J. Kwon, "Tendermint: Consensus without mining," *Draft v. 0.6, fall*, vol. 1, no. 11, 2014.

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

[56] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pp. 245–256, IEEE, 2011.

[57] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: a scalable and decentralized trust infrastructure," in *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pp. 568–580, IEEE, 2019.

[58] E. Alchieri, F. Dotti, O. M. Mendizabal, and F. Pedone, "Reconfiguring parallel state machine replication," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 104–113, IEEE, 2017.

[59] J. Sousa and A. Bessani, "From byzantine consensus to bft state machine replication: A latency-optimal transformation," in *2012 Ninth European Dependable Computing Conference*, pp. 37–48, IEEE, 2012.

[60] C. Cachin, "Yet another visit to paxos," *IBM Research, Zurich, Switzerland, Tech. Rep. RZ3754*, 2009.

[61] T.-H. H. Chan, R. Pass, and E. Shi, "Pala: A simple partially synchronous blockchain.," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 981, 2018.

[62] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," *ArXiv*, vol. abs/1707.01873, 2017.

[63] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," *The Libra Assn., Tech. Rep*, 2019.

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

[65] B. Chase and E. MacBrough, "Analysis of the xrp ledger consensus protocol," *arXiv preprint arXiv:1802.07242*, 2018.

[66] D. Mazières, G. Losa, and E. Gafni, "Simplified scp," 2019.

[67] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[68] M. Stadler, "Publicly verifiable secret sharing," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 190–199, Springer, 1996.

[69] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous bft protocols," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 803–818, 2020.

[70] C. Berger and H. P. Reiser, "Scaling byzantine consensus: A broad analysis," in *Proceedings of the 2nd workshop on scalable and resilient infrastructures for distributed ledgers*, pp. 13–18, 2018.

[71] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: a scalable and decentralized trust infrastructure," in *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pp. 568–580, IEEE, 2019.

[72] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1–34, 2019.

[73] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and D. Mohaisen, "Exploring the attack surface of blockchain: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1977–2008, 2020.

[74] N. Anita and M. Vijayalakshmi, "Blockchain security attack: a brief survey," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–6, IEEE, 2019.
[75] C. Ye, G. Li, H. Cai, Y. Gu, and A. Fukuda, "Analysis of security in blockchain: Case study in 51%-attack detecting," in *2018 5th International conference on dependable systems and their applications (DSA)*, pp. 15–24, IEEE, 2018.
[76] D. K. Tosh, S. Shetty, X. Liang, C. A. Kamhoua, K. A. Kwiat, and L. Njilla, "Security implications of blockchain cloud with analysis of block withholding attack," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 458–467, IEEE, 2017.
[77] S. Sayeed and H. Marco-Gisbert, "Assessing blockchain consensus and security mechanisms against the 51% attack," *Applied sciences*, vol. 9, no. 9, p. 1788, 2019.
[78] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.
[79] R. C. Lunardi, R. A. Michelin, H. C. Nunes, C. V. Neu, A. F. Zorzo, and S. S. Kanhere, "Consensus algorithms on appendable-block blockchains: impact and security analysis," *Mobile Networks and Applications*, vol. 27, no. 4, pp. 1408–1420, 2022.
[80] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *IEEE transactions on dependable and secure computing*, vol. 8, no. 4, pp. 564–577, 2010.
[81] A. Clement, E. Wong, L. Alvisi, M. Dahlin, M. Marchetti, *et al.*, "Making byzantine fault tolerant systems tolerate byzantine faults," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, The USENIX Association, 2009.
[82] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "Spin one's wheels? byzantine fault tolerance with a spinning primary," in *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pp. 135–144, IEEE, 2009.