SoK: Scalability Techniques for BFT Consensus

Christian Berger*†, Signe Schwarz-Rüsch*‡, Arne Vogel*‡,
Kai Bleeke§, Leander Jehl¶, Hans P. Reiser‖, and Rüdiger Kapitza‡

* The first three authors contributed equally to this work
†University of Passau, Passau, Germany, Email: cb@sec.uni-passau.de
‡ Friedrich-Alexander-Universtität Erlangen-Nürnberg, Erlangen, Germany, Email: {ruesch, vogel, kapitza}@cs.fau.de
§ Technische Universität Braunschweig, Braunschweig, Germany, Email: bleeke@ibr.cs.tu-bs.de
¶ University of Stavanger, Stavanger, Norway, Email: leander.jehl@uis.no
‖ Reykjavik University, Reykjavik, Iceland, Email: hansr@ru.is

***Abstract*—With the advancement of blockchain systems, many recent research works have proposed distributed ledger technology (DLT) that employs Byzantine fault-tolerant (BFT) consensus protocols to decide which block to append next to the ledger. Notably, BFT consensus can offer high performance, energy efficiency, and provable correctness properties, and it is thus considered a promising building block for creating highly resilient and performant blockchain infrastructures. Yet, a major ongoing challenge is to make BFT consensus applicable to large-scale environments. A large body of recent work addresses this challenge by developing novel ideas to improve the scalability of BFT consensus, thus opening the path for a new generation of BFT protocols tailored to the needs of blockchain. In this survey, we create a systematization of knowledge about the novel scalability-enhancing techniques that state-of-the-art BFT consensus protocols use. For our comparison, we closely analyze the efforts, assumptions, and trade-offs these protocols make.**

***Index Terms*—Byzantine Fault Tolerance, Consensus, Scalability, Blockchain**

I. Introduction

Blockchain-based distributed ledger technology (DLT) experienced increasing adaptation and growing popularity in recent years. The original Bitcoin protocol uses a Proof-of-Work (PoW) scheme to achieve agreement on blocks of transactions, also called Nakamoto consensus [1]. Bitcoin and many of its more recently upcoming competitors aim to realize secure and decentralized applications. While Bitcoin's application is to allow its users to send and receive digital peer-to-peer cash, other blockchains, such as Ethereum [2], even allow building more complex applications by submitting generic code (called smart contracts) to the blockchain, where functions of smart contracts can be invoked by users sending transactions. A blockchain's security and decentralization depend on a large number of network participants. At the same time, the system performance, i. e., the throughput, should suffice to match the application's requirements.

a) Agreement for Blockchain Networks: Blockchains use an *agreement protocol* to decide which block they append next to the ledger. Ideally, agreement should work on a large-scale (meaning *many* nodes) and geographically dispersed environment. PoW achieved agreement and successfully allowed open membership by securing the blockchain network against Sybil attacks [3]. This is because PoW couples the probability of a node being allowed to decide the next block towards the computational resources it utilized over a certain time span. Because PoW achieves agreement *without coordination between the nodes* other than disseminating the decided blocks, it can scale well for a large number of nodes. As of this writing, there are over 15,000 reachable Bitcoin nodes[1].

Nevertheless, PoW comes with inherent design problems. In particular, it (1) wastes energy and computing resources, (2) usually does not scale up its performance when utilizing more resources, thus making the scheme very inefficient, and (3) it does not guarantee consensus finality [4], a property that ensures a block, once decided, is never changed later on.

b) Coordination-based Byzantine Fault Tolerant (BFT) Agreement: Recent research papers try to work around these problems by proposing to utilize *coordination-based BFT agreement protocols* (which we will simply refer to as BFT protocols as of now) used in well-conceived protocols like PBFT [5], initially proposed more than 20 years ago. The benefits of PBFT and related protocols like BFT-SMaRt [6] lie in proven protocol properties and various performance optimizations allowing these protocols to achieve up to the magnitude of 10^5 transactions per second.

In particular, BFT algorithms can be used as a Proof-of-Stake (PoS) variant [7], in which blockchain nodes are granted permission to participate in the agreement, depending on the stake (i. e., native cryptocurrency of the blockchain) they own. In coordination-based BFT protocols, the decision about which block is being appended to the blockchain is canonical among all correct nodes [7]; thus, they ensure consensus finality, and they are at the same time energy-efficient, i. e., energy is only consumed for meaningful computations.

A. Motivation

a) The Scalability Challenge for BFT Agreement: Traditional BFT protocols have a problem with scaling to large system sizes. To illustrate this with a brief example, let us

978-8-3503-1019-1/23/$31.00 ©2023 IEEE

[1]https://bitnodes.io/

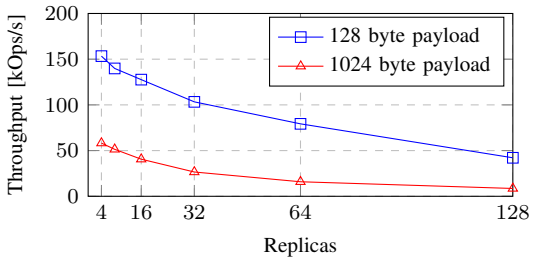


Fig. 1: Performance of BFT-SMaRt (measured by [8]).

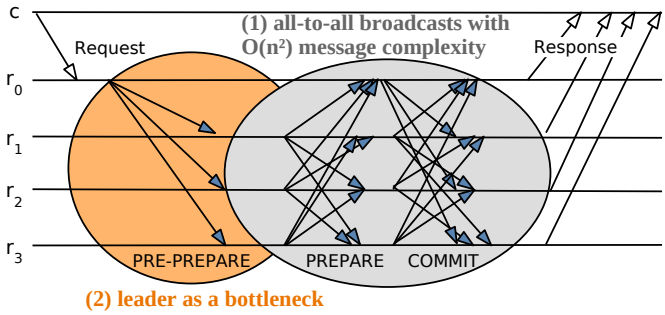


Fig. 2: Scalability problems of PBFT and BFT-SMaRt.

review the well-known PBFT protocol, which is efficient (with performance in the magnitude of 10^5 transactions per second) for small-sized systems but suffers a noticeable performance decrease (of down to a magnitude of 10^3) when several hundred nodes are in the system. Fig. 1 shows the declining performance of a newer protocol, BFT-SMaRt (with the same agreement pattern as PBFT), when the system size increases.

PBFT and BFT-SMaRt share two main problems (see Fig. 2): First, the *normal operation* message complexity is $O(n^2)$ because all replicas exchange their votes using all-to-all broadcasts. This makes the underlying communication topology of the protocol essentially a clique. Further, the more nodes are in the system, the more resources are consumed for only verifying message authenticators from all other nodes.

Second, the protocol flow seems imbalanced: An additional burden is being put on the leader because the leader needs to receive transactions from all clients and then disseminate them in a batch to all other replicas. This essentially makes the leader's bandwidth or its computational capabilities to produce message authenticators a limiting factor for the performance of the BFT protocol since all transactions of the system need to be channeled through a single leader.

b) The Age of Novel BFT Blockchains: Research on BFT consensus becomes increasingly necessary and practical. Recently, many BFT protocols have been proposed for usage in blockchain infrastructures, such as HotStuff [9], SBFT [10], Tendermint [11], [12], Algorand [13], Avalanche [14], Mir-BFT [15], RedBellyBC [16], and Kauri [17].

These protocols aim at making BFT consensus more scalable, thus delivering high throughput at low latency in systems with hundreds or even thousands of participants. Scalability is essential to allow a blockchain to grow its ecosystem and satisfy the demands of many decentralized finance (DeFi) applications.

For instance, as of the time of writing, the Avalanche mainnet consists of 1294 validators[2] and has a total value locked of 2.78 billion USD[3]. But which techniques does the blockchain generation of BFT protocols employ to improve its scalability over well-known BFT protocols such as PBFT? In this survey, we strive to explore the vast design space of novel protocols and analyze their ideas for scaling Byzantine consensus.

c) A Closer Look on Scalability-Enhancing Techniques: As shown in Fig. 2, traditional BFT protocols do not scale well. One way to improve the scalability of BFT protocols is to optimize the protocol logic in a way that (1) reduces bottleneck situations and distributes the transaction load as evenly as possible among the available capacities of every replica and (2) utilizes clever aggregation techniques to reduce the overall message (or: authenticator) complexity in the system.

Many more abstract ideas have been developed to make consensus more scalable – these ideas concern, for example, optimizing communication flow, parallelizing consensuses (i. e., *sharding*), utilizing special cryptographic primitives, or using trusted hardware components.

B. Research Questions

The newer "blockchain generation" of BFT protocols requires scalable Byzantine consensus, which made exploring, advancing, and combining several scalability-enhancing techniques a broad and ongoing research field. In this paper, we want to create a systematization of knowledge on these ongoing efforts, which leads to our two main research questions:

R1 Which novel techniques exist for scaling Byzantine consensus?

R2 How do recent research papers combine existing scalability-enhancing techniques or ideas in a novel way to achieve better scalability than traditional BFT protocols?

Scalability here relates to the number of nodes in the system. We analyze which techniques can improve scalability in BFT protocols, e. g., compared to traditional protocols like PBFT, and also cover the selection of smaller committees as a scalability technique. Surveying the mechanisms of how to provide open membership is not the focus of this work, and we do not focus on defenses against Sybils in open membership systems, e. g., computing a proof of work or depositing stake.

C. Contributions

The main ambitions of our survey paper are to review state-of-the-art research papers on BFT protocols to identify and classify the scalability-enhancing techniques that have been developed. We further investigate the assumptions, ambitions, and trade-offs with which these techniques are used. In particular, our main contributions are the following:

- We conduct a systematic search for exploring scalability-enhancing techniques for BFT consensus.

[2]https://explorer-xp.avax.network/validators
[3]https://defillama.com/chain/Avalanche

- Moreover, we create a taxonomy that classifies and summarizes all of the found techniques.
- Further, we also create a comparison of the new generation of scalable BFT protocol designs.
- We comprehensively discuss the different ideas on an abstract level and pinpoint the design space from which these BFT protocols originate.

D. Outline

In §II, we first give an overview of the basics of BFT protocols. Next, we present our methodology in §III, which is based on a systematic literature search. Further, in §IV, we present our survey on scalability-enhancing techniques for Byzantine consensus, trying to answer the research questions from above. After that, we summarize the efforts of related surveys in §V and conclude in §VI.

II. BFT IN A NUTSHELL

In this section, we review a few basics of BFT protocols.

A. Assumptions

1) The Byzantine Fault Model: A faulty process may behave arbitrarily in the Byzantine fault model, even exhibiting malicious and colluding behavior with other Byzantine processes. The model always assumes that only a threshold f out of n participants is Byzantine, while all others $(n - f)$ are correct and show behavior that exactly matches the protocol description. Although described as arbitrary, the behavior and possibilities of a Byzantine process are still limited by its resources and computational feasibility, e.g., it can not break strong cryptographic primitives. Lamport showed with the Byzantine generals' problem that achieving consensus with f Byzantine participants is impossible if $f \geq n/3$ and solvable for $f < n/3$ in the partially synchronous or asynchronous model (c. f., §II-A2) [18]–[20]. An adversary is often modeled as being in control over all f Byzantine processes. The adversary's access to information can be *limited* through private channels between replicas or *unlimited* if the adversary is modeled to have full disclosure on all messages sent over the network. Lastly, the adversary is assumed either *static*, i.e., has to make his selection initially without the possibility to change it later on, or *adaptive*, in which the adversary can change the nodes as long as not exceeding the threshold f at any given time.

2) Synchrony Models: BFT protocols rely on synchrony models to capture temporal behavior and timing assumptions, which are important for the concrete protocol designs. In this subsection, we review popular models (see Tab. I) and discuss how they affect practical system design.

a) Asynchronous System Model: No assumptions are made about upper bounds for network transmissions or performing local computations. These are said to complete *eventually*, meaning they happen after an unknown (but *finite*) amount of time. For instance, the network cannot "swallow up" messages by infinitely delaying them. The asynchronous model is the most general, yet it complicates the design of protocols in this model since no timers can be used in the protocol description. It was proven impossible to deterministically achieve consensus in the presence of faults in an asynchronous system [21]. This problem is solvable in a synchronous system [18] where timers can be used to detect failures.

System model	Bound δ exists?	Bound holds ...
Synchronous	known δ	always
Eventually synchronous	known δ	after unknown GST
Partially synchronous	unknown δ	always
Asynchronous	unbounded	

TABLE I: Overview over different synchrony models.

b) Synchronous System Model: Here we assume that strict assumptions can be made about the timeliness of all events. In particular, the synchronous system model assumes the existence of a known upper bound δ for the time needed for both message transmissions over the network and local computations. In practice, choosing δ to model a system can be bothersome: If the correctness of decisions depends on it, it must not be underestimated; however, if it is chosen too large, it may negatively impact the performance of a system.

c) Partially Synchronous System Model: Dwork et al. proposed a sweet spot between the synchronous and asynchronous model [22]. Partial synchrony comes in two versions: *Unknown bounds* partial synchrony assumes an unknown bound that always holds. *Global stabilization time (GST)* partial synchrony assumes the bound is initially known but only holds *eventually*, i. e., after some unknown time span which is modeled by the GST. The latter is often also referred to as *eventual synchrony* since the system behaves exactly like a synchronous system as soon as GST is reached. Partial synchrony is popular among many BFT protocols (like PBFT) that guarantee liveness only under partial synchrony but always remain safe even when the system is asynchronous.

B. State Machine Replication and Consensus

State machine replication (SMR) is a technique for achieving fault tolerance by replicating a centralized service on several independent replicas, which emulate the service. Replicas agree on inputs (transactions) proposed by clients and thus ensure a consistent state and matching results. Such agreement is typically achieved through a consensus protocol. A *consensus* protocol guarantees that all correct participants eventually decide on the same value from a set of proposed input values. Moreover, an SMR protocol additionally requires *output consolidation*: The client needs to collect at least $f+1$ matching responses from different replicas to assert its correctness. Formally, SMR should satisfy the following guarantees:

- *Safety* (*Linearizability*): The replicated service behaves like a centralized implementation that executes transactions atomically one at a time [23].
- *Liveness* (*Termination*): Any transaction issued by a correct client eventually completes [24].

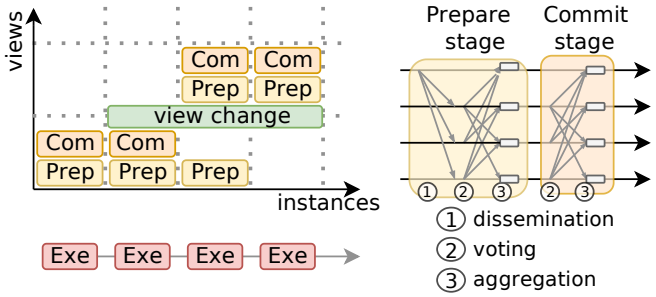


Fig. 3: A simplified BFT SMR protocol model.

C. BFT Simplified

A BFT protocol implements SMR in a Byzantine fault model. To explain BFT protocols, we employ an abstract model (see Fig. 3) that contains many common design aspects from BFT protocols, and we borrow some terminology from PBFT. BFT protocols can be leaderless [25], [26] or work with multiple leaders [15], [27], [28], but most BFT protocol designs operate using a single leader.

To implement SMR, replicas must repeatedly agree on a block containing one or more inputs. We refer to one such agreement as an *instance*. To reach agreement typically requires the successful execution of two or more *protocol stages*. The two protocol stages, PREPARE and COMMIT of PBFT are shown in Fig. 3. Each protocol stage includes *dissemination* of one or multiple proposals, *voting* for a proposal by all replicas, and *aggregation* of votes. In some protocols and stages, replicas only vote on whether aggregation was successful. Thus, dissemination may be omitted, as shown in the commit stage in Fig. 3. During aggregation, matching votes from different replicas are gathered to form a *quorum certificate*: Quorum certificates contain votes from sufficiently many replicas to ensure that no two different values can receive a certificate; the value is now *committed*.

After reaching an agreement in one instance, transactions in the decided (committed) block are ready for their *execution* in all correct replicas. Subsequently, correct replicas reply back to the respective clients.

In addition to instances, BFT protocols typically operate in logical views, where each *view* describes one composition of replicas and may use a predefined leader or certain dissemination pattern. While some protocols perform all stages of different instances in one view as long as agreement can be reached, as is shown in Fig. 3, other protocols change the view for every stage or instance. To change the view requires a *view change* mechanism. If not sufficiently many replicas reach agreement, e. g., due to a faulty leader, the view change mechanism can synchronize replicas, replace the protocol leader, and eventually ensure progress by installing a leader under whom agreement instances finally succeed. During view change, replicas exchange quorum certificates to ensure that agreements from previous views are continued in the next.

III. METHODOLOGY

We create this survey by systematically reviewing a large body of relevant research work. To find research work and determine its relevance, we conducted a systematic search, which we briefly explain in this section. We performed a systematic search focusing on the scalability of BFT consensus. Since we aim to increase transparency and reproducibility, we briefly present our search methodology to gather and select these research papers, which mainly originated from the field of distributed systems, and especially the BFT and blockchain communities. Our main ambitions are the following:

- Find distinct approaches in the literature for improving the *scalability of BFT consensus*.
- Identify and classify scalability-enhancing techniques to create a taxonomy for scalable BFT.

A. Search Strategy

For literature research, we used Semantic Scholar [29], whose API allows the automated download of a large number of references. We manually crosschecked with Google Scholar to ensure relevant papers were included. We employed the following search queries without filters on the publication date:

Search 1:
Query = Byzantine consensus scalability blockchain
Publication Type = Journal or Conference
Sorted by Relevance (Citation Count) = 250 most cited publications

Search 2:
Query = Byzantine consensus scalability blockchain
Publication Type = Journal or Conference
Sorted by Date = 250 newest publications

For each search query, on December 17, 2021, we downloaded all results (954 papers) and sorted them locally by relevance (Search 1) and by date (Search 2). The first search favors relevance (citation count) to ensure that we can cover the most influential works for the topic, while the second search favors publication date to ensure that we can also regard the most recent publications, which might not have been cited often enough due to their recency. With these searches, we cover all publications with at least 11 citations and further capture all publications that have been published since January 2021. By using two different sortings, we aim to include both the most relevant and most recent works. We found that the search engine could handle synonyms well, e. g., papers would not evade our systematic search when using “agreement” instead of “consensus”. After that, we combined the results of these queries and sanitized them for duplicates.

B. Selection Criteria

Further, we employ a set of selection criteria that can be applied to determine if a paper found by the search queries is original research and *potentially relevant*. This means it is selected for inclusion in the survey as long as no exclusion criteria apply. These criteria also encompass conditions used to assess the quality of a found paper.

Inclusion criteria – A paper is included if its contributions satisfy one of the following criteria:

- The paper presents a novel technique for *improving the scalability* of communication-based Byzantine consensus
- The paper *combines existing scalability-enhancing techniques*, or ideas, in a novel way, achieving better results than state-of-the-art protocols

Exclusion criteria – The exclusion process is applied *after inclusion*. A paper is excluded if only one of the following criteria applies:

- The paper is not a research paper (no practical reports, workshop invitations, posters, or other surveys).
- The paper does not cover the scalability aspect sufficiently from a technical point of view or does not originate from the field of computer science (i. e., not originating from the right 'field', e. g., we do not want papers from business informatics or social science).
- The presented paper proposes a scalability mechanism but does not present a careful experimental evaluation showing how it can actually improve scalability (not enough validation).
- The paper lacks relevance: while the paper presents some scalability mechanism(s), the presented mechanism is an incremental refinement of an earlier proposed mechanism.
- The paper does not focus on communication-based Byzantine consensus but presents a 'Proof-of-X' consensus variant.

C. Selection Procedure and Results

We selected papers as *relevant* in the following way: In the first phase, we conducted a fast scan, in which a single assessor examined each paper only to check if the paper could be of potential interest. Moreover, in this step, only papers that obviously do not qualify are sorted out. For instance, if the field is not computer science, the paper's topic is not related to scalability, or the paper is not a research paper but a short abstract or workshop invitation. In a second review round, all remaining papers are examined by two different reviewers to validate their relevance: A paper is *relevant* iff at least one inclusion criterion applies, and none of the exclusion criteria applies. All papers are then tagged either *relevant* or *not relevant*. In the final phase, assessor conflicts are resolved by involving a third assessor and discussing the disagreement.

In the end, 52 papers have been selected as relevant for inclusion in our SoK paper. Out of these, 13 papers are published in 11 different journals, 7 papers are published in online archives (e. g., arXiv), and 32 are published in proceedings of 25 different conferences. The papers are spread over many conferences and journals, many not specific to the topic of blockchain. The most frequent publication channel, besides arXiv, is VLDB, with 3 papers.

IV. SoK: Scalability-enhancing Techniques

Improving the scalability of Byzantine consensus is an ongoing research field that requires the exploration, advancement, and combination of several methods and approaches. In order to divide the different directions of approaches, we identified the following categories for scalability-enhancing techniques which aim to increase the scalability (and efficiency) of Byzantine consensus:

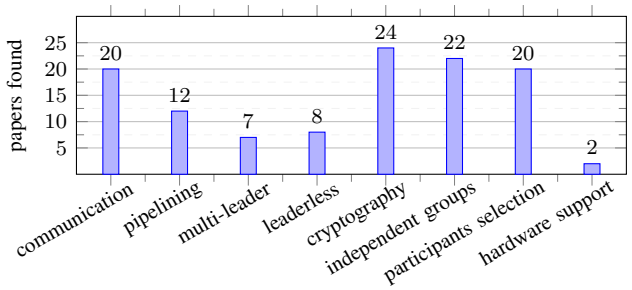


Fig. 4: For each category: How many papers used at least one scalability-enhancing technique fitting to this category.

- *Communication topologies and strategies*. How can the communication flow be optimized? Increasing the communication efficiency might require a suitable *communication topology*, e. g., tree-based or overlay / gossip-based.
- *Pipelining*. What benefit can be achieved by employing pipelining techniques for parallel executions of agreement instances?
- *Cryptographic primitives*. How can *suitable cryptographic primitives* serve as building blocks for new scalability-enhancing techniques?
- *Independend groups*. How can transactions be ordered and committed *in independent groups*, e. g., through sharding?
- *Consensus committee selection*. How are roles in achieving agreement distributed among network nodes? Does one (or multiple) flexibly selected *representative committee(s)* decide?
- *Hardware support*. How can we improve consensus efficiency using *trusted execution environments (TEEs)*?

In Fig. 4, we show an overview of how many papers we found to fit into each category.

A. Communication Topologies and Strategies

Improving the scalability of coordinating agreement between a large number of n nodes requires avoiding bottleneck situations, such as burdening too much communication effort on a single leader. An asymmetric utilization of network or computing resources hinders scalability, as observed in many traditional BFT protocols resembling PBFT. Instead, one of the main goals of the "blockchain generation" of BFT protocols is to distribute the communication load among replicas as evenly as possible. For instance, some approaches try to balance network capabilities under a single leader by using message forwarding and aggregation along a communication tree [17], [30] or gossip [31], while others rely on utilizing multiple leaders [15], [27] or even work fully decentralized without a leader [16], [26], [32], [33].

Another reason why the well-known PBFT protocol does not scale well is its all-to-all broadcast phases which make

the network topology a *clique*, which is usually always a bad choice for scalability as it means an incurred $O(n^2)$ number of messages (and necessary message authenticators). Unsurprisingly, this design is not an ideal fit for a large network size n and was later replaced by protocols that only require linear message complexity by collecting votes [9], [10].

In the following, we review and explain different approaches for improving communication flow in BFT protocols categorized by the network topology they employ.

1) Star-based: Recently, many BFT protocols have proposed to employ linear communication complexity by employing a star-based communication topology [9], [10], [34]–[38]. We briefly explain this idea and its limitations on the example of HotStuff [9] (which is illustrated in Fig. 5).

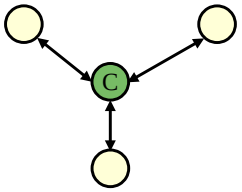
The HotStuff leader acts as a *collector*, which gathers votes from other replicas, and, as soon as a quorum of votes is received, creates a *quorum certificate* that can convince any node to progress to the next protocol stage. The rather costly *view-change* subprotocol of PBFT can be avoided by adding a further protocol stage. Thus, agreement in HotStuff requires four protocol stages (PREPARE, PRE-COMMIT, COMMIT, and DECIDE) shown in Fig. 5b.

Important for scalability is that the cost of transmitting message authenticators can be reduced by a simple aggregation technique: The leader compresses $n - f$ signatures into a single threshold signature of fixed size. Because the threshold signature scheme uses the quorum size as a threshold, a valid threshold signature implies a quorum of replicas is among the signers. The threshold signature has the size of $O(1)$ – a significant improvement over letting the leader transmit $O(n)$ individual signatures. HotStuff also employs pipelining and leader rotation which are addressed in §IV-B.

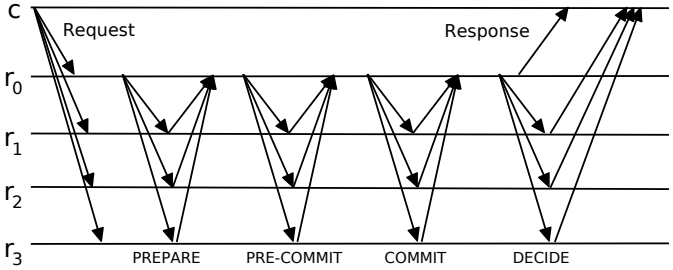
As we can see in Fig. 5b, the communication flow still is imbalanced: Every follower replica communicates only with the leader, but the leader still has to communicate with all other replicas. This bottleneck can be alleviated by tree-based or randomized topologies, as we explain next.

2) Tree-based: A few novel BFT protocols employ a tree-based communication topology [17], [30], [39], [40]. This has the advantage that the leader is relieved of the burden of being the sole aggregation and dissemination point for votes and the generated quorum certificates. The tree-based communication pattern can be introduced into either the PBFT algorithm, as done in ByzCoin [30], or HotStuff, as done in Kauri [17], without requiring significant changes to the protocol. ByzCoin was the first to show the potential of this topology, while Kauri later showed how to overcome the shortcomings of this approach.

The tree-based communication topology raises several problems, such as added latency, compared to the star topology and the necessity to react to failures of internal and leaf nodes. In the following, we briefly explain these challenges and the solutions proposed. Fig. 6 shows the communication pattern in Kauri. Compared to the pattern in Fig. 5, it is clear that each level in the tree increases latency. To address this issue, Kauri only considers trees of height 3, as shown in the Figure.

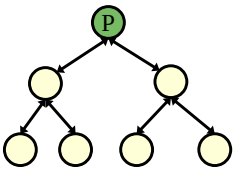


(a) Star topology: A collector (C) collects votes and distributes quorum certificates (in Figure 5b, replica r_0 has this role).

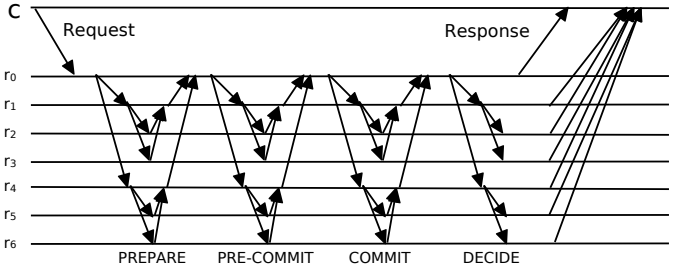


(b) HotStuff has linear communication complexity: The leader collects the votes from the other replicas and distributed the quorum certificates. Aggregation can be achieved through threshold signatures.

Fig. 5: Linear communication over a star-based network topology on the example of HotStuff.



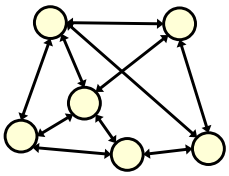
(a) Tree topology: The proposer (P) disseminates messages and collects votes along a tree.



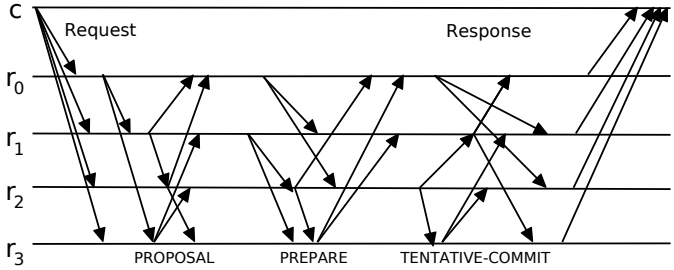
(b) In Kauri, the leader disseminates a proposal along a communication tree. The inner nodes forward messages to their children and collect and aggregate their children's votes, to be returned to the parent.

Fig. 6: Communication over a tree-based network topology on the example of Kauri.

Additionally, in the star topology, only leader failure is critical and typically requires a view change. In the tree topology, the failure of an internal node prevents the aggregation of votes from its children and thus requires reconfiguration. Further, the failure of a leaf node may cause its parent to wait indefinitely for its contribution. Kauri introduces a timeout for aggregation to handle leaf failure. To address the failure of internal nodes, Kauri proposes a reconfiguration scheme, which guarantees to find a correct set of internal nodes, given that failures lie below



(a) Randomized topology: Gossip between randomly chosen, connected nodes that are called *neighbors*.



(b) In Gosig, replicas communicate with a *fanout* (i. e., number of randomly selected neighbors) to disseminate a block or votes for a block within each protocol stage. Each instance is started by a randomly selected leader.

Fig. 7: A randomized communication strategy through gossip on the example of Gosig [41].

a threshold k. Otherwise, Kauri falls back on a star-based topology. The threshold k here depends on the tree layout but lies at $\sqrt{n}$ for a balanced tree.

To achieve competitive throughput and utilize the available bandwidth despite the latency added through the tree overlay and through timeouts on leaf nodes, Kauri relies on pipelining. We will see more details on how pipelining can serve as a means for increasing scalability in §IV-B.

3) Gossip and Randomized Topology: Furthermore, many recent BFT protocols rely on gossip, or randomized communication topologies, for instance Internet Computer Consensus [42], Tendermint [12], Algorand [13], Gosig [41], scalable and leaderless Byz consensus [43], Avalanche [14] and RapidChain [44].

As shown earlier, deterministic leader-based BFT consensus protocols are much affected by the high communication costs of broadcasts, i. e., if the leader disseminates a block of ordered transactions to all other nodes. An idea that enhances scalability is to relieve the leader and shift this burden equally to all nodes by disseminating messages through gossiping over a randomized overlay network – which was recently proposed by protocols like Algorand [13], Tendermint [12], or Gosig [41].

When using gossip, the leader proposes a message m to only a constant number of k other, randomly chosen nodes. Nodes that receive m forward the message to their own randomly chosen set of connected neighbors, just like spreading a rumor with high reliability in the network. Note that this technique is probabilistic, and the probability of success (as well as the propagation speed) depends on the fanout parameter k, the number of hops, and the number of Byzantine nodes present in the system. Gossip allows for the communication burden to be lifted from the leader, leading to a fairer distribution of bandwidth utilization, where each node communicates only with its $O(k)$ neighbors, thus improving scalability.

Fig. 7 shows the communication pattern of Gosig [41]. Similar to the tree-based overlay, the communication pattern leads to increased latency. As Kauri, Gosig uses pipelining to mitigate this latency. Different from Kauri, randomized topologies do not require reconfiguration in case of failures.

Random overlay network topologies can also be utilized in leaderless BFT protocols. Avalanche [14] is a novel, leaderless BFT consensus protocol that achieves consensus through a metastability mechanism that is inspired by gossip algorithms. In Avalanche, nodes build up confidence in a consensus value by iterative, randomized mutual sampling. Each node queries k randomly chosen other nodes in a loop while adopting the value replied by an adjusted majority of nodes so that eventually, correct nodes are being steered towards the same consensus value. In this probabilistic solution, nodes can quickly converge to an irreversible state, even for large networks.

B. Pipelining

Pipelining is a technique that allows replicas to run multiple instances of consensus concurrently, and it is employed with a special focus on improving the scalability of the BFT protocol, e. g., in HotStuff [9], Kauri [17], BigBFT [27], SBFT [10], Proof-of-Execution (PoE) [36], ResilientDB [45], Dispel [33], RapidChain [44], Hermes [46], RCC [47], Gosig [41], Narwhal-HotStuff/Tusk [48], and Jolteon/Ditto [37].

Further, pipelining boosts performance and scalability because replicas maximize their available resource utilization. For instance, the available bandwidth can be better utilized if data belonging to future consensus instances can be disseminated while replicas still wait to collect votes from previous, ongoing consensus instances. This means pipelining can systematically decrease the time replicas spend in an idle state, e. g., waiting to collect messages from others. Gosig [41], for example, pipelines both the BFT protocol, as committed nodes can start a round while still forwarding gossip messages on the previous one, and the gossip layer, by decoupling block and signature propagation. With pipelining, a replica can participate in multiple consensus stages simultaneously, and thus help to overcome *limitations* for system performance, such as *reusing quorum certificates* for multiple consensus stages in HotStuff or *mitigating tree latency* in Kauri.

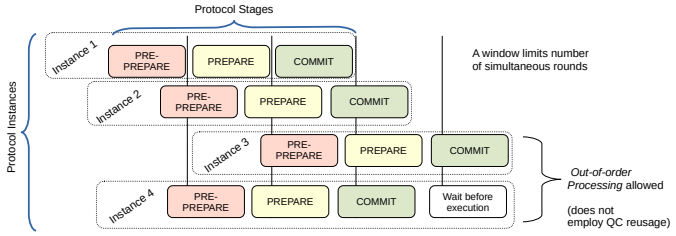
1) Out-of-Order Processing: A basic variant of pipelining was also introduced with PBFT: The leader can build a pipeline of concurrent consensus instances by starting multiple instances for a specific allowed range defined through a low and high watermark. Within this bound, pipelining is *dynamic*, allowing the protocol to start fewer or more concurrent instances, based on the load. PBFT employs *out-of-order* processing (see Fig. 8a), which means that replicas in the same view vote on and commit in all allowed consensus instances without waiting for preceding instances to complete first. Yet, in this case, executions must be delayed until all transactions within lower-numbered consensus instances have been executed.

Similar to PBFT, the SBFT protocol [10] allows the parallelization of blocks, in which multiple instances of the protocol can run concurrently, but it additionally introduces an adaptive learning heuristic to *dynamically optimize its block size*. This heuristic employs a configurable maximal recommended parallelism parameter, the number of currently ongoing, pending blocks, and the number of outstanding client requests. SBFT's heuristic strives to distribute and balance pending client transactions evenly among the recommended number of parallel executing consensus instances, and new requests are not required to wait when currently no decision block is being processed.

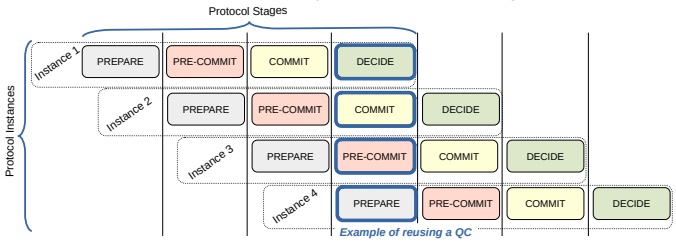
Moreover, PoE [36] is a new BFT design employed within ResilientDB [45] for achieving high throughput using a multi-threaded pipelined architecture. Its core innovation is to combine out-of-order processing and dynamic pipelining, as introduced in PBFT, with *speculative executions*, where transactions are executed before agreement. Since PoE allows replicas to execute requests speculatively, it introduces a novel view-change protocol that can rollback requests.

2) Chain-based Pipelining & QC Reusage: In chain-based protocols like HotStuff, a block is committed after having passed through several protocol stages, as shown in Fig. 5b. In each stage, the leader collects votes from the other replicas, aggregates these in a quorum certificate (QC), and disseminates this QC to all as proof of completing the respective protocol stage. HotStuff incorporates the idea of pipelining by making each protocol stage into a pipelining stage (see Fig. 8b), and thus allowing a QC to certify advanced stages of previous, concurrently executing consensus instances, e. g., a DECIDE QC of instance i can act as a COMMIT QC of instance $i+1$ and PRE-COMMIT QC of instance $i+2$. Pipelining in HotStuff is *logical*, meaning that concurrent instances are using the same messages. The number of pipelining stages in HotStuff equals its number of protocol stages, namely four. Narwhal-HotStuff and Tusk [48] as well as Jolteon and Ditto [37] built on HotStuff and use the same pipelining mechanism.

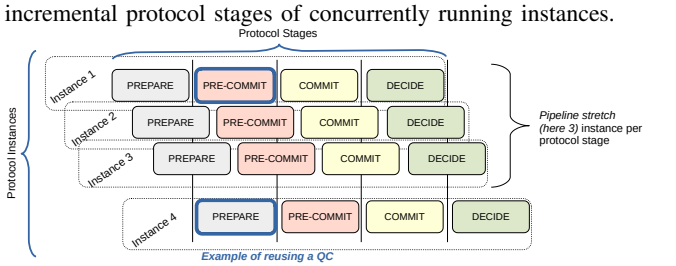
3) Multiplexing Consensus Instances per Protocol Stage: Chain-based pipelining can be further optimized by multiplexing consensus instances per protocol stage, as done by Kauri. For this purpose, Kauri refines the communication flow of HotStuff by proposing a tree topology with a fixed-sized fanout (number of communication partners). This load-balancing technique relieves the necessary uplink bandwidth (and thus also the time needed to send data) for message distribution at the root (leader). But at the same time, the communication tree increases the number of communication steps necessary for reaching agreement and thus impacts the overall latency of the SMR protocol. In Kauri, a more sophisticated pipelining method than in HotStuff acts as a solution to sustain performance despite this higher number of communication steps: The number of concurrently run consensus instances is decoupled from the fixed number of protocol stages by introducing a *stretching factor*, which is a parameterizable multiplicative to the fixed pipelining depth of HotStuff. For instance, if a pipeline stretch of 3 is used, the



(a) *Out-of-Order Processing* (here on the example of PBFT): The leader can start multiple instances for a given window of allowed consensus instances concurrently. QCs are not being reused.



(b) *Chain-based Pipelining* uses one pipeline stage per protocol stage (this example shows HotStuff). QCs can be reused to verify incremental protocol stages of concurrently running instances.



(c) *Multiplexing Consensus Instances per Protocol Stage*: A *pipelining stretch* number of consensus instances can be concurrently started per protocol stage (as done in Kauri). QCs can be reused.

Fig. 8: Pipelining variations of single leader BFT SMR.

Kauri leader can simultaneously start 3 consensus instances every time a HotStuff leader would start a single instance, leading to a total of 12 concurrent consensus instances instead of 4 (like in HotStuff) once the pipeline is filled. In contrast to PBFT, the pipeline stretch in Kauri is static and cannot dynamically adapt to the load, and instances cannot complete out-of-context (see Fig. 8c).

4) Pipelining in Multi-leader and Leaderless Protocols: Many BFT protocols rely on the idea of having not only one but several leaders that can concurrently start agreement instances (e. g., [15], [27], [28]). This design benefits scalability because it brings a fairer distribution of the task of broadcasting block proposals and can prove a suitable solution for the leader bottleneck problem. Yet, it also introduces novel challenges, in particular when it comes to the *coordination of leaders*, to prevent conflicts (i. e., identical transaction being proposed by multiple leaders in different blocks) and also to maintain *liveness* for all transactions in the presence of Byzantine leaders. To serve as an example, Mir-BFT [15] is a multi-leader protocol in which several leaders propose blocks independently and in parallel, by employing a mechanism that rotates the assignment

of a partitioned transaction hash space to the leaders. Another example is RapidChain [44], which is a BFT protocol that uses sharding and allows a shard leader to propose a new block while re-proposing the headers of the yet uncommitted blocks, thus creating a pipeline to improve performance.

BigBFT [27] is a pipelined multi-leader BFT design that aims to overcome the leader bottleneck of traditional BFT protocols by partitioning the space of sequence numbers (consensus instances) among selected leaders (coordination stage) and thus allowing multi-leader executions. Further, it logically separates the coordination stage from the dissemination and voting of new blocks. Pipelining is employed in two ways. Different leaders perform their instances concurrently. Additionally, the next coordination stage is run concurrently with the last agreement stages. Similarly, RCC [47] concurrently executes multiple instances with different leaders. Instead of a coordination stage, RCC uses a separate, independent instance of a BFT protocol for coordination.

A key innovation of Dispel [33] is its *distributed pipeline* which builds on previous work on leaderless BFT consensus [32]. In contrast to centralized pipelining (where the decision, when to create a new agreement instance, is only up to the leader(s)), all Dispel replicas can locally decide whether to start new consensus instances based on their available system resources like network bandwidth or memory. Dispel builds a consensus pipeline by creating pipelining stages for *individual tasks* within running consensus that consumes different system resources. In particular, it employs four stages: network reception, network transmission, CPU-intensive hash, and latency-bound consensus [33]. The idea of Dispel is that a replica can maximize its own resource utilization by executing four consensus instances, one for each stage, concurrently. We cover *leaderless protocols* in more detail later in §IV-E2.

C. Cryptographic Primitives

Efficient cryptography schemes can increase a protocol's scalability. We identified several approaches, most aimed at reducing the communication complexity via message aggregation: multi-signatures (§IV-C1), threshold signatures (§IV-C2), secret sharing (§IV-C3), erasure coding (§IV-C4), as well as efficiently selecting a committee of nodes using verifiable random functions (VRFs) (§IV-C5).

1) Multi-Signatures: Seven papers that match our criteria aggregate messages using multi-signatures: Kauri [17], Gosig [41], ByzCoin [30], Musch [34], AHL [38], BigBFT [27], and RepChain [49]. A multi-signature allows n participants, who want to jointly sign a common message m, to create a signature σ of m so that verification can confirm that all n participants have indeed signed m. This can be achieved by aggregating multiple signatures, e. g., via multiplication, resulting in a multi-signature whose combined size and verification cost is comparable to that of an individual signature [50].

In BFT systems, multi-signatures are often used to combine the votes of multiple participants to reduce the message complexity and the memory overhead of the protocol from quadratic to linear [17], [27], [34], [41]. A replica casts its vote by adding its partial signature share to the multi-signature, and the voting concludes once a quorum of signature shares has been received [17]. Most protocols make use of asynchronous non-interactive multi-signatures, i. e., BLS multi-signatures [50], [51]. Replicas aggregate their signature share in one small multi-signature. This reduces the signature size compared to individual replica signatures; however, the computation takes longer compared to, e. g., ECDSA signatures. Multi-signatures thus offer accountability, as it can be checked who signed a message. Further, they have the advantage that the order of signatures can be arbitrary, making them resilient against adaptive chosen-player attacks [41]. A node can sign the message multiple times, e. g., when it receives the same message from different nodes during gossip communication. To ensure correct verification, the number of times each node signs the message is tracked [41].

Signatures are collected during the voting phase, e. g., during gossip as in Gosig or during vote aggregation phases as in Kauri, where each replica receives votes from its children in the communication tree, and once a quorum of signatures is collected, the replica enters the next phase. BigBFT [27] is a multi-leader protocol where each leader proposes client requests in a block in instance r. All blocks are signed by all other nodes during the vote phase upon reception. The set of $n - f$ votes for a block is then aggregated in a multi-signature, which is piggybacked onto the proposed block in the next instance $r + 1$ to commit the block of instance r. BigBFT avoids the communication bottleneck of single-leader protocols and reaches consensus in two communication rounds by pipelining blocks using message aggregation and multi-signatures to reduce message complexity. In Musch [34], nodes exchange and aggregate signed hashes of blocks to create one collective signature instead of repeating multiple replica signatures, thus keeping the signature size constant. If not enough signature shares can be collected, a view change is performed, which also employs aggregated multi-signatures.

Another scheme that is used to aggregate signatures is the collective signing protocol CoSi [52], which can effectively aggregate a large number of signatures. It is based on Schnorr multi–signatures and, contrary to the non-interactive BLS multi-signatures, it requires a four-phase protocol run over two round-trips to generate a CoSi multi-signature. The participants can be organized in communication trees for efficiency and scalability, as discussed in §IV-A. A node can request a statement, i. e., a request, to be signed by a group of witnesses, i. e., the replicas, and the collective signature attests that the node, as well as the witnesses, have observed this request. It is used in ByzCoin [30] to reduce the cost of the underlying PBFT's prepare and commit phases, as well as in RepChain [49], to enable efficient cross-shard transactions without requiring multiple individual signatures. However, the security of two-round multi-signatures has been shown to be compromised [53].

2) Threshold Signatures: A total of nine papers utilize threshold signatures in their protocol design: Jolteon/Ditto [37], Saguaro [54], ICC [42], Narwhal/Tusk [48], PoE [36], Hot-Stuff [9], SBFT [10], Cumulus [55], and Dumbo [56]. In

threshold cryptosystems, participants each have a public key and a share of the corresponding private key: in (t, n)-threshold systems, at least t partial shares from participants are needed to decrypt or sign a message. Participants can sign a message with their secret key share, generating a signature share. This signature share can be verified or combined with others into an aggregated signature, which can again be verified. This makes threshold signatures a special case of multi-signatures, where instead of all participants (n-out-of-n) only a subset (t-out-of-n, $t < n$) have to participate. BLS multi-signatures can be transformed into threshold signatures; however, the implementation of threshold signature schemes is more complex, and multi-signatures require less computation [10].

Threshold signatures are, therefore, often used in BFT systems for aggregation of messages, similar to multi-signatures: a block or vote message is signed by a quorum of nodes (commonly with a threshold $t = 2f + 1$), the quorum's shares are combined in one authenticator, and the signature share is used as a replica's vote to confirm consensus [9], [36], [37], [42], [54], or for example to create a quorum certificate for side-chain checkpoints as in Cumulus [55]. BLS signatures can be used for this as well, as done by ICC [42] and PoE [36]: either by using the standard BLS scheme with a secret key shared amongst all participants, which creates unique and compact signatures, or by using BLS multi-signatures, where a signature share is a BLS signature which then gets combined into a new signature on an aggregate of the individual public key. PoE can use threshold signatures or message authentication codes depending on the number of participants in the network.

Threshold signatures can be used just as multi-signatures to split one phase of high-complexity broadcast communication into two phases of linear communication complexity. Dumbo [56] introduces two new asynchronous atomic broadcasting protocols. The first protocol, Dumbo1, reaches asymptotical efficiency, improving upon the design of threshold encryption and asynchronous common subsets of HoneyBadgerBFT [57]. The second protocol, Dumbo2, further reduces the overhead to constant by efficiently using multi-valued Byzantine agreement (MVBA) running over a reliable broadcast which outputs a threshold signature proof that of all receivers of an input, at least one receiver is an honest peer. SBFT [10] uses threshold signatures to reduce the communication complexity to linear by extending PBFT by $c+1$ collectors. Nodes send their messages to the collectors, who then broadcast the combined threshold signature once $3f + c + 1$ shares have been received. Using threshold signatures reduces the message size of the collector from linear to constant, and the client overhead is reduced as only one signature has to be verified. The execution is similarly aggregated by $c+1$ execution collectors who collect $f + 1$ signature shares. SBFT uses BLS signatures, though BLS multi-signatures instead of threshold signatures are used on the fast path as these require less computation.

Another usage of threshold cryptography is generating randomness used for leader election in asynchronous networks. Narwhal [48] uses an adaptively secure threshold signature scheme to generate a distributed perfect coin, while Jolteon [37] generates randomness for each view by hashing the threshold signature of the view number in order to create an asynchronous fallback protocol to circumvent the FLP impossibility. For ICC [42], a sequence of random beacon values is created to determine the permutation of the participants and thus the leader, similar to verifiable random functions (c. f., §IV-C5). Starting from a known initial value, a participant generates the next sequence of the random beacon by broadcasting its signature share of the current random beacon value. At least one honest participant's signature share is required to form a comprehensible random value.

3) Secret Sharing: In secret sharing approaches, a secret is split and distributed amongst a group of n nodes so that it can be reconstructed for cryptographic operations when a sufficient number (i. e., a threshold t) of shares are combined but no single node can reconstruct the full secret by itself. The splitting and distribution are typically performed by a dealer. This secret sharing amongst replicas can, similar to threshold signatures, reduce the overhead of multi-signatures. Contrary to the threshold signatures of §IV-C2, however, the secret sharing in FastBFT [40] requires hardware-based trusted compartments on all replicas. The trusted compartment on the leader can securely create secrets, split them, and deliver them to all other replicas. This is performed in a separate pre-processing phase before the agreement phase. Once the order has been established, the replicas release in the final step their shares of the secret to allow verification of the agreement. All secrets are one-time secrets, and a monotonic counter value is bound to the secret shares in order to prevent equivocation of the leader. For more details on trusted counters and trusted execution environments, we refer to §IV-F. Secret sharing can also be used to prevent leakage of sensitive data: clients in Qanaat [58] use $(f + 1, n)$-threshold secret-sharing to keep data confidential.

4) Erasure Coding: Three protocols use erasure codes: Dumbo [56], ICC2 [42], and DispersedLedger [59]. The first two aim to reduce the communication overhead and lessen the bottleneck on the leader, whereas DispersedLedger focuses on data storage and availability. Erasure codes are forward error correction codes that can efficiently handle bit erasures during transmissions. In $(n, n - 2t)$-erasure codes, a message m is split into n fragments of larger size so that any subset of $n - 2t$ fragments can be used to reconstruct the original message m even if fragments get lost or corrupted. Instead of broadcasting a given payload, a leader can encode this payload and send the individual fragments to different replicas. With a correct encoding, the payload can be reconstructed, even if some replicas remain unresponsive. While reducing network load on the leader, this technique adds computational overhead for encoding and decoding. In ICC, large blocks have to be disseminated, which can become a bottleneck, so as an improvement of their peer-to-peer gossip sub-layer (ICC0/1), they propose a subprotocol of reliable broadcast with $(n, n - 2t)$-erasure codes ($t < n/3$) in ICC2 in combination with threshold signatures. Dumbo's reliable broadcast protocol can be optimized as well using a $(n - 2t, n)$-erasure code

scheme in combination with a Merkle tree, which tolerates the maximal adversary boundary and thus helps honest nodes recover efficiently. In DispersedLedger, data is stored across n nodes via a verifiable information dispersal protocol making use of erasure codes. This guarantees data availability and allows separating the tasks of agreeing on a short, ordered log and of downloading large blocks with full transactions for execution. This decoupling of the protocol stages leads to faster progression of the protocol as the high-bandwidth task of downloading transactions is no longer on the critical path.

5) Verifiable Random Functions: Five papers incorporate verifiable random functions (VRFs) [60] into their design: Algorand [13], Beh-Raft-Chain [61], Cumulus [55], Proof-of-QoS [62], and DLattice [63]. A VRF is a cryptographic function $VRF_{sk}(x)$ that for an input string x returns both a hash value and a proof π. The hash value is here uniquely determined by both the user's secret key sk as well as the input x, but appears undistinguishable from a random value for anyone not in possession of sk. The proof π allows anyone who knows the public key pk to verify whether the hash value corresponds to x, without revealing sk [13]. VRFs in BFT protocols facilitate the selection of committees, which in turn can efficiently perform the consensus. Their non-interactive nature makes them desirable as it prevents any targeted attack on the leader(s) or committee members of the next instance, as their membership status is not known in advance. For more details on the usage of committees for scalability in BFT protocols, we refer to §IV-E.

Algorand uses a VRF based on the nodes' key pairs as well as publicly available blockchain information for cryptographic sortition, meaning to select the committee members in a private and non-interactive way so that nodes can independently determine whether they are in the committee for this instance. VRFs create a pseudo-random hash value that is uniformly distributed between 0 and $2^{hashlen} - 1$, meaning that nodes get selected at random to be in the committee. As VRFs do not require interaction between the nodes and are calculated using private information, a node's membership status cannot be determined in advance to launch a targeted DoS attack or allow malicious nodes to collude; instead, a node's membership status is only known retrospectively. Algorand also includes a seed in each instance, which is calculated using the VRF result in combination with the previous instance's seed; if this seed is not included in a proposed block, it is discarded as invalid. If multiple blocks get proposed by committee members, then the included seed value is used to determine a priority amongst these blocks. Furthermore, the VRF value can be used as a random value for coins, for example, in order to resume normal operation after a network partition. Algorand implements its VRFs over Curve25519 [64], [65], and shows that VRF and signature verification are CPU bottlenecks in the protocol. VRFs are used similarly in Beh-Raft-Chain [61] and DLattice [63] for cryptographic sortition in order to determine a node's role and membership status, e. g., (local) committee leader. Proof-of-QoS [62] splits nodes into regions, and one node per region is selected for the BFT committee based on its quality of service (QoS). Out of the κ nodes with the highest QoS in a region, the node with the smallest hash of id and seed value is selected. This node's membership status is confirmed with its VRF hash, and others verify the corresponding proof.

In Cumulus [55], VRFs are used in a novel cryptographic sortition protocol called Proof-of-Wait, which is used by this side-chain protocol to select the epoch's representative to interact with the mainchain. At most, one representative can be chosen per epoch, and it is based on nodes calculating a random waiting time based on their VRF output. The VRF is implemented based on the elliptic curve Secp256k1.

D. Independent Groups

In this section, we look at how subsets of all nodes, or groups, can process transactions independently from the rest of the nodes. Of the 22 papers in this section, we classified 16 as using sharding techniques. The remaining six papers are classified as hierarchical consensus.

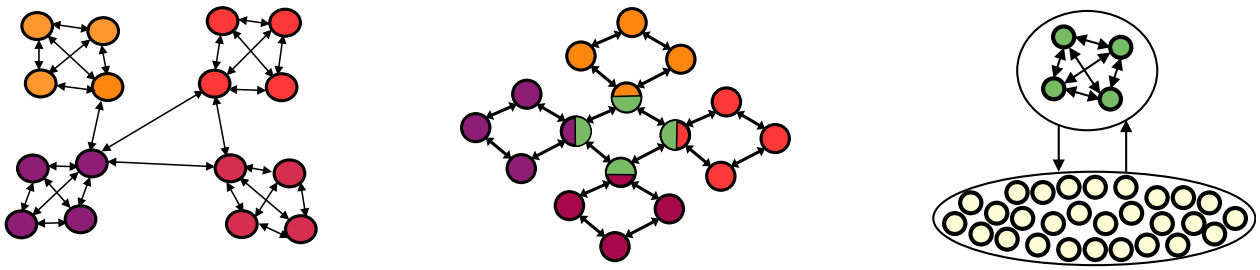
Sharding, traditionally used in database systems, splits a dataset into smaller subsets. In database systems, this is used when the data cannot fit onto a single machine anymore. This technique has been used by BFT protocols to improve scalability. In a sharding algorithm, the application's state is distributed over multiple, possibly overlapping, groups called shards [61], [66]–[68]. This allows efficient processing of transactions within shards, as only a subset of nodes has to participate in the consensus. Further, only the responsible group must execute the transaction.

Hierarchical consensus can but does not have to split state across different groups. Here, the groups use a higher-level consensus mechanism to coordinate. While shards can also communicate with each other, e. g., for cross-shard transactions, the higher-level consensus is a significant aspect of hierarchical consensus, which we use to distinguish these two categories.

1) Sharding: Sharding is a technique to split the system into multiple *shards*, each containing a subset of the replicated state and being managed by a subset of the nodes, as seen in Fig. 9a. Transactions that affect only a single shard can then be processed independently. This improves scalability by reducing the number of messages that have to be exchanged to reach consensus within shards. Within a shard, nodes typically run a classic consensus protocol like PBFT [66] or Raft [61].

Sharding also brings new challenges. First, as shards contain only a subset of nodes, special care must be taken so that the shards are not vulnerable. Next, once the shards are created, clients need to know where to send transactions. Lastly, some transactions affect multiple shards, which requires coordination between shards to process these multi-shard transactions.

The first challenge requires the protocol to assign nodes to shards while keeping the system safe and live. For this, different sharding algorithms assume different threat models. Assuming a global threshold of $\frac{1}{3}$ faulty nodes, there is the risk that more than $\frac{1}{3}$ of nodes within a shard are faulty, jeopardizing safety. This is especially a threat if an adaptive adversary is considered. Most algorithms stipulate that the adversary can only corrupt nodes between epochs, but not within an



(a) Replicas using sharding. Coordination between shards is commonly only required for cross-shard transactions.

(b) Hierachical consensus with two layers with "representatives" of the lower layer running the consensus in an upper layer.

(c) Schematic overview of a committee algorithm. Only a subset of all nodes performs consensus on transactions.

Fig. 9: Three ways of reducing consensus participants: sharding, hierarchical consensus, and consensus by committee.

epoch [44], [61], [66], [67]. This moves the challenge of safety to the shard formation. To ensure the distribution of malicious nodes within any shard is equal to the global share of malicious nodes, techniques like VRFs [60] or TEEs [38] can be used. A more challenging threat model is adaptive corruption without stipulations on when corruption can occur. In this setting, algorithms ensure safety by resampling the shards faster than an adaptive adversary can corrupt nodes [38], [69]. Some approaches form shards based on criteria such as reputation or past behavior to guarantee that well-behaved shards are formed [49], [61]. Chainspace [70] also allows for whole shards to be controlled by an adversary while still enforcing some restricted safety criteria. Explicitly, Chainspace can still guarantee encapsulation between state objects (smart contracts) and non-repudiation in case a complete shard is controlled by the adversary. The encapsulation of state objects guarantees that malicious smart contracts cannot interfere with non-malicious ones, i. e., the control mechanism of Chainspace. Combined with non-repudiation, where message authorship cannot get disputed, sources of inconsistencies can get identified by the control mechanism and thus punished.

With a sharding architecture, some care has to be taken so that transactions by clients end up in the correct shard. This is especially true for cross-shard transactions. The simplest solution is to have clients send transactions to all shards [49], [66], [70], [71]. This way the shards can decide if a transaction is relevant to the shard or discard the transaction otherwise. This is more work for clients but, as ignoring transactions not relevant for a shard is easy, the overhead for the shards is small. Transaction routing has also been adapted in different ways. One is that clients can send transactions to an arbitrary shard, which then forwards the transaction to the correct shard [68], [72]. In the Red Belly Blockchain [16], clients send balance and transaction requests to known proposers published in configuration blocks. The proposers can respond directly for balance (i. e., read) requests and forward transactions (i. e., write requests) to consensus instances.

Lastly, transactions can affect multiple shards [44], [58], [66]–[68], [71]. This occurs when a transaction requires data from multiple shards. As such, these shards must coordinate access to ensure consistency. The system then has to coordinate cross-shard transactions to make sure the transaction is accepted by each shard. Systems like RapidChain handle cross-shard transactions by splitting them into multiple sub-transactions that get handled by the respective shards [44]. This is possible as RapidChain uses the Unspent Transaction Output (UTXO) state model, which allows splitting transactions in the spending of existing outputs and the creation of new outputs. RapidChain calculates that more than 96% of transactions in their system are cross-shard transactions. As an optimization, Pyramid, which uses an account model, proposes to overlap shards, allowing cross-shard transactions to be processed by the nodes overlapping both shards [67]. This way, overlapping nodes efficiently process cross-shard transactions, requiring no additional work. The overlapping nodes generate blocks from the transactions and send them to the rest of the shard for commitment. In Cerberus [66], each shard locally reaches consensus on UTXO transactions. If input from other shards is necessary, each shard will send its own local input to the other affected shards, thereby promising to process the transaction once all the required input is available by the other shards. As all shards do the same procedure, they will either receive all required inputs and process the transaction, or all abort the transaction. In SharPer [68], which also uses an account model, cross-shard transactions are handled by clients sending the transactions to some leader of a shard from which data is needed. This leader is then responsible for sending the transaction to all nodes of all other involved shards.

2) Hierarchical Consensus: In a hierarchical consensus scheme, nodes on lower hierarchy levels can operate in independent groups and use higher levels to coordinate. The papers in this section use different approaches to improve scalability. The biggest benefit, as with sharding, is that the smaller groups allow for more efficient communication, as the number of communicating nodes is reduced [54], [73], [74].

This architecture poses multiple questions. Firstly, how is the hierarchy structured, and how is it determined? How many layers are there, and what information has to get shared between layers? Finally, and most importantly for scalability, what degree of independence can lower levels have, and when are higher levels of consensus necessary?

Regarding the hierarchy structure, of the papers considered,

four picked a two-layer hierarchy [61], [73]–[75], while two chose a variable number of layers [39], [54]. For the two-layer structure, a structure using "representatives" is most common [61], [73], [75], which is also used by X-Layer [39] for multiple layers. In this case, as seen in Fig. 9b, the lower levels have their "representative", commonly leaders, representing the group in the upper layer. Similarly, GeoBFT [45] shares transactions globally by sending them to the leaders of other clusters before executing them. The difference is that global sharing does not require coordination between the leaders, but the leaders share the transaction within their own cluster. In another way, DP-Hybrid [74] chooses to use PoW as the mechanism for the upper layer where all nodes can participate. Focusing on processing transactions in wide area networks, Saguaro [54] uses multiple layers, with different layers composed of edge devices, edge servers, fog servers, and cloud servers. The structure of hierarchies can be dynamic [61], [75], where new shards are created if enough new nodes join the network, or static as in the cases of DP-Hybrid [74] and Saguaro [54], where the algorithm is focused on companies using the blockchain.

Next is the question on what information the higher level works. There are two broad approaches to this: either with the actual transactions [45], [74]–[76], or with "prepackaged" blocks of transactions [54], [73] arranged in the upper layer. For example, in GeoBFT [45] or DP-Hybrid [74], every transaction is shared through the upper layer with the other groups. In C-PBFT [73], on the other hand, the lower layer already constructs blocks of transactions, which are then confirmed by adding the block header to the upper-layer blockchain.

As with sharding, it can happen that some transaction requires data from multiple groups. For this, affected groups can coordinate with their least common ancestor group [54]. As all transactions are globally shared in GeoBFT [45], each cluster has access to all transactions, making cross-cluster transactions cheap. As global sharing is performed before execution, this leads to higher latencies.

Different layers can have different failure modes [54], [76], e. g., when different applications use a shared blockchain. This can be used to optimize the system, e. g., if the applications require different security or fault tolerance levels. Then it might be possible to use different consensus protocols in separate lower layers, such as BFT or crash fault-tolerant protocols.

Further, there are protocols that measure the behavior of nodes and reward good behavior, e. g., to allow well-acting nodes to become leaders more often [61], [73]. The behavior-measuring metrics consist of the detection of misbehaving nodes [61] and qualitative measures like payment times and amounts [73]. "Good" participants are thus rewarded for their good metrics, creating incentives for good behavior.

E. Selection of Consensus Committees

As described in §IV-A, one approach to improve scalability is to avoid bottleneck situations. In this direction, several approaches regarding nodes' roles for agreement exist. Of the 20 papers in this section, 10 focus on consensus by committee, eight on hierarchical consensus (c. f., §IV-D2), and two on randomized sampling. One approach is that only a substantially smaller subset of participants, i. e., committees, participate in finding agreement [13], [35], [44], [46]. As these committees are significantly smaller than the whole set of participants, consensus protocols—which, in theory, would not scale for the whole set of nodes—become sufficiently efficient again. The remaining nodes often take a passive role and only observe and verify the committee's work. A related technique is a hierarchical consensus where multiple layers are used for consensus [39], [54], [73], [75], [76], which faces similar problems as consensus by committee. In this section, we investigate protocols using these techniques to see how committees are formed, how they work, and what aspects should be considered for scalability.

1) Committee: In a committee, only a subset of nodes participates in the consensus algorithm, as depicted in Fig. 9c, while non-committee members observe the results of the consensus [13], [46], [77]. As only a substantially smaller subset of the total nodes participate in the consensus algorithm, the communication efforts needed inside of the committee are reduced. The crucial question is: how is it decided which participants will be included in the committee without being vulnerable to attacks? The main committee formation scheme is based on randomness. Randomness makes it unlikely that a critical amount of malicious nodes will enter the committee. Different algorithms use different randomness to select committee nodes. For example, Algorand utilizes VRFs [13] (c. f., §IV-C5), DRBFT picks committee nodes based on previous blocks [77], and RapidChain uses verifiable secret sharing to generate randomness within committees [44]. Some algorithms modify the committee formation based on metrics such as stake [13], [63] or quality characteristics [62] such as bandwidth or latency to prioritize nodes by some weight. In permissionless protocols like Algorand, the stake is necessary to prevent Sybil attacks based on pseudonyms. Otherwise, attackers could create an arbitrary number of nodes to gain control of the committee to influence the consensus maliciously. The weighting of quality metrics incentivizes nodes to provide more efficient services to become committee nodes. Thus, preferred nodes are presumably better committee members than randomly selected ones.

Malicious committee members pose a risk to the safety and liveness of protocols. Inside the committee, BFT protocols are used to tolerate malicious behavior. Additionally, some algorithms use different mechanisms to reduce the risk of a malicious committee. Commonly the results of the committee are verified. This verification is both to check that progress is made, e. g., that blocks are proposed, but also of the blocks themselves [35], [77]. For example, in Proteus, a committee is replaced with new members if the committee does not generate valid blocks [35]. To encourage participation, committee members who act maliciously risk forfeiture of their stake in protocols requiring a deposit to join the committee [63].

By definition, committees consist of only a subset of the total nodes. This opens up the risk that an adversary could gain control of a committee by chance or by adaptive corruptions

while controlling less than $\frac{1}{3}$ of the total nodes. One way to overcome this risk of adaptive corruption is to change out the committee members regularly [78]. For example, in Algorand [13] any committee member is replaced after any message sent by the member. Thus, committee members are immediately replaced once identified as potential victims, making it impossible for attackers to target committee members. In contrast, for Dumbo [56] or in PoQ [62], committees are persistent for one block generation. This makes committee members vulnerable but limits the attack impact to one block. Similarly, in AHL each committee is replaced after every epoch [38]. This is assumed to be safe even in the presence of an adaptive attacker model where nodes are not instantly corrupted but rather after some time. RapidChain [44] also replaces committee members after every epoch, though as an optimization, it replaces only a subset of committee members. AHL and RapidChain make limitations on the threat model. AHL relies on TEEs, which are assumed to only fail by crashing. RapidChain assumes that the adaptive adversary can only corrupt nodes at the start of the protocol and in between epochs, but not within an epoch. In Hermes [46], which does not assume an adaptive adversary, committees can get replaced by view changes. Non-committee members initiate a view change after not receiving a block over some period of time, after receiving an invalid block proposal, or after multiple proposals with the same sequence number.

2) Randomized Sampling: We have seen how leaders can become the bottleneck regarding scalability. Randomized sampling can be used to create leaderless consensus protocols, thus avoiding bottlenecks at a single leader [14], [43]. With randomized sampling, nodes only communicate with a subset of the total nodes. In the process, the nodes exchange information locally about their values. These values can be the value a node has locally decided as in [14], or DecicionVectors as in [43] where the decisions of all nodes are exchanged. A global consensus is achieved by running these local updates multiple times so that, over time the nodes converge to a single decision. With these local updates, there is no need for a leader to drive the consensus forward. This removes the common bottleneck in other consensus protocols, as no single node is responsible for disseminating or collecting any information to and from all other nodes. The required communication stays constant for each node as it only needs to exchange information with a configurable but constant subset of local nodes [14], [43]. Nodes learn of proposed values from others. In Avalanche, this happens after the node queries neighboring nodes, while in [43], nodes can also actively push their value to other nodes. However, they only provide probabilistic safety guarantees.

F. Hardware Support

The scalability of BFT protocols can also be improved via hardware support as offered by trusted execution environments (TEEs), as has been done in two papers matching our criteria: FastBFT [40] and AHL [38]. The most commonly used TEE is Intel's Software Guard Extensions (SGX) due to its high availability; however, many approaches are independent of the underlying TEE, allowing the use of e. g., Trusted Platform Modules (TPMs), ARM TrustZone, or AMD SEV-SNP. SGX is an x86 instruction set extension that allows the creation of so-called *enclaves*, in which confidentiality of execution and integrity of data is ensured from privileged software via hardware-based memory encryption and checksums. Enclaves can be remotely attested: users can verify an enclave's code and data, and before execution of the enclave, the provided code and data is hashed to create a measurement. This measurement can then be compared against the value of the verified enclave to ensure that the user communicates with a genuine, correct, and unmodified version of the expected enclave.

TEEs can therefore be used as a trusted subsystem in BFT protocols with a *hybrid* fault model: while the remainder of the BFT system can still behave arbitrarily faulty, the trusted subsystem is assumed to behave correctly and can only fail by crashing. This can be used to prevent equivocation, i. e., sending conflicting messages to different communication partners in the protocol. Primitives that make use of such trusted subsystems are e. g., trusted counters [79] or attested append-only memory [80]. FastBFT [40] employs trusted monotonic counters that are provided by the TEE running on the leader replica. A counter value extends every message sent by the leader, and as every value can only be used once and the counter is monotonically increasing, it can thus be detected if the leader equivocates. Relying on trusted hardware and therefore preventing equivocation allows reducing the complexity of BFT protocols, e. g., by decreasing the number of replicas from $3f + 1$ to $2f + 1$ or the required communication rounds for agreement, or by using less expensive cryptographic primitives (see also §IV-C).

TEEs are also used to efficiently aggregate messages [38], [40], e. g., by combining a quorum of $2f + 1$ messages into a proof issued by the TEE [38]. Here, the leader collects and aggregates other nodes' signatures into a single authenticated message, while nodes forward their signed messages to the leader and verify the created multi-signature. Furthermore, TEEs can also be used as a source for a trusted randomness beacon, which can be used to efficiently partition the system for sharding [38] (see also §IV-D1).

G. Summary of BFT Protocols

All discussed protocols are shown in Tab. II. We list their assumptions regarding the synchrony model, the number of faults that can be tolerated, membership of nodes (i. e., whether nodes are static or can dynamically join or leave the network), and the guarantees for safety and liveness. Further, we give an overview of which scaling techniques have been employed and combined in the protocols. PBFT, the starting point for many protocols, assumes a partially synchronous network of static nodes, whereas many of the blockchain BFT protocols target a dynamic network of nodes or the asynchronous model. Many protocols replace PBFT's single leader with multiple leaders to share the load and reduce its bottleneck, or with a leaderless approach. Not all blockchain BFT protocols keep PBFT's clique communication; we take clique as the default and list in

TABLE II: Comparison of scalable BFT protocols regarding their assumptions, goals, and scaling techniques.

BFT Protocol	Assumptions			Guarantees		Scaling Techniques						
	Network (for safety+liveness)	Fault	Members	Safety	Liveness	Leader	Crypto	Message Exchange	Pipelining Strategy	Consensus	Parallel-ization	Other
ByzCoin [30]	part. sync	PoW bound	dynamic	det.	det.	single PoW-elected	multi sig.	tree	-	-	-	-
Lim et al. [43]	async	$f < \frac{1}{2}n$	static	prob.	prob.	none	-	gossip	-	rand. sampling	-	-
Tendermint [11]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	prob.	single rotating	-	gossip	-	-	-	-
RepChain [49]	sync	$f < \frac{1}{3}n$	dynamic	prob.	prob.	single	multi sig.	-	-	-	sharding	-
Ostraka [81]	conf.	$f < \frac{1}{2}n$	dynamic	conf.	conf.	conf.	-	-	-	-	sharding	-
Mitosis [82]	part. sync	paramet.	dynamic	conf.	conf.	conf.	-	-	-	-	sharding	-
Kauri [17]	part. sync	$f < \frac{1}{3}n$	static	det.	det.	single	multi sig.	tree	configurable pipe-lining stretch	-	-	-
HotStuff [9]	part. sync	$f < \frac{1}{3}n$	static	det.	det.	single rotating	threshold sig.	star	one pipeline stage per protocol stage	-	-	-
X-Layer PBFT [39]	part. sync	paramet.	static	prob.	prob.	single	-	tree	-	hierarchical	-	-
CHECO [28]	async	$f < \frac{1}{3}n$	dynamic	prob.	prob.	multi	-	-	-	-	-	-
Beh-Raft-Chain [61]	sync	$f < \frac{1}{2}n$	dynamic	prob.	prob.	single	VRF	-	-	hierarchical	sharding	-
SHBFT [75]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	det.	single multi	-	-	-	hierarchical	hierarchical	-
Hermes [46]	part. sync	$f < \frac{1}{3}n$	dynamic	prob.	prob.	single	-	-	chain-based pipelining	committee	-	-
SharPer [68]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	det.	single	-	-	-	-	sharding	-
FastBFT [40]	part. sync	$f < \frac{1}{2}n$	dynamic	det.	det.	single	secret sharing	tree	-	-	-	HW/TEE
ResilientDB [45]	sync	$f < \frac{1}{3}n$ per cluster	static	det.	det.	multi	-	-	c.f. PoE	-	clustering	-
BFT-Store [83]	part. sync	paramet.	dynamic	det.	det.	single rotating	erasure coding	-	-	-	-	storage scalability
Red Belly [16]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	det.	none	-	-	-	-	sharding	-
RCC [47]	part. sync	$f < \frac{1}{3}n$	static	det.	det.	multi	-	-	pipelining blocks across rounds	-	-	-
RapidChain [44]	sync	$f < \frac{1}{3}n$	dynamic	prob.	prob.	single random select.	-	gossip	intrashard pipelining	committee	sharding	-
Pyramid [67]	part. sync	$f < \frac{1}{3}n$	dynamic	prob.	prob.	multi	-	-	-	-	sharding	-
Proteus [35]	async	$f < \frac{1}{2}n$	static	prob.	prob.	single	-	star	-	committee	-	-
Proof-of-QoS [62]	part. sync	$f < \frac{1}{3}n$ per committee	dynamic	det.	det.	single	VRF	-	-	committee	-	-
Proof-of-Execution [36]	part. sync	$f < \frac{1}{3}n$	static	det.	det.	single	threshold sig.	star	Out-of-order processing	speculative execution	-	-
Musch [34]	part. sync	$f < \frac{1}{2}n$	static	det.	det.	single	multi sig.	star	-	-	-	-
AHL [38]	part. sync	$f < \frac{1}{3}n$	static	prob.	prob.	single	multi sig.	star	-	committee	sharding	HW/TEE
Dumbo [56]	async	$f < \frac{1}{3}n$	static	prob.	prob.	multi	erasure coding threshold sig.	-	-	committee	-	-
DP-Hybrid [74]	part. sync	paramet.	dynamic	prob.	prob.	none single	-	-	-	hierarchical	hierarchical	-
DLattice [63]	sync	$f < \frac{1}{2}n$	dynamic	prob.	prob.	none	VRF	-	-	committee	-	-
DBFT [32]	part. sync	$f < \frac{1}{3}n$	static	det.	det.	none	-	-	-	-	-	-
Cumulus [55]	part. sync	$f < \frac{1}{3}n$	dynamic	prob.	prob.	single	threshold sig. VRF	-	-	-	-	-
Chainspace [70]	async	$f < \frac{1}{2}n$	dynamic	prob.	prob.	single	-	-	-	-	sharding	-
CAPER [76]	async	$f < \frac{1}{3}n$	static	det.	det.	single	-	-	-	hierarchical	-	-
BlockTree [84]	async	$f < \frac{1}{2}n$	dynamic	prob.	prob.	single	-	-	-	-	sharding	-
Dispel [33]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	det.	none	-	-	*distributed* pipeline	-	-	-
BigBFT [27]	part. sync	$f < \frac{1}{3}n$	static	det.	det.	multi	multi sig.	-	pipelining blocks across rounds	-	-	-
Gosig [41]	part. sync	$f < \frac{1}{3}n$	static	det.	prob.	random select.	multi sig.	gossip	pipelines gossiplayer *and* BFT protocol	-	-	-
RingBFT [72]	part. sync	$f < \frac{1}{3}n$	static	det.	det.	single	-	ring	-	-	sharding	-
DispersedLedger [59]	async	$f < \frac{1}{2}n$	static	det.	det.	none	erasure coding	-	-	-	-	-
Saguaro [54]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	det.	single	threshold sig.	-	-	hierarchical	hierarchical	-
Qanaat [58]	part. sync	$f < \frac{1}{4}n$	static	det.	det.	single	secret sharing	-	-	-	sharding	-
Narwhal-HotStuff [48]	part. sync	$f < \frac{1}{2}n$	static	det.	det.	rotating	threshold sig.	star	c.f. HotStuff	-	-	-
Tusk [48]	async	$f < \frac{1}{3}n$	static	prob.	prob.	rotating	threshold sig.	star	c.f. HotStuff	-	-	-
Avalanche [14]	part. sync	paramet.	dynamic	prob.	prob.	none	-	gossip	-	rand. sampling	-	-
C-PBFT [73]	part. sync	$f < \frac{1}{2}n$	static	det.	det.	single	-	-	-	hierarchical	hierarchical	-
DRBFT [77]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	det.	single	-	-	-	committee	-	-
SBFT [10]	sync	$n = 3f + 2c + 1$	static	det.	det.	single	threshold sig.	star	learning heuristic for dynamic batching	-	-	-
Cerberus [66]	async	$f < \frac{1}{3}n$	static	det.	det.	single	-	-	-	-	sharding	-
GearBox [69]	part. sync	$f < \frac{1}{2}n$	dynamic	det.	prob.	single	-	-	-	committee	sharding	-
ByShard [71]	conf.	$f < \frac{1}{3}n$	conf.	conf.	conf.	single	-	-	-	hierarchical	sharding	-
ICC0 / ICC1 [42]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	det.	single rotating	threshold sig.	gossip	-	-	-	-
ICC2 [42]	part. sync	$f < \frac{1}{3}n$	dynamic	det.	det.	single rotating	threshold sig. erasure coding	gossip	-	-	-	-
Jolteon [37]	part. sync	$f < \frac{1}{3}n$	static	det.	det.	single	threshold sig.	star	c.f. HotStuff	-	-	-
Ditto [37]	async	$f < \frac{1}{3}n$	static	prob.	prob.	single	threshold sig.	star	c.f. HotStuff	-	-	-
Algorand [13]	sync	$f < \frac{1}{3}n$	dynamic	prob.	prob.	random select.	VRF	gossip	-	committee	-	-

Tab. II which protocols deviate from this pattern. While many protocols target improving scalability within the consensus group while increasing the number of nodes, protocols tagged as "committee", "sharding", and "hierarchical" generally target scalability and performance improvements by using subsets of nodes participating in consensus. Some listed approaches are generic frameworks that are not fixed to one specific protocol, e. g., RCC [47], Ostraka [81], Mitosis [82], RingBFT [72], and ByShard [71]. Here, we list assumptions and guarantees if explicitly stated in the corresponding papers or list them as configurable if applicable.

V. Related Work

Several surveys systematically analyze blockchain protocols; some contain a more or less detailed treatment of scalability.

Alsunaidi et al. created a survey of blockchain consensus algorithms, focusing on performance and security [85]. The authors distinguish between *proof-based* (e. g., PoW) and *voting-based* (e. g., BFT). Scalability is a mentioned challenge, but is not analyzed beyond categorizing proof-based protocols as "strongly" and vote-based protocols as "weakly" scalable.

Bano et al. presented an SoK about blockchain consensus protocols [86], where the main contribution is a systematization framework that tracks the chronological evolution of blockchain consensus protocols and a categorization using this framework. While the framework explains that consensus scalability can be achieved by advancing from *hybrid single committee consensus* to *hybrid multiple committee consensus* (e. g., through sharding), it does not treat scalability mechanics for consensus in general, e. g., for consensus within a single committee.

Berger et al. created a short survey in 2018 that broadly analyzes scalability techniques used in BFT consensus protocols [87]. Possibly, our survey can be best understood as progressing this effort by (1) using a systematic methodology, (2) increasing the level of detail, and (3) applying the analysis to contemporary research works, thus extending the scope by many papers that have been published just recently.

Ferdous et al.'s survey on blockchain consensus introduced a taxonomy of desirable properties [88]. While scalability is one such property, the technical aspects are not discussed.

Hafid et al. analyze the scalability of blockchain platforms with a focus on first and second-layer solutions [89], i. e., changes to the blockchain, e. g., the block structure using DAGs or increasing block sizes, and mechanisms implemented outside of it, e. g., side-chains, child-chains, or payment channels. The authors propose a taxonomy based on committee formation and consensus within a committee and compare sharding-based protocols. Scalability outside of sharding is not considered.

Huang et al. analyze blockchain surveys and focus on theoretical modeling, analysis models, performance measurements, and experiment tools [90]. Scalability is only considered with regard to sharding and multi-chain interoperability.

Jennath et al. give a general overview of common blockchain consensus protocols, such as PoW, PoS, Proof-of-Elapsed-Time (PoET), BFT, and Federated Byzantine agreements [91]. Lao et al. consider IoT blockchains and their consensus strategies [92], for which they compare consensus protocols using similar categories. Neither survey considers BFT scalability.

Liu et al. analyze recent blockchain techniques and claim that consensus-based scaling is limited, especially with Moore's law nearing its end [93]. They discuss and evaluate scaling concerning topology and hardware assistance, e. g., off-chain or parallel-chain computations or sharding.

Meneghetti et al. presented a survey on blockchain scalability [94]; however, they focus more on smart contract executions, particularly sharding, than on the consensus mechanism.

Monrat et al.'s survey on blockchain applications, challenges, and opportunities [95] provides a blockchain taxonomy and describes potential applications. The authors also describe common consensus algorithms but do not focus on scalability.

Salimitari and Chatterjee created a survey on blockchain consensus protocols in IoT [96]. They evaluate various blockchain consensus protocols for use in IoT scenarios. The survey categorizes consensus protocols in rough categories such as PoW, PoS, BFT, VRF-based, and sharding-based solutions.

Vukolić contrasts PoW-based algorithms to BFT SMR protocols [4]. Vukolić identifies scalability to many consensus nodes as a blocker for the adoption of blockchain consensus. As of 2016, Vukolić identifies optimistic BFT protocols and relaxed fault models such as XFT or hybrid fault models with trusted hardware as potential solutions.

VI. Conclusions and Open Challenges

One of the major ongoing challenges in the field of blockchain is making Byzantine consensus applicable to large-scale environments. To address this challenge, a large body of research has focused on developing novel techniques to improve the scalability of BFT consensus, paving the way for a new generation of BFT protocols tailored to the needs of blockchain. In this SoK paper, we employed a systematic literature search to explore the design space of recent BFT protocols along with their ideas for scaling up to hundreds or thousands of nodes. We created a taxonomy of scalability-enhancing techniques, which categorizes these ideas into communication and coordination strategies, pipelining, cryptographic primitives, independent groups, committee selection, and trusted hardware support. As shown in Tab. II, many BFT protocols employ not only one idea but rather a combination of several ideas. We also see that a less vigorously explored research field seems to be the incorporation of trusted execution environments, which is inviting for future research works. Further, we comprehensively discussed all ideas on an abstract level and pinpointed the design space from which their corresponding BFT protocols originated.

Some open challenges regarding BFT scalability remain: While we have identified and categorized these scalability techniques, we cannot compare their *effectiveness* solely on the basis of the papers' evaluation results, and a common evaluation platform for these protocols is not yet available [97]. Tab. II shows a wide range of combinations of techniques that have so far been applied; however, *further combinations* may exist that lead to valid, performant, and highly scalable protocols and which have to be identified. Not only the techniques' effectiveness is important, but their *complexity* regarding computational resource requirements, implementation effort, or proof of correctness in a protocol is also relevant as well and may differ widely. Finally, depending on the specific *application requirements*, different agreement protocols may be more suitable for specific deployment settings than others. As the BFT protocol landscape is extensive, developing a guideline for selecting the most fitting protocol according to these requirements may be helpful.

Acknowledgments

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant number 446811880 (BFT2Chain).

REFERENCES

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] V. Buterin. (2014) A Next-Generation Smart Contract and Decentralized Application Platform. https://ethereum.org/en/whitepaper/.

[3] J. R. Douceur, "The Sybil Attack," in *Int. Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 251–260.

[4] M. Vukolić, "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication," in *Int. Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.

[5] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*, 1999, pp. 173–186.

[6] A. Bessani, J. Sousa, and E. E. Alchieri, "State Machine Replication for the Masses with BFT-SMART," in *2014 44th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, 2014, pp. 355–362.

[7] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[8] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT Consensus in the Lens of Blockchain," *arXiv preprint arXiv:1803.05069*, 2018.

[9] M. Yin, D. Malkhi, M. Reiter, G. Gueta, and I. Abraham, "HotStuff: BFT Consensus with Linearity and Responsiveness," in *ACM Symp. on Principles of Distributed Computing (PODC)*, 2019, pp. 347–356.

[10] 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 *49th Annu. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2019, pp. 568–580.

[11] E. Buchman, "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains," Ph.D. dissertation, University of Guelph, 2016.

[12] D. Cason, E. Fynn, N. Milosevic, Z. Milosevic, E. Buchman, and F. Pedone, "The Design, Architecture and Performance of the Tendermint Blockchain Network," in *40th Int. Symp. on Reliable Distributed Systems (SRDS)*, 2021, pp. 23–33.

[13] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *26th Symp. on Operating Systems Principles (SOSP)*, 2017, pp. 51–68.

[14] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and Probabilistic Leaderless BFT Consensus through Metastability," *arXiv:1906.08936 [cs]*, Aug. 2020, arXiv: 1906.08936. [Online]. Available: http://arxiv.org/abs/1906.08936

[15] C. Stathakopoulou, T. David, and M. Vukolić, "Mir-BFT: High-Throughput BFT for Blockchains," *arXiv preprint arXiv:1906.05552*, 2019.

[16] T. Crain, C. Natoli, and V. Gramoli, "Red Belly: A Secure, Fair and Scalable Open Blockchain," in *IEEE Symp. on Security and Privacy (SP)*, 2021, pp. 466–483.

[17] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable BFT Consensus with Pipelined Tree-Based Dissemination and Aggregation," in *28th Symp. on Operating Systems Principles (SOSP)*, 2021, pp. 35–48.

[18] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 228–234, 1980.

[19] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, Jul. 1982.

[20] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[21] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with one Faulty Process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[22] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, p. 288–323, apr 1988.

[23] M. Castro, "Practical Byzantine Fault Tolerance," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2000.

[24] N. A. Lynch, *Distributed Algorithms*. Elsevier, 1996.

[25] F. Borran and A. Schiper, "A Leader-Free Byzantine Consensus Algorithm," in *Int. Conf. on Distributed Computing and Networking*. Springer, 2010, pp. 67–78.

[26] K. Antoniadis, A. Desjardins, V. Gramoli, R. Guerraoui, and I. Zablotchi, "Leaderless Consensus," in *IEEE 41st Int. Conf. on Distributed Computing Systems (ICDCS)*, 2021, pp. 392–402.

[27] S. Alqahtani and M. Demirbas, "BigBFT: A Multileader Byzantine Fault Tolerance Protocol for High Throughput," in *IEEE Int. Performance, Computing, and Communications Conf. (IPCCC)*, 2021, pp. 1–10.

[28] K. Cong, Z. Ren, and J. Pouwelse, "A Blockchain Consensus Protocol with Horizontal Scalability," in *2018 IFIP Networking Conf. (IFIP Networking) and Workshops*. IEEE, 2018, pp. 1–9.

[29] "Semantic Scholar," Accessed at: 2022-01-10. [Online]. Available: https://www.semanticscholar.org/

[30] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing," in *25th USENIX Security Symp. (USENIX Security)*, 2016, pp. 279–296.

[31] E. Buchman, J. Kwon, and Z. Milosevic, "The Latest Gossip on BFT Consensus," *arXiv preprint arXiv:1807.04938*, 2018.

[32] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "DBFT: Efficient Leaderless Byzantine Consensus and its Application to Blockchains," in *2018 IEEE 17th Int. Symp. on Network Computing and Applications (NCA)*, 2018, pp. 1–8.

[33] G. Voron and V. Gramoli, "Dispel: Byzantine SMR with Distributed Pipelining," *arXiv preprint arXiv:1912.10367*, 2019.

[34] M. M. Jalalzai and C. Busch, "Window Based BFT Blockchain Consensus," in *2018 IEEE Int. Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 971–979.

[35] M. M. Jalalzai, C. Busch, and G. G. Richard, "Proteus: A Scalable BFT Consensus Protocol for Blockchains," in *2019 IEEE Int. Conf. on Blockchain (Blockchain)*, 2019, pp. 308–313.

[36] S. Gupta, J. Hellings, S. Rahnama, and M. Sadoghi, "Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation," in *24th Int. Conf. on Extending Database Technology EDBT*, 2021, pp. 301–312.

[37] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang, "Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback," in *Financial Cryptography and Data Security*, I. Eyal and J. Garay, Eds. Cham: Springer, 2022, pp. 296–315.

[38] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards Scaling Blockchain Systems via Sharding," in *2019 Int. Conf. on Management of Data*. ACM, Jun. 2019, pp. 123–140.

[39] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, "A Scalable Multi-Layer PBFT Consensus for Blockchain," *IEEE Trans. on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1146–1160, 2021.

[40] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing," *IEEE Trans. on Computers*, vol. 68, no. 1, pp. 139–151, 2018.

[41] P. Li, G. Wang, X. Chen, F. Long, and W. Xu, "Gosig: A Scalable and High-Performance Byzantine Consensus for Consortium Blockchains," in *11th ACM Symp. on Cloud Computing*, 2020, pp. 223–237.

[42] J. Camenisch, M. Drijvers, T. Hanke, Y.-A. Pignolet, V. Shoup, and D. Williams, "Internet Computer Consensus," in *ACM Symp. on Principles of Distributed Computing (PODC)*, 2022, pp. 81–91.

[43] J. Lim, T. Suh, J. Gil, and H. Yu, "Scalable and Leaderless Byzantine Consensus in Cloud Computing Environments," *Information Systems Frontiers*, vol. 16, no. 1, pp. 19–34, Mar. 2014.

[44] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *2018 ACM SIGSAC Conf. on Computer and Communications Security*, 2018, pp. 931–948.

[45] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, "ResilientDB: Global Scale Resilient Blockchain Fabric," *Proc. of the VLDB Endowment*, vol. 13, no. 6, pp. 868–883, 2020.

[46] M. M. Jalalzai, C. Feng, C. Busch, G. Richard III, and J. Niu, "The Hermes BFT for Blockchains," *IEEE Trans. on Dependable and Secure Computing*, 2021.

[47] S. Gupta, J. Hellings, and M. Sadoghi, "RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing," in *IEEE 37th Int. Conf. on Data Engineering (ICDE)*, 2021, pp. 1392–1403.

[48] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: A DAG-Based Mempool and Efficient BFT Consensus," in *17th European Conf. on Computer Systems (EuroSys)*, 2022, p. 34–50.

[49] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan, "RepChain: A Reputation-Based Secure, Fast, and High Incentive Blockchain System via Sharding," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4291–4304, 2021.

[50] D. Boneh, M. Drijvers, and G. Neven, "Compact Multi-Signatures for Smaller Blockchains," Cryptology ePrint Archive, Paper 2018/483, 2018.

[51] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Springer, 2001, pp. 514–532.

[52] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning," in *Symp. on Security and Privacy (SP)*, 2016, pp. 526–545.

[53] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs, "On the Security of Two-Round Multi-Signatures," in *Symp. on Security and Privacy (SP)*, 2019, pp. 1084–1101.

[54] M. J. Amiri, Z. Lai, L. Patel, B. T. Loo, E. Lo, and W. Zhou, "Saguaro: Efficient Processing of Transactions in Wide Area Networks using a Hierarchical Permissioned Blockchain," *arXiv preprint arXiv:2101.08819*, 2021, arXiv: 2101.08819.

[55] F. Gai, J. Niu, S. Ali Tabatabaee, C. Feng, and M. Jalalzai, "Cumulus: A Secure BFT-based Sidechain for Off-chain Scaling," in *IEEE/ACM 29th Int. Symp. on Quality of Service (IWQOS)*, 2021, pp. 1–6.

[56] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster Asynchronous BFT Protocols," in *ACM SIGSAC Conf. on Computer and Communications Security*, 2020, pp. 803–818.

[57] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The Honey Badger of BFT Protocols," in *ACM SIGSAC Conf. on Computer and Communications Security*, 2016, pp. 31–42.

[58] M. J. Amiri, B. T. Loo, D. Agrawal, and A. E. Abbadi, "Qanaat: A Scalable Multi-Enterprise Permissioned Blockchain System with Confidentiality Guarantees," *arXiv preprint arXiv:2107.10836*, 2021.

[59] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, "DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks," in *19th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2022, pp. 493–512.

[60] S. Micali, M. Rabin, and S. Vadhan, "Verifiable Random Functions," in *40th Annu. Symp. on Foundations of Computer Science*, 1999, pp. 120–130.

[61] L.-e. Wang, Y. Bai, Q. Jiang, V. C. M. Leung, W. Cai, and X. Li, "Beh-Raft-Chain: A Behavior-Based Fast Blockchain Protocol for Complex Networks," *IEEE Trans. on Network Science and Engineering*, vol. 8, no. 2, pp. 1154–1166, 2021.

[62] B. Yu, J. Liu, S. Nepal, J. Yu, and P. Rimba, "Proof-of-QoS: QoS Based Blockchain Consensus Protocol," *Computers & Security*, vol. 87, 2019.

[63] T. Zhou, X. Li, and H. Zhao, "DLattice: A Permission-Less Blockchain Based on DPoS-BA-DAG Consensus for Data Tokenization," *IEEE Access*, vol. 7, pp. 39 273–39 287, 2019.

[64] S. Goldberg, M. Naor, D. Papadopoulos, and L. Reyzin, "NSEC5 from Elliptic Curves: Provably Preventing DNSSEC Zone Enumeration with Shorter Responses," Cryptology ePrint Archive, Paper 2016/083, 2016.

[65] S. Goldberg, D. Papadopoulos, and J. Včelák, "Verifiable Random Functions (VRFs)," IETF, Internet-Draft draft-goldbe-vrf-00, 2017.

[66] J. Hellings, D. Hughes, J. Primero, and M. Sadoghi, "Cerberus: Minimalistic Multi-Shard Byzantine-Resilient Transaction Processing," *arXiv preprint arXiv:2008.04450*, 2020.

[67] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A Layered Sharding Blockchain System," in *IEEE Conference on Computer Communications (INFOCOM)*, 2021, pp. 1–10.

[68] M. J. Amiri, D. Agrawal, and A. El Abbadi, "SharPer: Sharding Permissioned Blockchains Over Network Clusters," in *Int. Conf. on Management of Data*. ACM, 2021, pp. 76–88.

[69] B. David, B. Magri, C. Matt, J. Nielsen, and D. Tschudi, "GearBox: An Efficient UC Sharded Ledger Leveraging the Safety-Liveness Dichotomy," Cryptology ePrint Archive, Paper 2021/211, 2021.

[70] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A Sharded Smart Contracts Platform," in *2018 Network and Distributed System Security Symp.* Internet Society, 2018.

[71] J. Hellings and M. Sadoghi, "ByShard: Sharding in a Byzantine Environment," in *Proc. of the VLDB Endowment*, vol. 14, 2021, pp. 2230–2243.

[72] S. Rahnama, S. Gupta, R. Sogani, D. Krishnan, and M. Sadoghi, "RingBFT: Resilient Consensus over Sharded Ring Topology," *arXiv preprint arXiv:2107.13047*, 2021.

[73] X. Xu, D. Zhu, X. Yang, S. Wang, L. Qi, and W. Dou, "Concurrent Practical Byzantine Fault Tolerance for Integration of Blockchain and Supply Chain," *ACM Trans. on Internet Technology*, vol. 21, no. 1, pp. 1–17, 2021.

[74] F. Wen, L. Yang, W. Cai, and P. Zhou, "DP-Hybrid: A Two-Layer Consensus Protocol for High Scalability in Permissioned Blockchain," in *Blockchain and Trustworthy Systems*, Z. Zheng, H.-N. Dai, X. Fu, and B. Chen, Eds. Springer, 2020, vol. 1267, pp. 57–71.

[75] Y. Li, L. Qiao, and Z. Lv, "An Optimized Byzantine Fault Tolerance Algorithm for Consortium Blockchain," *Peer-to-Peer Networking and Applications*, vol. 14, no. 5, pp. 2826–2839, 2021.

[76] M. J. Amiri, D. Agrawal, and A. E. Abbadi, "CAPER: A Cross-Application Permissioned Blockchain," *Proc. of the VLDB Endowment*, vol. 12, no. 11, pp. 1385–1398, 2019.

[77] Y. Zhan, B. Wang, R. Lu, and Y. Yu, "DRBFT: Delegated Randomization Byzantine Fault Tolerance Consensus Protocol for Blockchains," *Information Sciences*, vol. 559, pp. 8–21, 2021.

[78] C. Matt, J. B. Nielsen, and S. E. Thomsen, "Formalizing Delayed Adaptive Corruptions and the Security of Flooding Networks," *Cryptology ePrint Archive, Paper 2022/010*, 2022.

[79] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "TrInc: Small Trusted Hardware for Large Distributed Systems," in *6th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2009.

[80] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested Append-Only Memory: Making Adversaries Stick to Their Word," in *21st ACM SIGOPS Symp. on Operating Systems Principles (SOSP)*, 2007, p. 189–204.

[81] A. Manuskin, M. Mirkin, and I. Eyal, "Ostraka: Secure Blockchain Scaling by Node Sharding," in *IEEE European Symp. on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 397–406.

[82] G. A. Marson, S. Andreina, L. Alluminio, K. Munichev, and G. Karame, "Mitosis: Practically Scaling Permissioned Blockchains," in *Annu. Computer Security Applications Conf.*, 2021, pp. 773–783.

[83] X. Qi, Z. Zhang, C. Jin, and A. Zhou, "A Reliable Storage Partition for Permissioned Blockchain," *IEEE Trans. on Knowledge and Data Engineering*, vol. 33, no. 1, pp. 14–27, 2020.

[84] L. Vishwakarma and D. Das, "BlockTree: A Nonlinear Structured, Scalable and Distributed Ledger Scheme for Processing Digital Transactions," *Cluster Comput*, vol. 24, no. 4, pp. 3751–3765, 2021.

[85] S. J. Alsunaidi and F. A. Alhaidari, "A Survey of Consensus Algorithms for Blockchain Technology," in *Int. Conf. on Computer and Information Sciences (ICCIS)*. IEEE, 2019, pp. 1–6.

[86] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "SoK: Consensus in the Age of Blockchains," in *1st ACM Conf. on Advances in Financial Technologies (AFT)*, 2019, p. 183–198.

[87] C. Berger and H. P. Reiser, "Scaling Byzantine Consensus: A Broad Analysis," in *2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers (SERIAL)*, 2018, p. 13–18.

[88] M. S. Ferdous, M. J. M. Chowdhury, M. A. Hoque, and A. Colman, "Blockchain Consensus Algorithms: A Survey," *arXiv preprint arXiv:2001.07091*, 2020.

[89] A. Hafid, A. S. Hafid, and M. Samih, "Scaling Blockchains: A Comprehensive Survey," *IEEE Access*, vol. 8, pp. 125 244–125 262, 2020.

[90] H. Huang, W. Kong, S. Zhou, Z. Zheng, and S. Guo, "A Survey of State-of-the-Art on Blockchains: Theories, Modelings, and Tools," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–42, 2021.

[91] H. S. Jennath and S. Asharaf, "Survey on Blockchain Consensus Strategies," in *ICDSMLA 2019*, A. Kumar, M. Paprzycki, and V. K. Gunjan, Eds. Springer, 2020, pp. 637–654.

[92] L. Lao, Z. Li, S. Hou, B. Xiao, S. Guo, and Y. Yang, "A Survey of IoT Applications in Blockchain Systems: Architecture, Consensus, and Traffic Modeling," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–32, 2020.

[93] Y. Liu, K. Qian, J. Chen, K. Wang, and L. He, "Effective Scaling of Blockchain Beyond Consensus Innovations and Moore's Law," *arXiv preprint arXiv:2001.01865*, 2020.

[94] A. Meneghetti, T. Parise, M. Sala, and D. Taufer, "A Survey on Efficient Parallelization of Blockchain-Based Smart Contracts," *arXiv preprint arXiv:1904.00731*, 2019.

[95] A. A. Monrat, O. Schelén, and K. Andersson, "A Survey of Blockchain from the Perspectives of Applications, Challenges, and Opportunities," *IEEE Access*, vol. 7, pp. 117 134–117 151, 2019.

[96] M. Salimitari and M. Chatterjee, "A Survey on Consensus Protocols in Blockchain for IoT Networks," *arXiv preprint arXiv:1809.05613*, 2018.

[97] M. J. Amiri, C. Wu, D. Agrawal, A. E. Abbadi, B. T. Loo, and M. Sadoghi, "The Bedrock of Byzantine Fault Tolerance: A Unified Platform for BFT Protocol Design and Implementation," *arXiv preprint arXiv:2205.04534*, 2022.