ALDER: Unlocking blockchain performance by multiplexing consensus protocols

Kadir Korkmaz, Joachim Bruneau-Queyreix, Sonia Ben Mokhtar, Laurent Réveillère

ALDER: Unlocking blockchain performance by multiplexing consensus protocols

Kadir Korkmaz
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI
UMR 5800, F-33400 Talence, France

Joachim Bruneau-Queyreix
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI
UMR 5800, F-33400 Talence, France

Sonia Ben Mokhtar
Liris - CNRS
Lyon, France

Laurent Réveillère
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI
UMR 5800, F-33400 Talence, France

***Abstract*—Most of today's online services (*e.g.*, social networks, search engines, marketplace places) are centralized, which most users recognize as unsatisfactory for various reasons (*e.g.*, centralized governance, censorship, loss of control over personal data). Blockchain technologies promise a new Web revolution (Web 3.0) through the decentralization of online services. However, one of the fundamental limitations for this revolution to happen at a planetary scale is the poor performance of today's permissionless blockchains. In this paper, we propose ALDER, a generic construction that multiplexes off-the-shelf permissionless blockchain protocols to address the performance bottleneck due to *store-validate-forward* block dissemination techniques in blockchain protocols. We apply ALDER to two representative blockchains, namely Algorand (Proof-of-Stake) and Bitcoin (Proof-of-Work), to illustrate the benefits it brings to blockchain performance. Our evaluations show that ALDER can drastically improve the throughput of blockchains when bottlenecks exist.**

***Index Terms*—Blockchain systems, performance.**

I. Introduction

Blockchain systems are consistent replicated systems that operate on networks of untrusted nodes that repeatedly agree on a block of data to append to a shared log of chained blocks. Since the emergence of Bitcoin [1], permissionless blockchains have demonstrated the ability to run arbitrary distributed applications [2] with the promise of supporting entire decentralized economies [3], business ecosystems across industries [4], decentralized service infrastructures [5] and even have ambitions to decentralize the entire web [6]. The realization of these latter use cases requires efficient and effective blockchain solutions. From the oldest design (*e.g.*, Bitcoin) to the most recent (*e.g.*, OHIE, Algorand), the race toward more efficient and effective blockchains is still on. Although some blockchain proposals aim to replace others, it is commonly accepted that there is no single blockchain that meets the needs of the wide variety of distributed applications [7]. In this context, solutions that aim to improve the performance of off-the-shelf blockchains make real sense.

A fundamental building block that is often considered a bottleneck in blockchain performance is the consensus protocol used to agree on the next block to extend the blockchain [8]. Significant efforts have been made to enhance the performance of existing consensus protocols that can be used in permissioned blockchains [9], [10], [11] where the number and identity of nodes are known to all. However, improving the performance of existing consensus protocols in permissionless blockchains remains an open problem. There are a variety of these protocols, *e.g.*, Proof-of-Work-based [1], [12], [13], [14], [15], Proof-of-Stake-based [3], [16], committee-based [3], [17], [14], [18] or leadlerless-based protocols [19], [20].

In this paper, we focus on leader-based permissionless blockchain consensus protocols. These protocols, deployed at the heart of major blockchains (*e.g.*, Bitcoin, Ethereum, Algorand), elect a leader responsible for proposing a block. Two effective ways for increasing the performance of a given leader-based blockchain protocol are to increase the size of the proposed blocks or to increase the frequency at which leaders propose blocks. Although these solutions seem simple to implement, they can rapidly reach their limits due to the *store-validate-forwad* block dissemination used by these blockchains. First, the improvements in block size do not necessarily compensate the resulting increase in block dissemination delay, which caps the effective throughput. Second, augmenting the block proposal frequency also has drawbacks because it impacts the way the protocol operates along with its performance. For example in a blockchain such as Bitcoin, a higher block proposal frequency increases the occurrences of forks, thus decreasing the mining efficiency [13] and reaching suboptimal throughput.

Contributions. To address the limitations of these two strategies, we present ALDER, a general construction that multiplexes permissionless blockchain consensus protocols to append not one (large) but several (smaller) blocks to the blockchain. Specifically, ALDER lets multiple nodes propose candidate blocks while allowing the blockchain system to compose and agree on a *macroblock*, an ordered set of blocks.ALDER leverages leader candidates not exploited in the original consensus protocols. Leaders independently propose concurrent blocks containing disjoint sets of transactions, the union of which constitutes a macroblock. To assess the effectiveness of ALDER, we apply its principles to two family-

979-8-3503-9730-7/22/$31.00

representative blockchains: Algorand[3], a scalable proof-of-stake blockchain, and the iconic Bitcoin. Our evaluation, involving up to 10,000 nodes deployed on 100 physical machines, shows that ALDER provides performance gains up to 300%.

Outline. First, we describe ALDER's foundations in Section II and detail its building blocks in Section III. Section IV illustrates the use of ALDER on two representative blockchains. We evaluate the resulting systems in Section V. Finally, we compare ALDER to the state of the art in Section VI before concluding in Section VII.

II. ALDER'S FOUNDATIONS

We assume a permissionless leader-based blockchain system composed of n node processes. The blockchain system enables all nodes to reach consensus on an ordered set of blocks proposed by elected leaders. Consensus is performed in a decentralized, tamper-proof, and publicly-verifiable way. Each newly accepted block extends the blockchain. We assume that nodes implement a blockchain protocol operating under network synchrony or eventually network synchrony assumptions. We model the blockchain protocol as the composition of several steps: the election of leaders, the creation of blocks, the dissemination of blocks throughout the system, and a consensus algorithm that allows all nodes to agree on a block. The blockchain protocol satisfies safety and liveness properties, that have been translated into *common prefix*, *chain quality* and *chain growth* defined in previous work [21], [22], [23]Abstracting from the specifics of consensus protocols employed in leader-based permissionless blockchains, we provide the informal definitions of these properties:

- *Chain-quality*: any (large enough) subset of an honest node's chain contains blocks from honest nodes.
- *Chain-growth*: the chain of any honest node grows at least at the rate of successful rounds.
- *Common prefix*: the chains of all honest nodes must be identical, except for a few tail blocks that are not yet stabilized, *i.e.*, if two honest nodes discard a sufficient number of blocks from their respective chains, they obtain the same prefix.

Our work excludes leaderless consensus [19] and leaderless blockchain models such as Avalanche [20].

A. Store-validate-forward block dissemination bottleneck

Throughput and latency are key metrics to assess the performance of blockchain protocols. They respectively represent the amount of data that a blockchain protocol can append to the chain per unit of time, and the duration before a proposed block is added to the ledger. Two effective ways to improve the performance of blockchains are increasing the frequency at which blocks are proposed, and increasing the block size. While these solutions seem straightforward, they rapidly reach their limits. This is due to the *store-validate-forward* block dissemination mechanism used in these protocols that requires nodes to validate the content of each received block before forwarding it to their neighbors.

Increasing the block size can be counterproductive because nodes should first validate each block before forwarding it. Therefore, larger blocks would be added to the blockchain at a slower rate, which in the optimistic scenario would keep a steady throughput, but with increased latency. There are also drawbacks to increase the frequency of block proposals such as in consensus protocols whose probabilistic termination properties allow the existence of forks, usually with proof-of-work based blockchains (*e.g.*, Bitcoin or Ethereum). As the frequency of block proposals increases, the number of forks appearing in the system increases, resulting in a larger share of generated blocks not appended to the ledger. This lack of efficiency, in this example the mining power efficiency, makes the envisioned performance improvement strategy suboptimal, with a significant proportion of blocks generated quickly and not used to improve blockchain throughput. At this point, the fundamental limits of these two strategies (larger block and faster block generation) are reached, preventing any further performance gains for blockchain consensus protocols.

B. ALDER*: multiplexed blockchain consensus*

The work presented in this paper aims to improve the performance of existing leader-based permissionless blockchain protocols by circumventing the store-validate-forward bottleneck encountered when increasing the block size or the block generation frequency. Our approach, ALDER, consists in multiplexing the execution of blockchain consensus protocols. We call multiplexing a consensus instance the process by which the nodes in the system propose and agree to append a *macroblock*, *i.e.*, a set of blocks totally ordered, to the blockchain instead of a single block per round. The blocks composing the resulting macroblock contain disjoint sets of transactions.

With these new capabilities, the resulting blockchain protocol disseminates a larger number of smaller blocks compared with the original protocol. Then, the consensus protocol decides on a set of these blocks, gathered together into a macroblock by each node. This construction allows circumventing the store-validate-forward bottleneck by fine-tuning the operations of the multiplexed blockchain protocol: either by increasing the number of blocks inside a macroblock (a result of increased block generation frequency), or by increasing the size of each block inside a macroblock, or a mix of both. As a result, the dissemination of these blocks from different nodes in the system optimizes the consumption of network resources and increases the throughput of the blockchain system.

III. ALDER: MULTIPLEXING BLOCKCHAIN CONSENSUS PROTOCOL

In this section, we present the principles of ALDER that transform a leader-based Blockchain Consensus Protocol (*BCP*) into its multiplexed version *BCP++* to increase its performance. Fig. 1 illustrates this transformation on a *BCP* in its canonical form that executes in time-based rounds: leader election, block proposal, block dissemination, and consensus.

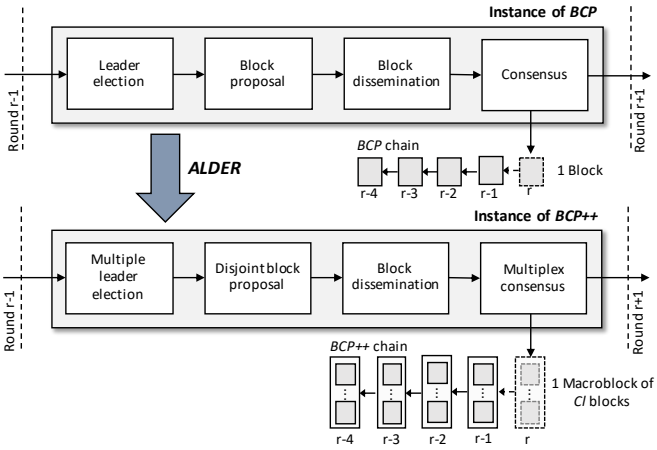
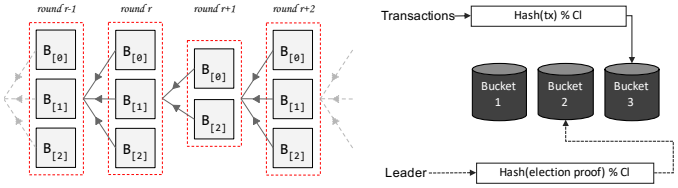


Fig. 1: Multiplexing a blockchain *BCP* using ALDER



(a) Example of a chain of macroblocks with a concurrency level $Cl = 3$. Dashed lines represent macroblocks, and solid squares indicate the composing blocks referencing the previous macroblock hash.

(b) Mapping a transaction to a bucket, and a leader to a bucket with $Cl = 3$. The *election proof* is a blockchain consensus protocol-dependent value.

Fig. 2: ALDER macroblockchain structure and bucket assignment

In each round, *BCP++* elects several leaders, leveraging the election mechanism of the original *BCP* that identifies a node (sometimes several, either by design principle or by side effect) that must propose a candidate block. *BCP++* partitions the transaction hash space into Cl disjoint regions called *buckets*, where Cl is the concurrency level of *BCP++*, which denotes the number of blocks composing the macroblocks. *BCP++* assigns a separate transaction bucket to each leader in a publicly verifiable way. Then, leaders propose blocks with disjoint sets of transactions using the assigned transaction bucket. Finally, *BCP++* runs a multiplexed version of the *BCP* consensus to agree on an ordered composition of the proposed blocks, *i.e.*, a *macroblock*. The nodes in the system wait for this decision before assembling the agreed-upon list of blocks and appending the locally constructed *macroblock* to the chain before proceeding to the next round.

To multiplex *BCP*, ALDER leverages three primitives: (1) transaction space partitioning, (2) multiple leader election and bucket assignment, and (3) multiplexed consensus and macroblocks. We now detail these primitives and how they relate to each other.

A. Transaction hash space partitioning

Appending multiple blocks of transactions in a given round poses the problem of duplicated transactions and duplication attacks. Indeed, in a simple approach, leaders create blocks with the transactions they possess. As a result, some blocks could include transactions already present in other proposed blocks. This redundancy would reduce the throughput gain envisioned in our approach and increase the complexity of the transaction execution phase. Duplication of transactions also opens the doors to duplication attacks by an adversary controlling Byzantine nodes. When Byzantine nodes are elected, the adversary can wait to learn about the blocks proposed by honest leaders and have Byzantine leaders propose blocks containing the same transactions, thus reducing the performance gain of *BCP++*. To address this problem, ALDER creates a transaction hash space partitioned into Cl disjoint regions called *buckets*, deterministically assigns each transactions to one of the buckets, and forces leaders to propose blocks with transactions from a unique bucket. Cl is the concurrency level of *BCP++*, set at its bootstrap. Fig. 2b illustrates the mapping of a received transaction, tx to one of the three available buckets.

Upon receiving a transaction tx, a node first checks its validity before computing its image from a secure cryptographic hash function. To determine the corresponding bucket number that will store the transaction, the modulo Cl operation is used against the hash image of the transaction. Nodes regularly remove transactions from their buckets as transactions appear in appended macroblocks.

B. Multiple leader election and bucket assignment

ALDER extends the leader election of *BCP* to ensure that enough leaders are identified, to avoid macroblocks from being partially filled, and to assign a bucket to each elected leader in an unforgeable and publicly verifiable way. To this end, ALDER leverages the protocol-dependent tamper-resilient *election proof* employed in *BCP* to prove the leader's legitimacy in conducting the task of creating and proposing a block. For example, an election proof in Bitcoin is a solution to the cryptographic puzzle set by the system. Regarding Algorand, the election proof results from a cryptographic sortition procedure evaluating a verifiable random function on publicly known variables.

The bucket assignation in *BCP++* results directly from the computation of the election proof modulo the number of buckets Cl. Fig. 2b illustrates the bucket assignment to a leader, with $Cl = 3$. Each block proposed by a leader must exclusively include transactions whose hashes fall within the bucket assigned to the leader. In this way, any node can validate the correct fabrication of a received block by checking its contents against the bucket number assigned to the block proposer. Using the described construction, leaders of ALDER submit disjoint blocks that can be aggregated to construct one macroblock.

We distinguish two bucket assignment approaches based on whether the election proof is known before or after the

block is created. In the first case, the bucket assignment approach is straightforward as one can directly compute the assignment decision from the existing proof. However, considering blockchain protocols, *e.g.*, PoW-based ones, where nodes have to commit to the content of a block to generate an election proof, the assignment decision cannot be derived from the generated proof without risking a mismatch between the committed block and the resulting assignment. In this configuration, the bucket assignment approach consists in conducting altogether the block creation and bucket assignment by having each node commit to Cl blocks from the Cl disjoint transaction buckets. Once the election proof is known, the node derives the assigned bucket and corresponding block. In section IV, we detail bucket assignment techniques for each approach.

C. Multiplexed consensus and macroblocks

Multiplexing consensus requires the nodes to agree on not one, but a set of proposed blocks *i.e.*, a macroblock. Although multiplexing is highly implementation-dependent, we formalize the consensus results required to preserve the safety and liveness properties of the blockchain model we consider. The consensus decision $d = \{h_i\}_{i\in[0..Cl-1]}$ on a macroblock takes the form of a Cl-long vector of block hash values h_i where i is the bucket number from which each block originates. Each node in the system listens for the blocks proposed by the leaders and locally builds a macroblock based on the consensus decision produced.

A macroblock is a logical composition of up to Cl blocks of size bs bytes. Each block in the macroblock contains the election proof of the leader proposing a block, along with the reference to the previously appended macroblock in the form of a hash image of the previous consensus decision d. Fig. 2a depicts an example of a chain of macroblocks.

Upon receiving a block, each node proceeds through a series of validity and semantic checks regarding the block content and its creation process. Namely, the nodes verify: the leader's election legitimacy carried out by the election proof delivered with the block; the validity of a bucket assignment to the leader; the validity of the transactions themselves; the transactions' membership to the claimed bucket; and the presence of the hash value referencing the previously appended macroblock. If any verification returns a negative result, the node discards the block. If the block passes all checks, it is stored in a global data structure until the end of the round, and is passed on to other nodes in the system according to the block dissemination protocol of *BCP++*. This semantic check is not limited to ALDER and can be extended according to other protocol-specific constraints.

Upon a node receives all blocks for a given macroblock in a given round, it verifies that all blocks refer to the same previously appended macroblock, and then applies a global verification to assess double-spending attempts. Indeed, a malicious party might try to double-spend by appending transactions referencing the same unspent transaction outputs into different blocks within the same macroblock. In such a case, ALDER deterministically solves double-spending attempts by considering as valid the first transaction listed in the macroblock, and discarding the others. More specifically, invalid transactions are left unexecuted.

ALDER does not require that each macroblock contain exactly Cl blocks, as shown in Fig. 2a. Indeed, the blockchain protocols considered for *BCP* include existing protocols that can produce empty blocks, such as Algorand [3].

In consensus protocols with probabilistic termination (such as PoW-based ones), there could be valid blocks generated by legitimate leaders competing for the same bucket number. In this case, ALDER requires an additional protocol-dependent deterministic consensus rule assigning a priority to competing blocks.

D. ALDER Security Analysis

In this section, we give a sketch of the security of ALDER. ALDER leverages three primitives: multiple leader election, transaction hash space partitioning for disjoint block proposal, and multiplex consensus and macroblocks. In the following, we analyze each of these components in light of the vulnerabilities they may introduce.

a) Multiple leader election: ALDER extends the election mechanism of the *BCP* by allowing the election of multiple leaders. In practice, ALDER does not replace the leader election algorithm with another algorithm. Instead, it leverages the existing leader election protocol, which often already elects multiple leader candidates (*e.g.*, in Algorand and Bitcoin). If the protocol does not natively enable multiple leader candidates, ALDER extends the leader election algorithm. In both cases, the resulting election mechanism does elect byzantine nodes in greater proportions than in the base protocol, and does not introduce vulnerabilities into the resulting blockchain beyond those of the underlying election protocol (in the same way as Bitcoin, Bitcoin++ will have forks and these will be fixed similarly).

b) Disjoint block proposal: Having multiple leaders proposing blocks may introduce vulnerabilities. To solve this problem, ALDER relies on a mechanism that allows candidate leaders to propose disjoint blocks. While correct leaders will faithfully follow the protocol, malicious leaders may very well propose blocks without respecting their assigned bucket. In this case, the proposed block will be rejected by the correct nodes, which will eventually check whether the leaders have built their block using the legitimate bucket.

c) Multiplex consensus: Additional verification steps must be performed by the correct nodes in order to validate a macroblock and execute its transactions. Specifically, a correct node that reconstructs a macroblock must perform the following verification. First, it verifies that all blocks composing the macroblock contain the hash of the same previous macroblock. Then, it verifies that all blocks were generated by legitimate leaders. For example, in the case of Bitcoin, the correct node checks the cryptographic puzzle solution. In the case of Algorand, it checks the proof of election resulting from the evaluation of verifiable random

functions with the node's stake in the system. In addition, a correct node verifies that the assignment of the bucket to the leader is valid. Finally, a correct node checks that all transactions within blocks are valid (with respect to UTXO -unspent transaction output- semantics and the buckets they originate from). In addition, each node checks for double-spending transactions in the blocks composing the macroblock. Where applicable, the total order established over the set of transactions resulting from the ordered set of blocks within the macroblock allows nodes to ignore transactions spending the same UTXO more than once. The above verification allows correct nodes to discard blocks submitted by malicious leaders and discard invalid transactions submitted by malicious clients.

d) Safety and liveness sketch: We explain why the macroblockchain resulting from ALDER preserves the *chain-quality*, *chain-growth* and *common-prefix* properties of the original blockchain. Indeed, although ALDER increases the number of elected nodes, the proportion of elected Byzantine nodes remains the same because ALDER does not change the intrinsic behavior of the election mechanism. Similarly, ALDER does not change the underlying way nodes are selected to participate in the consensus protocol, thus the ratio of Byzantine nodes is unchanged. Therefore, ALDER does not allow Byzantine nodes to take further control over the consensus protocol, and thus does not change the properties of the blockchain. For these reasons, the chain-quality, chain-growth and common-prefix properties of the resulting blockchain remains.

IV. Case studies

In this section, we explain the application of ALDER on two different permissionless blockchains, each representative of a different blockchain family: We begin with Algorand [3], one of the most scalable stake- and committee-based blockchain protocols achieving fast transaction confirmation. Then, we cover the case of Bitcoin, the original PoW-based blockchain protocol described by Nakamoto [1]. We provide the full description of Bitcoin++ in Appendix IV-B.

A. Applying ALDER to Algorand

Algorand [3] is among the most scalable PoS-based permissionless blockchain. In the following, we explain its protocol, highlight its bottlenecks, and describe Algorand++.

1) Overview of Algorand: Fig. 3 depicts the main course of an Algorand round. First (❶), Algorand creates several committees responsible for conducting the following steps: block proposal ❸, reduction ❺, and the multiple steps ❻ of the binary agreement of Algorand's Byzantine agreement *BA*⋆. To do so, each node executes a cryptographic sortition that produces an *election proof* and a verifiable *priority* value used to determine its membership in the different committees. This cryptographic sortition elects nodes at random based on their weights (*i.e.*, their currency stake in the system) in a publicly verifiable, and non-interactive way relying on Verifiable Random Functions (VRFs) [24]. The sortition is designed to elect an expected number $\tau_{proposer}$ of block proposers and to assign each selected node a priority, along with its proof. This sortition protects nodes against an adversary aiming at learning the identity of committee nodes and forging targeted attacks. In addition, committees are different for each step of the protocol to prevent targeted attacks on committee members once they send a message.

Once elected as a member of the proposal committee (❷), a node builds a block before sending it along with the priority and election proof value to its neighbors (❸), disseminating these messages via gossiping. To reduce unnecessary communications(❹), and because only one of the proposed blocks will be appended to the chain, each node disseminates blocks based on the priority of the block proposer, ignoring blocks with lower associated priorities. The Byzantine agreement procedure *BA*⋆ from Algorand reduces (❺) the problem of agreeing on one among many block hashes to agreeing on one selected block hash or a default empty block hash. Nodes operate this reduction in two communication steps, and then reach consensus on one of these two values via a binary agreement called *BinaryBA*⋆ (❻) requiring 2 to 11 steps depending on whether the block proposed was honest or not. Nodes wait a certain amount of time to receive priority messages and blocks (respectively 10 seconds and 1 minute, empirically set by the authors [3]). If a node does not receive a block within this delay, it proceeds to the protocol step considering an empty block. Finally (❼), every node counts vote casts during the *BA*⋆ phase to learn about the outcome of the agreement procedure, reaching consensus.

2) Bottlenecks of Algorand: Algorand has transaction confirmation latency of the order of a minute. Despite the impressive performance of Algorand compared with the iconic Bitcoin blockchain (125-fold throughput improvement), it still suffers from performance limitations. In particular, when increasing the size of blocks appended to the chain, Algorand's throughput rapidly saturates. Indeed, the time of gossiping blocks in the network largely dominates the duration required for the protocol to reach consensus which remains constant at 15 seconds on average. This long gossip is a significant limitation to increasing throughput, as shown in our preliminary experiment depicted in Fig. 4.

3) Algorand++: In each round, Algorand++ elects multiple leaders to submit disjoint blocks. Then, Algorand++ decides on a subset of the proposed blocks in the form of a macroblock. Fig. 3 depicts the course of a round.

a) Multiple leader election and bucket assignment: Electing multiple leaders is already part of Algorand. Assigning buckets in an unforgeable and publicly verifiable way is performed by directly applying the Cl modulo operation over the election proof generated by the cryptographic sortition. To avoid buckets from being unassigned, Algorand++ must ensure a sufficiently high number of nodes being elected as block proposers. A too low $\tau_{proposer}$ value could lead to buckets being frequently unassigned, which would not only hinder the envisioned throughput gains but also lead to transactions with specific hashes being ignored for some period. To devise an appropriate value for $\tau_{proposer}$, we

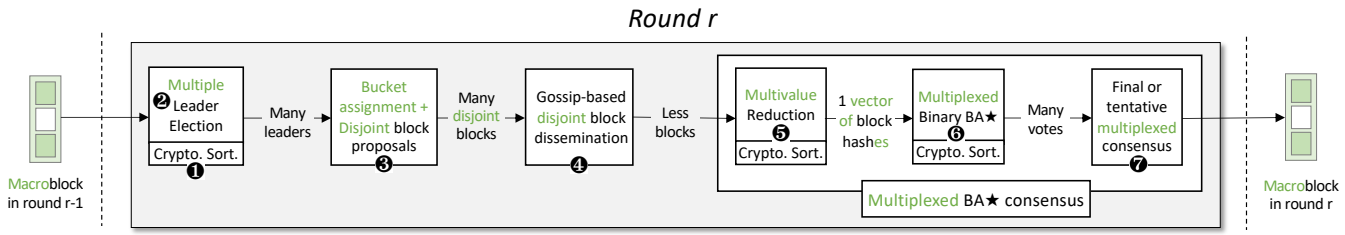
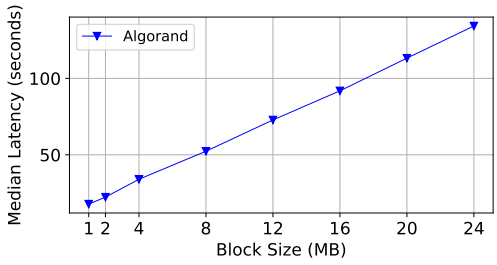
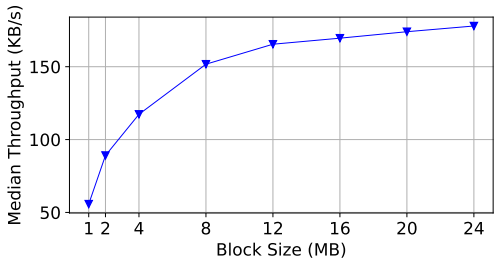


Fig. 3: Round structures of Algorand (black texts and arrows) and Algorand++ (in green)



(a) Latency



(b) Throughput

Fig. 4: Algorand bottleneck

rely on the uniform distribution of the probability $(\frac{1}{Cl})$ of bucket assignment to a leader, directly derived from the properties of hash functions employed in the sortition. In other words, each node has equal chances of being assigned one bucket over another; a bucket is assigned to at least one proposer with probability $1-(1-\frac{1}{Cl})^{\tau_{proposer}}$; and all buckets are assigned to at least one proposer with probability $p=\sum_{i=0}^{Cl}(-1)^{Cl-i}\binom{Cl}{i}(\frac{i}{Cl})^{\tau_{proposer}}$. We set $\tau_{proposer}$ so that $p=0.95$.

b) Disjoint block proposal and dissemination: The block proposal step of Algorand++ is very similar to that of Algorand, except that nodes can be assigned the same transaction bucket as other nodes. Because only one proposed block per bucket number will be appended to the chain, Algorand++ extends the original gossip-based dissemination protocol to reduce unnecessary communications. Similarly to Algorand, each block proposer employs an additional priority value derived by hashing the VRF hash output concatenated with publicly verifiable information of the node's stake in the system and the assigned bucket number. This priority value is then used during block dissemination to discard blocks originating from the same bucket but with lower priority.

c) Multiplexed consensus and macroblocks: The multiplexed *BA*$\star$ agreement protocol takes as input a set of hashes from blocks originating from different buckets. It produces a (possibly not full) vector of block hashes composing the macroblock to be appended at the end of the round. In the multiplexed reduction step, committee members reduce the problem of agreeing on a set of proposed blocks to agreeing either on a *vector* of Cl disjoint block hashes ordered by their bucket number (some of which could be default empty block hashes as in Algorand), or on a vector consisting only of empty block hashes. Then, the multiplexed Byzantine binary agreement is executed to decide one of the two possible solutions. Once reached, nodes gather the blocks Algorand++ has agreed on, proceed through the checks described in section III-C, and build the *macroblock* corresponding to the consensus decision before appending it and continuing to grow the chain.

B. *Applying* ALDER *to Bitcoin*

We first explain the Bitcoin protocol, highlight its bottlenecks, and briefly describe Bitcoin++. The full explanations are provided in Appendix IV-B.

1) Overview of Bitcoin: The global course of a round is depicted in Fig. 6 and includes the following steps: (1) each node builds a 1-MB block of transactions and generates a cryptographic puzzle from the associated block header that contains, among others, the hash of the previously appended block, a Merkle root of the transaction set, and a nonce value; (2) Then each node attempts to solve the cryptographic puzzle and disseminates the block to its neighbors via a gossip protocol when it finds a valid solution. Bitcoin sets the puzzle difficulty to obtain one solution, *i.e.*, one proposed block, every 10 minutes on average. (3) Upon receiving a valid block, the nodes append the latter to their blockchain; (4) If a fork occurs, *i.e.*, more than one valid block exist to extend the chain, the nodes rely on *the longest chain consensus rule* by adopting the fork representing the largest amount of computation.

2) Bottlenecks of Bitcoin: The two main bottlenecks of Bitcoin limiting performance are the small block size and the low frequency at which blocks are added to the chain [13]. Increasing the block size would result in a longer block dissemination time, consequently increasing the occurrences of forks. Similarly augmenting the block generation frequency leads to the same results as it shortens the average duration between the discovery of two valid cryptographic puzzle

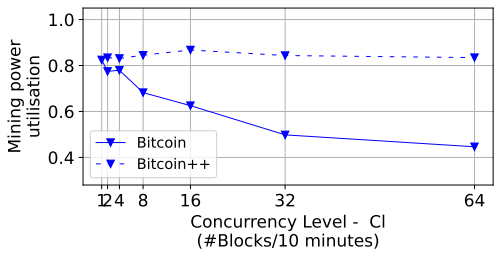


Fig. 5: Ratio of the amount of data appended to the chain over the amount of data generated by the mining nodes

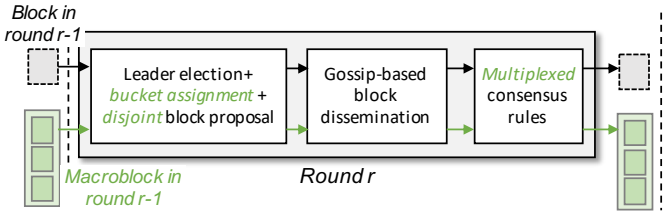


Fig. 6: Round structures of Bitcoin (black texts and arrows) and Bitcoin++ (in green)

solutions. When forks are detected and solved, part of the generated blocks are left orphan, which leaves the system with suboptimal throughput compared to the case where there is no fork. For example, increasing the block generation frequency by a factor of 64 leads to more than half of the generated blocks being discarded (*cf.*, Fig. 5).

3) Bitcoin++ overview: Bitcoin++ allows multiple nodes to independently propose blocks containing disjoint sets of transactions and grow the chain by appending a subset of the proposed blocks in each round. To obtain Cl leaders in the same time interval as in Bitcoin (10 minutes on average), the mining difficulty is divided by Cl.

a) Bucket assignment and disjoint block proposal: For the bucket assignment to be unforgeable and verifiable, ALDER requires that the assignment derives from the election proof of Bitcoin++. In Bitcoin, each node creates a block and then attempts to find an election proof, *i.e.*, a solution to the crypto puzzle. To prevent each node from mining a block that may not match the resulting bucket assignment, Bitcoin++ modifies the cryptographic puzzle by having each node commit and mine a set of Cl disjoint blocks, one of which will be selected once the puzzle is solved. To do so, Bitcoin++ replaces the Merkle tree of the block's transaction set from each block header, *i.e.*, from the puzzle, by the root of the Merkle tree that takes as leaves the Merkle roots of the blocks to which the node is committed. When found, the node gossips its election proof and its block along with the Merkle path necessary to validate the block content.

Because the bucket assignment process may assign one bucket to more than one leader, the time required to obtain Cl disjoint blocks could be greater than 10 minutes, which could inhibit the performance gain envisioned by ALDER. Bitcoin++ addresses this issue by introducing a mechanism reducing the competition for the same bucket numbers. Bitcoin++ adds to each block header a Cl-long array called *sibling*. During the mining process, the miner fills this array with the hashes of the valid blocks it receives from other nodes and for specific bucket numbers in the same round. Suppose the miner solves the puzzle and finds an election proof pointing to a bucket already in use in the sibling array. In that case, the Bitcoin++ node deterministically selects the next available bucket in the sibling array.

b) Multiplexed consensus and macroblocks: Each node waits to collect Cl valid blocks from Cl different buckets, then constructs a macroblock, conducts the series of checks described in section III-C, and appends it to its chain before proceeding to the next round.

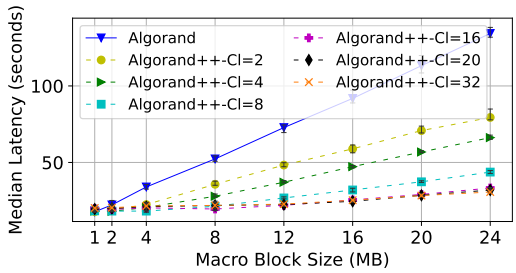
In Bitcoin++, forks occur when two valid blocks compete for the same bucket number in the same macroblock, causing the chain to be extended by two valid macroblocks. To narrow down the fork space probability, Bitcoin++ employs an additional consensus rule with lower priority than the Bitcoin's longest-chain rule: when receiving multiple blocks for the same bucket, nodes prioritize the blocks based on their hash values (the highest wins) to help them decide which one to consider. We distinguish two cases by differentiating when the existence of another candidate block is known. If a node learns that another valid block exists for an already filled bucket in the macroblock for which it is currently mining, then the node selects the block with the highest hash value and updates its sibling array accordingly. If a node learns about another valid block for a macroblock other than the current one, the node selects and mines on top of the chain of macroblocks containing the highest amount of computational power, *i.e.*, applying the longest chain rule of Bitcoin.

V. Evaluation

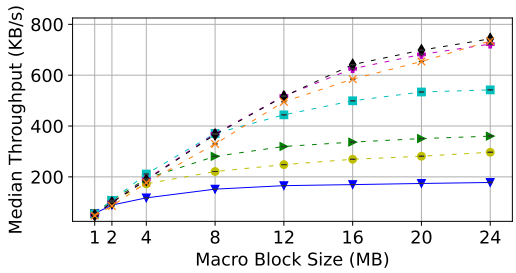
This section evaluates the extent of ALDER's ability to improve the performances of the two blockchains we considered. We first detail our implementation and evaluation environment before presenting the performance of Algorand++ and Bitcoin++.

A. Implementation and evaluation environment

We implemented all baseline protocols and their multiplexed versions using Golang. Experiments were carried out on Grid'5000 [25] with powerful physical machines with 18 cores, 96 GB of memory, and 25Gbps network connectivity. In all experiments, we emulate wide-area network conditions as in major blockchain propositions [3]: we cap the bandwidth for each process to 20 Mbps and add a one-way latency of 50 milliseconds to each communication link. Each node communicates with 8 peers for Bitcoin and Algorand as recommended by the authors. In addition, we rely on a registry service to bootstrap the system: at startup, a node registers itself to this service and receives a list of available peers. Each node has access to a pre-initialized transaction pool to populate block payloads.

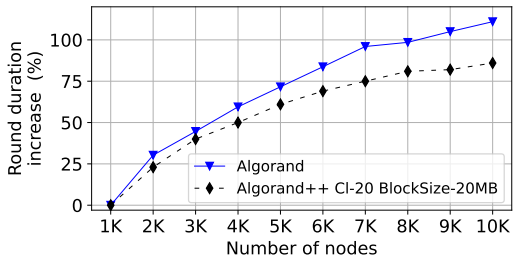


(a) Round latency

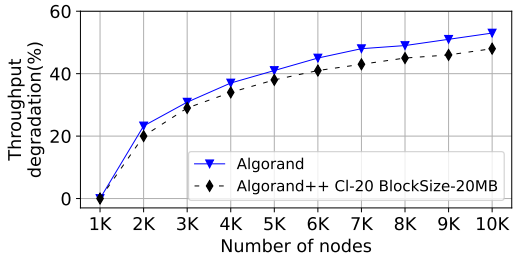


(b) Throughput

Fig. 7: Performance comparison between Algorand++ and Algorand with various concurrency levels and macroblock size



(a) Latency degradation (lower is better)



(b) Throughput degradation (lower is better)

Fig. 8: Throughput and latency degradation at scale

B. Algorand++ Performance Evaluation

We conducted two sets of experiments to evaluate the performance improvements of Algorand++ compared to Algorand. In the first set of experiments, we compare the performance of both protocols using 1,000 nodes on 10 machines. In the second set, we deployed up to 10,000 nodes on 100 machines to compare the scalability characteristics of both protocols.

To evaluate the performance of Algorand and Algorand++, we varied the macroblock size from 1 MB to 24 MB, and the concurrency level of Algorand++ from 1 to 32. Elected leaders in Algorand++ build blocks of size the expected macroblock size divided by Cl., while leaders in Algorand consider block size matching the size of macroblocks appended to the Algorand++ chain. Each experiment lasts 150 rounds.

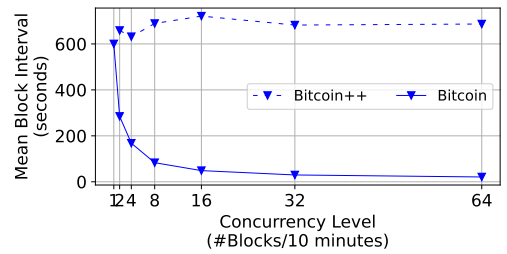
We measure the round latency of the two protocols, indicating the time it takes for a block to be appended, *i.e.*, the duration between the time a block is proposed and the time all nodes observe this block in the chain. The results are depicted in Fig. 7a. The round latency of Algorand increases rapidly as the size of the appended blocks becomes larger, reaching a latency of 134 seconds for blocks of 24 MB. Algorand++'s latencies largely outperform the ones of Algorand for macroblock sizes larger or equal to 2 MB. For instance, for 4 MB blocks, Algorand++'s latencies are respectively 66% and 53% the latencies of Algorand with Cl=2 and Cl=8. The gap between the two blockchains protocols becomes considerably high with larger block sizes. Indeed, for 24 MB blocks, we observe latencies decreased to respectively 56% and 22% the ones of Algorand with $Cl = 2$ and $Cl = 32$.

We also evaluate the effective throughput of the two protocols. Because macroblocks may contain fewer blocks than expected, *i.e.*, Cl, we cannot derive throughput directly from the round latency and the macroblock size. We define effective throughput as the amount of data appended to the blockchain per second. As shown in Fig. 7b, Algorand++ outperforms Algorand in all configurations by reaching up to 743 KB/s with Cl=20 and 24 MB blocks. This configuration is 3.17 times more throughput efficient than Algorand with 24 MB blocks, 5.35 times more efficient than the Algorand configuration that exposes the same round latency (with 4-MB blocks), presents 12 times the throughput of the original Algorand configuration with 1 MB blocks. Our results also highlight the limits Algorand++. Indeed, we reach the maximum throughput with $Cl = 20$, and increasing the concurrency level does not improve performance further.

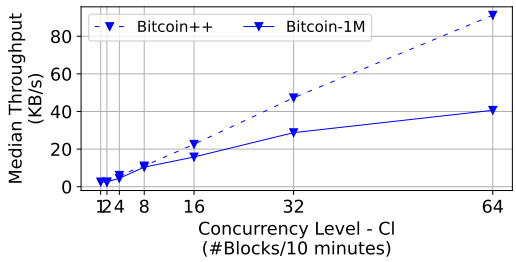
To evaluate how both protocols scale, we vary the number of machines in the testbed from 10 to 100 with 100 nodes per machine, emulating up to 10,000 nodes. We use 1 MB blocks for Algorand and 20 MB blocks and $Cl = 20$ for Algorand++. We measure performance degradation, *i.e.*, throughput decrease and latency increase relative to the performances of each protocol in their baseline configuration with 1,000 nodes. Results in Fig. 8 show that the two protocols present similar trends in performance degradation. Further, ALDER helps to slightly diminish the degradation of performance at large scales thanks to better usage of network resources, as illustrated by the configuration with 10,000 nodes that shows 6% lower throughput degradation on Fig. 8b and 33% lower latency increase on Fig. 8a.

C. Bitcoin++ Performance Evaluation

We conducted experiments with 1000 nodes running on 20 machines to evaluate the performance of Bitcoin++ and



(a) Latency



(b) Throughput

Fig. 9: Round latency and throughput of Bitcoin++ with various block generation frequencies

compare them to the ones of Bitcoin with similar cryptographic puzzle difficulty, *i.e.*, with similar block generation frequencies. To this end, we measure both protocols' mining power utilization, defined as the ratio between the amount of data appended to the blockchain and the amount of data generated by the system in the form of blocks. We also measure each protocol's throughput and block/macroblock latency (block time interval). We vary the concurrency level of Bitcoin++ from 1 to 64, *i.e.*, increasing the block generation frequency from one block every 600 seconds to one block every 9.37 seconds. To do so we reduce the cryptographic puzzle difficulty inversely proportional to the concurrency level. We considered blocks of size 1 MB. In the Bitcoin++ case, this translates into nodes gathering Cl blocks into a macroblock, making macroblocks of size Cl MB. Each experiment runs until the chains of all nodes share a common prefix of 90 macroblocks.

Fig. 9 shows the median throughput and mean block interval time for both protocols. The configuration $Cl = 1$ represents the original Bitcoin protocol providing a throughput of 1.6KB/s with a mean latency of 618 seconds. Bitcoin++ always provides a higher throughput than Bitcoin. With $Cl = 64$, Bitcoin++ provides 91 KB/s, 2.25 times the throughput of Bitcoin with the same block generation frequency, and a x57 improvement to the original Bitcoin protocol. To better understand the origins of such throughput gains, we correlate these results with the mining power utilization ratio depicted in Fig. 5 that remains steady between 0.9 and 0.95 in the Bitcoin++ case when the concurrency level increases, while dropping to 0.44 with Bitcoin. This shows that Bitcoin++ can effectively collect the generated blocks at a high block generation frequency, while Bitcoin can hardly do so. Indeed, in Bitcoin with $Cl = 64$, more than 56% of the generated blocks are not included in the blockchain shared by all nodes at the end of the experiment. This measure also helps understand the extent of fork occurrences in each protocol. The corollary of these results is visible on the mean macroblock block interval that remains at 600-650 seconds for all Bitcoin++ configurations while dropping to approximately 10 seconds for Bitcoin with $Cl = 64$. Indeed, Bitcoin can append blocks quickly, but forks occur very frequently with higher block frequency. Bitcoin++ does not suffer from this problem because macroblocks are added at a slow rate (once per 600-650 seconds), which leaves enough time for any forks to be resolved.

VI. Related Work

The last decade has witnessed the emergence of propositions to improve the performance of some existing permissionless blockchains. Among the proposed approaches, sharding techniques have been applied in OHIE [15] to Bitcoin, composing many parallel instances of Bitcoin's Nakamoto consensus protocol. Similarly, the second version of Ethereum's consensus protocol employs sharding [26]. Decoupling leader election from transaction serialization in the Bitcoin protocol has been proposed in BitcoinNG [13]. BitcoinNG lets each leader appends multiple blocks before another leader is found and takes over with proposing new blocks. Layer-2 solutions, such as rollups [27], have been proposed to address performance improvement without handling the complexity of consensus protocols. Despite all these techniques suffering from the same performance bottleneck due to the store-validate-forward block dissemination mechanism, they are solutions that can be applied to a specific blockchain protocol only. Contrarily, ALDER presents a generic solution that can be applied to many blockchain protocols and solves the store-validate-forward block dissemination bottleneck.

Generic constructions were proposed to improve the performance of BFT consensus and state-machine-replication algorithms that can be used in closed/permissioned blockchains such as Hyperledger Fabric[2]. For example, ISS provides a generic construct for state-machine replication that encapsulates all the consensus logic under a new primitive called *sequenced-broadcast*. ISS implements state-machine-replication by multiplexing multiple instances of sequenced-broadcast which operate concurrently on a partition of the domain of client requests. Similar to ALDER, Mencius [28], BFT-Mencius [11], Mir-BFT [29], OMADA [10], RCC [30], and ISS are solutions that employ multiple leaders and distribute the cost of leadership among them to improve the performance of existing consensus algorithm. All these algorithms assume that the space of consensus sequence number and the partition of the domain of client requests can be evenly and deterministically shared between the multiple leaders. This assumption is valid in the case of closed and deterministic systems where the identities of nodes are known in advance, and where the consensus algorithm produces final results

frequently. However, this assumption does not hold in the case of open blockchain systems that do not rely on deterministic BFT consensus algorithms such as Algorand and Bitcoin. In this kind of system, sequence numbers and transaction domain partitions cannot be distributed between contributing nodes to the consensus because the identities of contributing nodes are not known in advance. ALDER handles this case by relying on dynamic bucket assignment according to election proof obtained from the underlying protocol. To the best of our knowledge, ALDER is the first generic construction of its kind that targets permissionless blockchain consensus protocols to improve them with multiple leaders, unlike previously listed solutions,

VII. Conclusion

We presented ALDER, a set of reusable principles that improve the performance of leader-based permissionless blockchains that cannot reach their full performance due to the *store-validate-forward* block dissemination mechanism they rely on. ALDER alleviates this problem by multiplexing the consensus protocols of these blockchains to append a macroblock, *i.e.*, a set of disjoint blocks, per consensus instance. We applied ALDER's principles to two family-representative permissionless blockchains (Algorand and Bitcoin) and evaluated the performance of the resulting blockchain protocols. Our evaluation, involving up to 10,000 nodes, shows that ALDER provides performance improvements up to 300% in comparison with the baseline protocol.

Acknowledgment

This work was partially funded by Région Nouvelle-Aquitaine, under grant 2018-1R50117 (project B4IOT), and the French Agence Nationale de la Recherche (ANR), under grant ANR-21-CE25-0021 (project GenBlock)

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference (EuroSys 18)*, 2018, pp. 1–15.

[3] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th symposium on operating systems principles (SOSP 17)*, 2017, pp. 51–68.

[4] "Hyperledger – Open Source Blockchain Technologies." [Online]. Available: https://www.hyperledger.org/

[5] R. B. Uriarte and R. DeNicola, "Blockchain-based decentralized cloud/fog solutions: Challenges, opportunities, and standards," *IEEE Communications Standards Magazine*, vol. 2, no. 3, pp. 22–28, 2018.

[6] S. Raval, *Decentralized applications: harnessing Bitcoin's blockchain technology*. " O'Reilly Media, Inc.", 2016.

[7] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Computing Surveys (CSUR 21)*, vol. 54, no. 8, pp. 1–41, 2021.

[8] R. Guerraoui *et al.*, "The consensus number of a cryptocurrency," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC 19)*, 2019, pp. 307–316.

[9] C. Stathakopoulou, M. Pavlovic, and M. Vukolić, "State machine replication scalability made simple," in *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys 22)*, 2022, pp. 17–33.

[10] M. Eischer and T. Distler, "Scalable byzantine fault-tolerant state-machine replication on heterogeneous servers," *Computing*, vol. 101, no. 2, pp. 97–118, 2019.

[11] Z. Milosevic, M. Biely, and A. Schiper, "Bounded delay in byzantine-tolerant state machine replication," in *2013 IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS 13)*. IEEE, 2013, pp. 61–70.

[12] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[13] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, 2016, pp. 45–59.

[14] 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 *USENIX Security Symposium*, 2016, pp. 279–296.

[15] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "Ohie: Blockchain scaling made simple," in *2020 IEEE Symposium on Security and Privacy (S&P 20)*. IEEE, 2020, pp. 90–105.

[16] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EuroCrypt 18)*. Springer, 2018, pp. 66–98.

[17] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, "Solida: A blockchain protocol based on reconfigurable byzantine consensus," *arXiv preprint arXiv:1612.02916*, 2016.

[18] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 18)*, 2018, pp. 931–948.

[19] K. Antoniadis, A. Desjardins, V. Gramoli, R. Guerraoui, and M. I. Zablotchi, "Leaderless consensus," in *IEEE 41st International Conference on Distributed Computing Systems (ICDCS 21)*, 2021.

[20] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and probabilistic leaderless bft consensus through metastability," *arXiv preprint arXiv:1906.08936*, 2019.

[21] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 18)*, 2018, pp. 913–930.

[22] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EuroCrypt 17)*. Springer, 2017, pp. 643–673.

[23] R. Pass and E. Shi, "Fruitchains: A fair blockchain," in *Proceedings of the ACM symposium on principles of distributed computing (PODC 17)*, 2017, pp. 315–324.

[24] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.

[25] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab *et al.*, "Grid'5000: A large scale and highly reconfigurable experimental grid testbed," *The International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[26] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining GHOST and Casper," *arXiv:2003.03052 [cs]*, May 2020.

[27] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain Scaling using Rollups: A Comprehensive Survey," *IEEE Access*, pp. 1–1, 2022.

[28] C.-S. Barcelona, "Mencius: building efficient replicated state machines for wans," in *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 08)*, 2008.

[29] C. Stathakopoulou, T. David, and M. Vukolic, "Mir-bft: High-throughput bft for blockchains," *arXiv preprint arXiv:1906.05552*, 2019.

[30] S. Gupta, J. Hellings, and M. Sadoghi, "Rcc: Resilient concurrent consensus for high-throughput secure transaction processing," in *2021 IEEE 37th International Conference on Data Engineering (ICDE 21)*. IEEE, 2021, pp. 1392–1403.