KUDZU: Fast and Simple High-Throughput BFT

VICTOR SHOUP, Offchain Labs
JAKUB SLIWINSKI, Anza
YANN VONLANTHEN, ETH Zurich

arXiv:2505.08771v1 [cs.DC] 13 May 2025

We present KUDZU, a high-throughput atomic broadcast protocol with an integrated fast path. Our contribution is based on the combination of two lines of work. Firstly, our protocol achieves finality in just two rounds of communication if all but p out of $n = 3f + 2p + 1$ participating replicas behave correctly, where f is the number of Byzantine faults that are tolerated. Due to the seamless integration of the fast path, even in the presence of more than p faults, our protocol maintains state-of-the-art characteristics. Secondly, our protocol utilizes the bandwidth of participating replicas in a balanced way, alleviating the bottleneck at the leader, and thus enabling high throughput. This is achieved by disseminating blocks using erasure codes. Despite combining a novel set of advantages, KUDZU is remarkably simple: intricacies such as "progress certificates", complex view changes, and speculative execution are avoided.

1 INTRODUCTION

Recent years have seen a remarkable surge in popularity and development of resilient distributed systems. The area of blockchain has become a hotbed of research, where systems akin to decentralized world-computers [6, 12, 36] compete to introduce ever-improving protocols. At the heart of any such system is a *atomic broadcast* protocol [13], which allows all replicas in the network to agree on a stream of transactions.

The crucial requirement that these protocols must satisfy is *Byzantine fault tolerance (BFT)* [28], which is the ability of a system composed of n different replicas to continue to function even if some of the replicas fail in arbitrary (potentially adversarial) ways. Moreover, the network connecting the replicas can be unreliable, or even controlled by an attacker. These harsh conditions meant that early global decentralized systems suffered severe disadvantages compared to centralized counterparts, hindering adoption. Despite the challenging setting, research has continued to improve once infamously slow decentralized protocols, and decentralized systems are closing the performance gap.

One key dimension of atomic broadcast performance is the *finalization latency* of new transactions. Historically, the protocols with the best finalization latency are protocols, such as PBFT [15], where a designated leader proposes a block of transactions, and which work in the partially synchronous communication model [19]. In this model liveness is only guaranteed during periods of time where the network is well behaved, but correctness (i.e., safety) is guaranteed unconditionally. For such protocols, it is natural to measure finalization latency as the amount of time that may elapse between when the leader proposes a block and when all other replicas finalize that block. In measuring finalization latency, we assume the leader is honest and the network is well behaved. Such protocols assume $n \geq 3f + 1$, where f is a bound on the number of corrupt replicas, and achieve finalization latency as low as 3δ, where δ is the longest actual message latency between the replicas of the system. However, even 2δ finalization latency is possible in situations where no more than p replicas fail and $n \geq 3f + 2p + 1$ for $p \geq 0$ (or even $n \geq 3f + 2p - 1$ for $p \geq 1$). Protocols achieving 2δ finalization latency in these circumstances employ a special *fast path*. This type of protocol was first explored in [30]. Ideally, such protocols would still maintain a finalization latency of 3δ, even if more than p replicas fail, by running a traditional 3δ slow path alongside the fast path. However, not all fast path protocols enjoy this property (in particular, the protocol in [30] does not). Moreover, fast path protocols typically suffer from added complexity, such as complex view-change logic, "progress certificate" messages, or speculative execution logic. This

added complexity has led to a history of errors (indeed, as pointed out in [1], the protocol in [30] has a liveness bug).

Another crucial dimension of atomic broadcast performance is its *throughput*, that is, the number of transactions that the protocol can process over time given the fixed bandwidth available at every replica. Unfortunatlely, as has been observed and reported in several works [17, 31, 34], leader-based protocols often suffer from a severe "bandwidth bottleneck" at the leader. However, this leader bottleneck can be easily eliminated while still maintaining the leader-based structure and all of its practical advantages [32]. This is done by using erasure codes to ensure that the leader can disseminate large blocks with *low* and *well-balanced* communication complexity.

Another desirable feature of leader-based protocols is lightweight view-change logic that supports frequent leader rotation. In older protocols, such as PBFT, a leader is generally kept in place for an extended period of time, and a complex and somewhat inefficient view-change subprotocol is used to switch to a new leader if the current leader is suspected of being faulty by other replicas. A newer breed of protocols, typified by HotStuff [39], employ extremely lightweight view-change logic that supports frequent leader rotation. Frequently rotating leaders can be beneficial for multiple reasons, for instance, to increase fairness when block production comes with rewards (e.g., maximal extractable value), or to increase censorship resistance.

In this work, our aim is to unite and improve the state-of-the-art in these key dimensions, and to do so while keeping the protocol as simple as possible.

Our contribution. We present Kudzu[1]: a fast and high-throughput atomic broadcast protocol that is remarkably simple compared to its predecessors. Kudzu is the first BFT protocol that combines an optimistic 2δ latency fast path integrated into a 3δ latency slow path, with high-throughput data dispersal, as well as lightweight view change logic that supports frequent leader rotation. We provide a detailed description of the protocol and rigorously prove its security.

(1) **Fast path.** In a network of $n = 3f + 2p + 1$ replicas, Kudzu achieves finalization latency of 2δ if at most p replicas are corrupt.
(2) **Integrated slow path.** If more than p replicas fail, Kudzu maintains the best possible finalization latency of 3δ by running a slow path alongside the fast path.
(3) **High throughput.** High throughput is achieved by Kudzu using erasure codes to ensure that the leader can disseminate large blocks with low and well-balanced communication complexity.
(4) **Lightweight view change.** Kudzu employs an extremely simple and efficient view change logic that allows for frequent leader rotation.

Kudzu is the first atomic broadcast protocol to satisfy all of these properties simultaneously. It also enjoys other properties, such as optimistic responsiveness (the protocol proceeds as fast as the network will allow, with no artificially introduced delays), and a block time (the delay between successive honest leaders proposing a block) of just 2δ.

Technical Intuition. Inspired by DispersedSimplex [32], Kudzu introduces the minimal changes to that protocol required to incorporate a fast path. A designated leader in Kudzu distributes the block data by sending erasure coded fragments to all other replicas. In turn, these replicas then broadcast the fragments themselves, together with a first-round vote on the cryptographic hash identifying the block. The voting logic carefully incorporates a rule that makes sure that a block can already be finalized given $n - p$ first-round votes. As a result, replicas can already reassemble the block and finalize it in 2δ latency in the good case where the network is well behaved, the leader is honest, and at most p replicas are corrupt. Another vote-counting rule is used to finalize the block on a slower, 3δ-latency path, if more than p replicas are corrupt. To avoid getting stuck altogether,

[1]Kudzu is an insidious, fast-growing vine, also known as Mile-a-Minute.

a replica may also vote for a couple of other blocks than the one it initially received from the leader, including a special "timeout" block. The main innovation of our paper is the logic for determining when these "extra" votes are cast — this logic is deceptively simple, but carefully maintains a very delicate balance between liveness and safety. With this innovation, we avoid intricacies such as "progress certificates", complex view changes, and speculative execution as found in other protocols (such as Banyan [35], SBFT [22], Kuznetsov et al. [27], and HotStuff-1 [23]).

2 RELATED WORK

2.1 Fast Path Protocols

A long line of work proposes consensus protocols with a fast path, typically called "fast", "early-stopping" or "one-step" consensus [10, 20, 21, 26, 30, 33]. FaB Paxos [30] introduces a parametrized model with $3f + 2p + 1$ replicas, where $p \geq 0$. The parameter p describes the number of replicas that are not needed for the fast path. These protocols can terminate optimally fast in theory (2δ, or 2 network delays) under optimistic conditions. The paper [30] proves a corresponding lower bound of $3f + 2p + 1$ on the number of replicas needed to achieve this fast behavior. However, as pointed out by Kuznetsov et al. [27] and Abraham et al. [3], this lower bound actually only applies to a restricted type of protocol. In fact, [27] presents a single-shot consensus protocol that uses only $3f + 2p^* - 1$ replicas, with $p^* \geq 1$, and proves a corresponding lower bound.

2.2 Integrated Slow Path Protocols

Interestingly, in practice, these fast path protocols might *increase* the finalization latency, as the fast path requires a round of voting between $n - p$ replicas, which could be slower than two rounds of voting between $n - f - p$ replicas that are concentrated in a geographic area. Banyan [35] performs the fast path in parallel with the 3δ mechanism, which is optimally fast if more than p replicas are faulty or exhibit higher network latency. However, Banyan can exhibit *unbounded* message complexity when there is a corrupt leader. Kudzu addresses this drawback, shares the same optimistic latency properties, improves throughput through balanced dispersal, benefits from a simpler design, and better properties during leader rotation (see section 2.4).

2.3 High-Throughput Protocols

Leader-based protocols such as PBFT [15], HotStuff [39] and Tendermint [11] suffer from a bandwidth bottleneck, as in these protocols the leader is responsible for disseminating transactions to all replicas. As mentioned above, this bottleneck has been well known for quite a while [17, 31, 34]. One way to alleviate this bottleneck is to move away from leader-based protocols to a more symmetric, leaderless protocol design where all replicas disseminate transactions. Such an approach was already taken in [17, 24, 31, 34]. These protocols typically have worse latency than leader-based protocols, and moreover, since many replicas may end up broadcasting the same transactions, the supposed improvement in throughput can end up being illusory (note that [34] actually tackles this duplication problem head on). Protocols such as HotStuff [39] and SBFT [22] aim to eliminate the all-to-all broadcasts that occur in the voting logic of many other atomic broadcast protocols, but in so doing increase the finalization latency of the protocol while doing nothing to eliminate the leader bottleneck.

A different approach to address leader bottlenecks makes use of erasure coding [36]. By splitting blocks into smaller, erasure-coded shares, the leader can transmit less data, leading to a balanced utilization of resources. This line of work shines by providing high throughput and low latency [32, 38].

2.4 View Changes

Some atomic broadcast protocols are notorious for having complex view changes, especially in the presence of a fast path [5]. In the case of Zzyzzva [25] and UpRight [16], safety errors were later pointed out [1]. Improved protocols such as Thelma, Velma and Zelma [2] revisit and correct these.

3 MODEL AND PRELIMINARIES

We consider a network of n replicas $P_1, \ldots, P_n$ called replicas. Up to f replicas can be Byzantine, i.e., deviate from the protocol in arbitrary ways, such as collude to attack the protocol. The remaining replicas follow the protocol and are referred to as honest. We aim to provide better latency if only up to p replicas do not cooperate. In other words, $n - p$ honest replicas including the leader are enough for the fast path to be effective.

To prove the security of Kudzu we assume

$$n \geq 3f + 2p + 1, \tag{1}$$

where $f \geq 1$ and $p \geq 0$. Moreover, to prove concrete bounds on message, communication, and storage complexity, we assume

$$n < 3(f + p + 1). \tag{2}$$

Assumption (2) is not a real restriction. Indeed, if $n \geq 3(f + p + 1) = 3f + 3p + 3$, then we can always *increase* p appropriately, leaving n and f alone, so that both (1) and (2) are satisfied. This only increases the overall performance of the protocol.

3.1 Network Assumptions

We will not generally assume network synchrony. However, we say the network is *δ-synchronous at time T* if every message sent from an honest replica P at or before time T to an honest replica Q is received by Q before time $T + \delta$. We also say the the network is **δ-synchronous over an interval** $[a, b + \delta]$ if it is δ-synchronous at time T for all $T \in [a, b]$.

While our protocols always guarantee *safety*, even in periods of asynchrony, *liveness* will only be guaranteed in periods of δ-synchrony, for appropriately bounded δ (and this synchrony bound may be explicitly used by the protocol). Thus, we are essentially working in the *partial synchrony* model of [19]. However, instead of assuming a single point in time (GST) after which the network is assumed to be synchronous, we take a somewhat more general point of view that models a network that may alternate between periods of asychrony and synchrony.

Replicas have local clocks that can measure the passage of local time. We do not assume that clocks are synchronized in any way. However, we do assume that there is no *clock skew*, that is, all clocks tick at the same rate (but we could also just assume the skew is bounded and incorporate that bound into the protocol's synchrony bound).

3.2 Problem Statement

The purpose of state machine replication is to totally order blocks containing transactions, so that all replicas output transactions in the same order. Our protocol orders blocks by associating them with natural numbered slots. Some leader replica is assigned to every slot. For every slot, either some block produced by the leader might be finalized, or the protocol can yield an empty block. The guarantees of our protocol can be stated as follows:

- **Safety**: If some honest replica finalizes block B in slot v, and another honest replica finalizes block B' in slot v, then $B = B'$.
- **Liveness**: If the network is in a period of synchrony, each honest replica continues to finalize blocks for slots $v = 1, 2, \ldots$

In addition to liveness and safety, we support a fast path:

- **Fast Termination**: If the network is in a period of synchrony, $n - p$ replicas behave momentarily honestly, and an honest leader proposes a block B at time t, then every honest replica finalizes B at time $t + 2\delta$.

3.3 Cryptographic Assumptions

3.3.1 Signatures and certificates. We make standard cryptographic assumptions of secure digital signatures and collision-resistant hash functions. We assume all replicas know the public keys of other replicas.

We use a k-out-of-n threshold signature scheme. We refer to a *signature share* and a *signature certificate*: signature shares from k replicas on a given message may be combined to form a signature certificate on that message. This can be implemented in various ways, e.g., based on BLS signatures [7–9]. The security property for such a threshold signature scheme may be stated as follows.

> **Quorum Size Property:** It is infeasible to produce a signature certificate on a message m, unless $k - f'$ honest replicas have issued signature shares on m, where $f' \leq f$ is the number of corrupt replicas.

For ease of exposition and analysis, we assume *static corruptions*, so the adversary must choose some number $f' \leq f$ replicas to corrupt at the very beginning of the protocol execution, and then does not corrupt any replicas thereafter. That said, we believe all of our the protocols are secure against *adaptive corrupts*, provided the threshold signature scheme is as well.

As we will see, we will need a one k-out-of-n threshold signature scheme with $k = n - f - p$, and another with $k = n - p$.

3.3.2 Information Dispersal. We make use of well-known techniques for *asynchronous verifiable information dispersal (AVID)* involving erasure codes and Merkle trees[14].

Erasure codes. For integer parameters $k \geq d \geq 1$, a (k, d)*-erasure code* encodes a bit string M as a vector of k *fragments*, $f_1, \ldots, f_k$, in such a way that any d such fragments may be used to efficiently reconstruct M. Note that for variable-length M, the reconstruction algorithm also takes as input the length β of M. The reconstruction algorithm may fail (for example, a formatting error)—if it fails it returns $\perp$, while if it succeeds it returns a message that when re-encoded will yield k fragments that agree with the original subset of d fragments. We assume that all fragments have the same size, which is determined as a function of k, d, and β.

Using a Reed-Solomon code, which is based on polynomial interpolation, we can realize a (k, d)-erasure code so that if $|M| = \beta$, then each fragment has size $\approx \beta/d$.

In our protocol, the payload of a block will be encoded using an $(n, f + p + 1)$-erasure code. Such an erasure code encodes a payload M as a vector of fragments $f_1, \ldots, f_n$, any $f + p + 1$ of which can be used to reconstruct M. This leads to a data expansion rate of at most roughly 3; that is, $\sum_i |f_i| \approx n/(f + p + 1) \cdot |M| < 3|M|$, where last inequality follows from assumption (2).

Merkle trees. Recall that a Merkle tree allows one replica P to commit to a vector of values $(v_1, \ldots, v_k)$ using a collision-resistant hash function by building a (full) binary tree whose leaves are the hashes of $v_1, \ldots, v_k$, and where each internal node of the tree is the hash of its two children. The root r of the tree is the commitment. Replica P may "open" the commitment at a position $i \in [k]$ by revealing v_i along with a "validation path" π_i, which consists of the siblings of all nodes along the path in the tree from the hash of v_i to the root r. We call π_i a *validation path from the root under r to the value v_i at position i*. Such a validation path is checked by recomputing the nodes along the corresponding path in the tree, and verifying that the recomputed root is equal to the

given commitment r. The collision resistance of the hash function ensures that P cannot open the commitment to two different values at a given position.

Encoding and decoding. For a given payload M of length β, we will encode M as a vector of fragments $(f_1, \ldots, f_n)$ using an $(n, f + p + 1)$-erasure code, and then form a Merkle tree with root r whose leaves are the hashes of $f_1, \ldots, f_n$. We define the *tag* $\tau := (\beta, r)$.

For a tag $\tau = (\beta, r)$, we shall call (f_i, π_i) a *certified fragment for* τ *at position* i if

- f_i has the correct length of a fragment for a message of length β, and
- π_i is a correct validation path from the root under r to the fragment f_i at position i.

The function *Encode* takes as input a payload M. It builds a Merkle tree for M as above with root r (encoding M as a vector of fragments, and then building the Merkle tree whose leaves are the hashes of all of these fragments). It returns

$$\big(\ \tau,\ \{(f_i, \pi_i)\}_{i \in [n]}\ \big),$$

where τ is the tag (β, r), β is the length of M, and each (f_i, π_i) is a certified fragment for τ at position i.

The function *Decode* takes as input

$$\big(\ \tau,\ \{(f_i, \pi_i)\}_{i \in \mathcal{I}}\ \big),$$

where $\tau = (\beta, r)$ is a tag, $\mathcal{I}$ is a subset of $[n]$ of size $f + p + 1$, and each (f_i, π_i) is a certified fragment for τ at position i. It first reconstructs a message M' from the fragments $\{f_i\}_{i \in \mathcal{I}}$, using the size parameter β. If $M' = \perp$, it returns $\perp$. Otherwise, it encodes M' as a vector of fragments $(f'_1, \ldots, f'_n)$ and Merkle tree with root r' from $(f'_1, \ldots, f'_n)$. If $r' \neq r$, it returns $\perp$. Otherwise, it returns M'.

Under collision resistance for the hash function used for the Merkle trees, any $f + p + 1$ certified fragments for given tag τ will decode to the same payload — moreover, if τ is the output of the encoding function, these fragments will decode to M (and therefore, if the decoding function outputs $\perp$, we can be sure that τ was maliciously constructed). This observation is the basis for the protocols in [18, 29, 37]. Moreover, with this approach, we do not need to use anything like an "erasure code proof system" (as in [4]), which would add significant computational complexity (and in particular, the erasure coding would have to be done using parameters compatible with the proof system, which would likely lead to much less efficient encoding and decoding algorithms).

4 KUDZU PROTOCOL

Kudzu iterates through slots, where in each slot there is a designated leader who proposes a new block, which is chained to a parent block. Leaders may be rotated in each slot, either in a round-robin fashion or using some pseudo-random sequence. The slot leader disseminates large blocks in a way that keeps the overall communication complexity low and avoids a bandwidth bottleneck at the leader. The communication is balanced, meaning that each replica, including the leader, transmits roughly the same amount of data over the network.

We describe our protocol as a few simple subprotocols that run concurrently with each other:

- Vote and Certificate Pool: data structure managing the votes and certificates;
- Complete Block Tree: data structure storing the reconstructed blocks;
- Main Loop: loop issuing votes that makes sure some blocks become *notarized* and *finalized*. Notarized blocks can be reconstructed by all replicas and are added to the Complete Block Tree. Finalized blocks are ordered and output by the protocol.

4.1 Protocol Data Objects

Definition 4.1 (block). A block B is of the form $\mathrm{Block}(v, \tau, h_p)$, where

- $v \in \{1, 2, \ldots\}$ is the slot number associated with the block (and we say *B is a block for slot v*),
- τ is the tag obtained by encoding B's payload M,
- h_p is the hash of B's parent block (or $h_p = \perp$ by convention if B's parent is a notional "genesis" block).

We also call a certified fragment for the tag τ a *certified fragment for B*.

The block $B_v^{\text{timeout}} = \mathsf{Block}(v, \perp, \perp)$ is a special *timeout block*.

Votes and certificates.

Definition 4.2 (votes and certificates). A *notarization vote from P_i for block B* is an object of the form $\mathsf{NotarVote}(B, \sigma_i, f_i, \pi_i)$, where σ_i is a valid signature share from P_i on the object $\mathsf{Notar}(B)$, and (f_i, π_i) is either a certified fragment for B at position i, or $(f_i, \pi_i) = (\perp, \perp)$ if $B = B_v^{\text{timeout}}$. A *notarization certificate for B* is an object of the form $\mathsf{NotarCert}(B, \sigma)$, where σ is a valid $(n - f - p)$-out-of-n signature certificate on the object $\mathsf{Notar}(B)$.

The notarization vote on the timeout block is also called the timeout vote, and the notarization certificate for the timeout block is called the timeout certificate.

A *first vote from P_i on block B* is an object of the form $\mathsf{FirstVote}(\sigma_i', \mathsf{NotarVote}(B, \sigma_i, f_i, \pi_i))$, where σ_i' is a valid signature share from P_i on the object $\mathsf{First}(B)$, and $\mathsf{NotarVote}(B, \sigma_i, f_i, \pi_i)$ is a notarization vote from P_i on block B. A *fast finalization certificate for $B \neq B_v^{timeout}$* is an object of the form $\mathsf{FirstCert}(B, \sigma)$, where σ is a valid $(n - p)$-out-of-n signature certificate on the object $\mathsf{First}(B)$.

A *finalization vote from P_i on block B* is an object of the form $\mathsf{FinalVote}(B, \sigma_i)$, where σ_i is a valid signature share from P_i on the object $\mathsf{Final}(B)$. A *finalization certificate for B* is an object of the form $\mathsf{FinalCert}(v, \sigma)$, where σ is a valid $(n - f - p)$-out-of-n signature certificate on the object $\mathsf{Final}(B)$.

4.2 Vote and Certificate Pool.

Each replica maintains a *pool* with votes and certificates. For every slot, the pool stores votes and certificates associated with the slot.

As we will see, by design, for any one slot, a honest replica can send only 1 first vote, 1 timeout vote, and 1 finalization vote. As we will also see later (in Section 5.3), for one slot a honest replica can only send at most 3 (non-timeout) notarization votes. Any votes exceeding this bound can only result from misbehavior and are not added to the pool. For example, if a replica receives more than three notarization votes for a given slot from some replica P, the replica can ignore these votes and conclude that P is corrupt. In particular, only one first vote per replica can be observed by the protocol loop in Protocol 1. When a first vote is added to the pool, also the contained notarization vote is added to the pool.

Whenever a replica receives enough votes, and it does not already have a corresponding certificate, it will generate the certificate, add it to the pool, and broadcast the certificate to all replicas. Similarly, whenever a replica receives a certificate, and it does not already have a corresponding certificate, it will add it to the pool, and broadcast the certificate to all replicas. For one slot it is impossible that the pool would receive or create more than: 1 timeout certificate, 1 fast finalization certificate, 1 finalization certificate, and 5 notarization certificates. The first bound is immediate, since there can be only one timeout certificate per round. The second and third bounds follow from the safety analysis below. The fourth bound follows from the analysis in Section 5.3.

4.3 Complete Block Tree.

Each replica also maintains a *complete block tree*, which is a tree of blocks rooted at a notional genesis block at slot 0. We will show that the number of blocks for a given slot is bounded by 5. A block $B = \mathsf{Block}(v, \tau, h_p)$ is added to the tree if each of the following holds:

- the certificate pool contains a notarization certificate for B and $B \neq B_v^{\text{timeout}}$;
- $h_p = \bot$ or the complete block tree contains a parent block with the hash h_p;
- the replica has received enough (i.e. $f + p + 1$) notarization votes to reconstruct the effective payload M of B as
$$M \leftarrow \mathit{Decode}(\tau, \{(f_i, \pi_i)\}_{i \in I}),$$
where $\{(f_i, \pi_i)\}_{i \in I}$ is the corresponding collection of certified fragments for τ;
- $M \neq \bot$ and satisfies some correctness predicate that may depend on the path of blocks (and their payloads) from genesis to block B.

A replica does not broadcast anything in addition to adding a block to the block tree.

4.4 Finalization

We say that a block B for slot v is *explicitly finalized by replica P* if the complete block tree of P contains B and the certificate pool of P contains either a fast finalization certificate for B or finalization certificate for B. In this case, we say that all of the predecessors of block B in the complete block tree are *implicitly finalized by P*. The payloads of finalized blocks may be then transmitted in order to the execution layer of the protocol stack of a replicated state machine.

4.5 Generating Block Proposals.

The logic for generating block proposal material B, $(f_1, \pi_1), \ldots, (f_n, \pi_n)$ in slot v in line 16 of Protocol 1 is as follows:

- build a payload M that validly extends the path in the complete block tree ending at a block B_{p} with hash h_p;
- compute
$$(\tau, \{(f_i, \pi_i)\}_{i \in [n]}) \leftarrow \mathit{Encode}(M);$$
- set $B \coloneqq \mathsf{Block}(v, \tau, h_p)$.

4.6 Validating Block Proposals.

To check if $\mathsf{BlockProp}(B, f_j, \pi_j)$ is a valid block proposal from the leader in slot v in line 18 of Protocol 1, replica P_j checks that the following conditions holds:

- B is of the form $\mathsf{Block}(v, \tau, h_p)$,
- (f_j, π_j) is a certified fragment for τ at position j.

Additionally, we require the following conditions to also be checked:

- the complete block tree contains a block with the hash h_p in a slot $v' < v$;
- the pool contains timeout certificates for slots $v' + 1, \ldots, v - 1$;

These last two conditions might not hold at a given point in time, but may hold at a later point in time, and so might need to be checked again when blocks or certificates are added.

4.7 Main Loop

The main protocol for P_j is described in Protocol 1. In the description, $\mathsf{leader}(v)$ denotes the leader for slot v — as mentioned, leaders may be rotated in each slot, either in a round-robin fashion or using some pseudo-random sequence.

As mentioned in Section 4.2, each replica only considers only one first vote that it receives from any other replica. To make this explicit in the protocol, we use a map *firstVote* from replicas to blocks to record these votes. The protocol also uses simple helper functions on this map.

- allVotes(*firstVote*): the total number of first votes for slot v contained in the pool,

- maxVotes(*firstVote*): the maximal number of first votes on some non-timeout block B for slot v contained in the pool,
- manyVotes(*firstVote*): returns the set of non-timeout blocks in slot v on which the pool contains at least $f + p + 1$ first votes.

For example, if the pool contains 1, 2, 3, 4 first votes on blocks B_1, B_2, B_3, B_v^{timeout} respectively (and $f + p + 1 = 2$), then allVotes$(v) = 10$, maxVotes$(v) = 3$, and manyVotes$(v) = \{B_2, B_3\}$.

The protocol also uses a subprotocol ReconstructAndNotarize(v, B), defined in Protocol 2.

Each replica P_j moves through slots $v = 1, 2, \ldots$. In each slot, it will enter a loop in which it waits for one of several conditions to trigger an action. These conditions are based on the objects in its pool and its complete block tree, as well as local variables.

- Lines 9–11 present the logic for the replica successfully exiting the slot by finding a block B for that slot in its complete block tree. In addition, if the replica did not broadcast a notarization vote for any other block (including the timeout block) for that slot, it will also broadcast a finalization block for B.
- Lines 12–13 present the logic for the replica unsuccessfully exiting the slot by obtaining a timeout certificate for that slot.
- Lines 14–17 present the logic for the replica proposing a block for that slot if it is the leader for that slot. It generates the block proposal as in Section 4.5, extending the path in the complete block tree ending at B_{p}. Here, B_{p} is either the genesis block or the block that it found in its complete block tree the last time it successfully exited a slot (other choices of B_{p} are possible).
- Lines 18–21 present the logic for the replica broadcasting a first vote for a non-timeout block B. It will do so only if B is a valid block proposed by the leader (as in Section 4.6) and has not already first voted. Recall that a first vote for B also includes a notarization vote for B.
- Lines 22–25 present the logic for the replica broadcasting a first vote for the non-timeout block for this slot. It will do so only if a sufficient amount of time has passed since it entered the slot and has not already first voted.
- Lines 26–27 present the logic for updating the map *firstVotes*. No other actions are taken.
- Lines 28–31 present the logic for the replica taking a "second look" at a block B, if it has received sufficiently many first votes for B. It will do so only if it has already first voted (and has not already taken a second look at B). If it can reconstruct a valid payload for B, it will broadcast a notarization vote for B (if it has not already done so); otherwise, it will broadcast a notarization vote for the timeout block.
- Lines 32–35 present the logic for the replica broadcasting a timeout vote under special circumstances. It will do so only if it has already first voted and

  $$\text{allVotes}(\textit{firstVote}) - \text{maxVotes}(\textit{firstVote}) \geq f + p + 1.$$

  We note that the quantity

  $$\text{allVotes}(\textit{firstVote}) - \text{maxVotes}(\textit{firstVote})$$

  cannot decrease as we add entries to *firstVote*. That is because, when we add an entry, the first term increases by 1 and the second either decreases by 1 or remains unchanged.

5 PROTOCOL ANALYSIS

We start by proving some helpful properties.

Protocol 1 Kudzu main loop for replica P_j

```
1:  B_p ← genesis                                              ▷ parent of the next block
2:  for v = 1, 2, ... do
3:      T_start ← clock()                                      ▷ slot-local initialisation
4:      done, proposed, firstVoted ← false
5:      notarized ← {}                                         ▷ blocks already notarized
6:      secondLook ← {}                                        ▷ blocks already reconsidered
7:      firstVote ← {}                                         ▷ map P_i ↦ B for their first vote

8:      while ¬done wait until either
9:          there exists a block B for slot v in the complete block tree ⇒
10:             B_p ← B; done ← true
11:             if notarized ⊆ {B} then broadcast FinalVote(B, σ_j)

12:         the pool contains a timeout certificate for v ⇒
13:             done ← true

14:         ¬proposed ∧ leader(v) = P_j ⇒
15:             proposed ← true
16:             generate block proposal material B, (f_1, π_1), ..., (f_n, π_n) extending block B_p
17:             for all i ∈ [n]: send BlockProp(B, f_i, π_i) to P_i

18:         ¬firstVoted ∧ received valid BlockProp(B, f_j, π_j) from leader(v) ⇒
19:             firstVoted ← true
20:             broadcast FirstVote(σ'_j, NotarVote(B, σ_j, f_j, π_j))
21:             notarized ← notarized ∪ {B}

22:         ¬firstVoted ∧ clock() > T_start + Δ_timeout ⇒
23:             firstVoted ← true
24:             broadcast FirstVote(σ'_j, NotarVote(B_v^timeout, σ_j, ⊥, ⊥))
25:             notarized ← notarized ∪ {B_v^timeout}

26:         received valid FirstVote(_, NotarVote(B, _)) from P_i and firstVote[P_i] = ⊥ ⇒
27:             firstVote[P_i] ← B

28:         firstVoted ∧ ∃B ∈ manyVotes(firstVote) \ secondLook
29:          and B's parent is in the complete block tree ⇒
30:             secondLook ← secondLook ∪ {B}
31:             ReconstructAndNotarize(v, B)

32:         firstVoted ∧ (allVotes(firstVote) − maxVotes(firstVote) ≥ f + p + 1)
33:          and B_v^timeout ∉ notarized ⇒
34:             broadcast NotarVote(B_v^timeout, σ_j, ⊥, ⊥)
35:             notarized ← notarized ∪ {B_v^timeout}
```

Protocol 2 ReconstructAndNotarize(v, B)

```
1: reconstruct payload for B
2: if reconstruction succeeds ∧ payload valid then
3:     if B ∉ notarized then
4:         broadcast NotarVote(B, σ_j, f_j, π_j)
5:         notarized ← notarized ∪ {B}
6: else
7:     if B_v^timeout ∉ notarized then
8:         broadcast NotarVote(B_v^timeout, σ_j, ⊥, ⊥)
9:         notarized ← notarized ∪ {B_v^timeout}
```

Lemma 5.1 (Validity Property). *Suppose that a block B for some slot v is added to the complete block tree of some replica. If the leader for slot v is honest, B must have been proposed by that leader.*

Proof. By the Quorum Size Property (see Section 3.3.1) for notarization certificates, at least $n - 2f - p$ honest replicas must have broadcast notarization shares for B. Since we are assuming $n \geq 3f + 2p + 1$, it follows that $n - 2f - p > 0$, so some honest replica P must have broadcast a notarization share for B. This could happen either at line 20 or at line line 31. In the first case, B must be the block that P received as a proposal from the leader. In the second case, since P received $f + p + 1$ first votes for B, one of these must be a first vote for B from some honest replica Q, and so B must be the block that Q received as a proposal from the leader. □

Lemma 5.2 (Completeness Property for Certificates). *If a certificate X appears in the vote and certificate pool (so X is a notarization, finalization, or timeout certificate) then X (or its equivalent) will eventually appear in the corresponding pool of every other replica. Moreover, if X appears in a replica's pool at a time T at which the network is δ-synchronous, it will appear in every replica's pool before time $T + \delta$.*

Proof. This is clear, since a certificate appearing in the vote and certificate pool is broadcast immediately. □

Lemma 5.3 (Completeness Property for Blocks). *If a block B appears in the complete block tree, then B will eventually appear in the corresponding tree of every other replica. Moreover, if B appears in a replica's tree at a time T at which the network is δ-synchronous, it will appear in every replica's tree before time $T + \delta$.*

Proof. We are relying on the Quorum Size Property (see Section 3.3.1) for notarization certificates: when a notarization certificate for a block B is added to the certificate pool, at least $n - 2f - p$ honest replicas must have already broadcast notarization votes for B, which contain B as well as fragments sufficient to reconstruct B's payload, since $n - 2f - p \geq f + p + 1$. □

5.1 Safety

Lemma 5.4 (Fast Finalization Implication). *Suppose a block B is fast finalized by some honest replica, then the number of honest replicas that first vote for anything other than B is at most p.*

Proof. Let $f' \leq f$ be the actual number of corrupt replicas. If some honest replica fast finalizes B, then — by the Quorum Size Property (see Section 3.3.1) for fast finalization certificates — at least $n - p - f'$ honest replicas first voted for B. So the number of honest replicas that first vote for anything other than B is at most $(n - f') - (n - p - f') = p$. □

Lemma 5.5 (Uniqueness of Fast Finalization Property). *If an honest replica receives $f + p + 1$ first votes for a block B in round v, then no block different from B can be fast finalized in round v by any honest replica.*

Proof. Suppose towards contradiction that and some replica P receives $f + p + 1$ first votes for a block B but a block $C \neq B$ is fast finalized by some honest replica Q. By the previous lemma, at most p honest replicas could first vote for anything other than C. Therefore, P can receive at most $f + p$ first votes for B, a contradiction. □

Lemma 5.6 (Absence of Fast Finalization Property). *If the inequality $allVotes(firstVote) - maxVotes(firstVote) \geq f + p + 1$ holds for a replica in round v, then no block can be fast finalized in round v by any replica.*

Proof. Suppose towards contradiction that a replica P fast finalized block B while for replica Q the inequality holds. By Lemma 5.4, at most p honest replicas fast vote for anything other than B. Therefore, Q can receive at most $f + p$ fast votes for anything other than B. Let $count_B$ denote the number of first votes for B received by Q. It follows that, at any point in time, for replica Q, we have

$$\mathsf{allVotes}(\mathit{firstVote}) - \mathsf{maxVotes}(\mathit{firstVote}) \leq \mathsf{allVotes}(\mathit{firstVote}) - \mathit{count}_B \leq f + p,$$

a contradiction. □

Lemma 5.7 (Incompatibility of Notarization and (Fast) Finalization Property). *Suppose that a valid block B for some slot v is (fast) finalized by some replica. If any replica obtains a notarization certificate for a block C in slot v, then $C = B$. (In particular, no other block for slot v can be added to the complete block tree of any replica and no timeout certificate can be obtained for slot v.)*

Proof. We first prove the Incompatibility of Notarization and *Fast* Finalization. Suppose towards contradiction that for slot v a fast finalization certificate exists for block B and a notarization certificate exists for block C, with $C \neq B$. By the Quorum Size Property (see Section 3.3.1) for notarization certificates, this implies that an honest replica broadcast a notarization vote for C.

- On the one hand, suppose that the notarization vote for C was sent by the protocol on line 31. Due to the condition on line 28, this means that $f + p + 1$ first votes were received for a block D. Note that D is a non-timeout block. Moreover, by the logic of ReconstructAndNotarize, either C is a timeout block or $D = C$. By the Uniqueness of Fast Finalization Property (Lemma 5.5) we know that $D = B$. Since B is assumed to be valid, C cannot be a timeout block, so we also have $B = D = C$, a contradiction.
- On the other hand, suppose that the notarization vote for C was sent on line 34. By the Absence of Fast Finalization Property (Lemma 5.6), this implies that no block is fast finalized, again a contradiction.

The Incompatibility of Notarization and Finalization Property follows from a standard quorum intersection argument, based on the fact that in each slot an honest replica issues a finalization vote only for a block only if it did not send a notarization vote for a different block in that slot (see line 11). Suppose towards contradiction that for slot v a finalization certificate exists for block B and a notarization certificate exists for block C, with $C \neq B$. By the Quorum Size property (see Section 3.3.1) for finalization and notarization certificates, if $f' \leq f$ is the number of corrupt replicas, then at least $n - f - p - f'$ honest replicas broadcast finalization votes for B, and a disjoint set of at least the same number of honest replicas broadcast notarization votes for C. This implies that there are at least $2(n - f - p - f')$ distinct honest replicas. However, under the assumption that $n \geq 3f + 2p + 1$, we have $2(n - f - p - f') \geq n - f' + 1$, a contradiction. □

We can now easily state and prove our main safety lemma:

Lemma 5.8 (Safety). *Suppose a replica P explicitly finalizes a block B for slot v, and a block C for slot $w \geq v$ is in the complete block tree of some replica Q. Then B is an ancestor of C in Q's complete block tree.*

Proof. By the Incompatibility of Notarization and (Fast) Finalization Property (Lemma 5.7), no timeout certificate for slot v can be produced. Let C' be the parent of C and suppose w' is the slot number of C'. Since C' is in Q's complete block tree, a notarization certificate for C' must have been produced, which means at least one honest replica must have issued a notarization vote for C', which means $v \leq w' < w$. The inequality $v \leq w'$ follows from the fact that there is no timeout certificate for slot v, and an honest replica will issue a notarization share for C only if it has timeout certificates for slots $w' + 1, \cdots, w - 1$. If $v = w'$, we are done by the Incompatibility of Notarization and (Fast) Finalization Property (Lemma 5.7), and if $v < w'$, we can repeat the argument inductively with C' in place of C. □

5.2 Liveness

Liveness follows from the following lemmas. The first lemma analyzes the optimistic case where the network is synchronous and the leader of a given slot is honest, showing that the leader's block will be committed.

Lemma 5.9 (Liveness I). *Consider a slot $v \geq 1$ and suppose the leader for slot v is an honest replica Q. Suppose that the first honest replica P to enter the loop iteration for slot v does so at time T_0. Suppose that the network is δ-synchronous over the interval $[T_0, T_0 + 4\delta]$ for some δ with $\Delta_{\text{timeout}} \geq 2\delta$. Then, Q will propose a block for slot v by time $T \leq T_0 + \delta$. Each honest replica will finish the loop iteration before time $T + 2\delta$ by adding Q's proposed block B to its complete block tree. Moreover, if $n - p$ replicas are honest, each honest replica will finalize B by time $T + 2\delta$. If more than p replicas are corrupt, each honest replica will finalize B by time $T + 3\delta$.*

Proof. By the Completeness Properties (Lemma 5.2 and Lemma 5.3), before time $T_0 + \delta$, each honest replica will enter slot v by time $T_0 + \delta$, having either a timeout certificate for slot $v - 1$ or a block for slot $v - 1$ in its complete block tree. Before time $T \leq T_0 + \delta$, the leader Q will propose a block B that extends a block B' with slot number $v' < v$. By the logic of the protocol, we know that Q must have timeout certificates for slots $v' + 1, \ldots, v - 1$ at the time it makes its proposal, as well as a notarization certificate for B'. Again by the Completeness Properties, before time $T + \delta$, each honest replica will have B' in its complete block tree and all of these timeout certificates in its certificate pool. Each honest replica will receive B before this time, and because $\Delta_{\text{timeout}} \geq 2\delta$, will broadcast a first vote for B by this time. Because all honest replicas broadcast a first vote for B, each such replica will only see ever see at most f first votes for any other block. It follows that each honest replica will only ever see

- manyVotes(*firstVote*) $\subseteq \{B\}$, and
- allVotes(*firstVote*) − maxVotes(*firstVote*) $\leq$ allVotes(*firstVote*) − $count_B \leq f$, where $count_B$ is the number of of first votes for B that it sees.

Therefore, honest replicas will not broadcast a notarization votes in slot v for anything other than B. Before time $T + 2\delta$, each honest replica will have added B to its complete block tree and broadcast a finalization vote on B. If $n - p$ replicas are honest, each honest replica will have added a fast finalization certificate to its pool by time $T + 2\delta$ as well. Otherwise, if more than p replicas are corrupt, each honest replica will finalize B before time $T + 3\delta$, when adding the finalization certificate for slot v to its pool. □

The second lemma analyzes the pessimistic case, when the network is asynchronous or the leader of a given round is corrupt. It says that eventually, all honest replicas will move on to the next round.

Lemma 5.10 (Liveness II). *Suppose that the network is δ-synchronous over an interval $[T, T + \Delta_{\text{timeout}} + 3\delta]$, for an arbitrary value of δ, and that at time T, some honest replica is in the loop iteration for slot v and all other honest replicas are in a loop iteration for v or a previous slot. Then, before time $T + \Delta_{\text{timeout}} + 3\delta$, all honest replicas exit slot v.*

Proof. By the Completeness Properties (Lemma 5.2 and Lemma 5.3), every honest replica will enter the slot v before time $T + \delta$. By time $T + \delta + \Delta_{\text{timeout}}$, every honest replica will broadcast a first vote either for a block proposal, or for B_v^{timeout}.

Consider two cases:

(a) At least $f + p + 1$ honest replicas broadcast a first vote for the same non-timeout block B.
(b) No set of $f + p + 1$ honest replicas broadcast a first vote for the same non-timeout block B.

Case (a). Since $f + p + 1$ replicas cast notarization votes for B, some honest replica did so. Since this replica had to have B's parent in its complete block tree, by Completeness Properties (Lemma 5.3) each honest replica will have B's parent in its complete block tree by time $T + \Delta_{\text{timeout}} + 2\delta$. All honest replicas observe all first votes from other honest replicas before time $T + \Delta_{\text{timeout}} + 2\delta$, and so each honest replica will call ReconstructAndNotarize(v, B) before that time, unless it has already exited slot v. If some honest replica has exited slot v before that time, the by Completeness Properties (Lemma 5.2 and Lemma 5.3), all honest replicas will exit the slot before time $T + \Delta_{\text{timeout}} + 3\delta$. Otherwise, assume no honest replica has exited before time $T + \Delta_{\text{timeout}} + 2\delta$. Whenever ReconstructAndNotarize(v, B) is called by some honest replica, it has received at least $f + p + 1$ first votes for B, and so can attempt to reconstruct B. If it fails to reconstruct B's payload or finds that it is invalid, then it and all honest replicas will do so and issue a timeout vote (this follows from collision resistance of the hash function and the fragment decoding logic). Otherwise, each honest replica will issue a notarization vote for B. Therefore, before time $T + \Delta_{\text{timeout}} + 3\delta$, each honest replica will either add B to its complete block tree or the timeout certificate to its pool, and proceed exit the slot.

(b). Consider some honest replica P. By time $T + \Delta_{\text{timeout}} + 2\delta$, P will have observed all votes of other honest replicas. If the only entries in *firstVote* are those from honest replicas, then the inequality allVotes(*firstVote*) − maxVotes(*firstVote*) $\geq f + p + 1$ must hold. To see this, if $f' \leq f$ is the number of corrupt replicas, then

$$\begin{aligned} \text{allVotes}(\textit{firstVote}) - \text{maxVotes}(\textit{firstVote}) &\geq (n - f') - (f + p) \\ &\geq n - 2f - p \\ &\geq f + p + 1 \quad (\text{since } n \geq 3f + 2p + 1). \end{aligned}$$

As we have already observed, the quantity allVotes(*firstVote*) − maxVotes(*firstVote*) cannot decrease as we add entries to *firstVote*. Therefore if P has not cast a timeout vote in slot v yet, it will do so by time $T + \Delta_{\text{timeout}} + 2\delta$, unless it has already exited slot v by that time. In either case, all honest replicas will exit slot v by time $T + \Delta_{\text{timeout}} + 3\delta$. □

5.3 Boundedness

We prove some simple results that allow us to bound message and storage complexity. Here, we are assuming both (1) and (2).

Let us consider notarization votes on non-timeout locks. An honest replica sends a first vote for at most one such block. Any notarization vote for some other non-timeout block B requires that

the replica found $B \in \text{manyVotes}(\textit{firstVotes})$. In other words, the replica has seen at least $f + p + 1$ other replica's first votes for B. At most one first vote per replica is considered when computing manyVotes(*firstVotes*). Therefore, an honest replica can cast at most $\lfloor n/(f + p + 1) \rfloor$ notarization votes beyond the first vote, and by (2), $\lfloor n/(f + p + 1) \rfloor \leq 2$, for a total of 3 notarization votes for non-timeout blocks.

Suppose there are $f' \leq f$ corrupt replicas, and so $n - f'$ honest replicas. The honest replicas therefore issue at most $3(n - f')$ notarization votes for non-timeout blocks per slot.

Now, to construct a notarization certificate for a non-timeout block B, we require that $(n - f - p - f')$ honest replicas cast a notarization vote for B. Therefore, by the result in the previous paragraph, there can be at most

$$N \coloneqq \lfloor (3(n - f'))/(n - f - p - f') \rfloor \tag{3}$$

distinct blocks for which a notarization certificate can be constructed.

We claim that $N \leq 5$. To see this, first note that the derivative of $(3(n - f'))/(n - f - p - f')$ with respect to f' is positive, and therefore the right-hand side of (3) is maximized when $f = f'$, and so

$$N \leq \lfloor (3(n - f))/(n - 2f - p) \rfloor.$$

So it suffices to show that $(3(n - f))/(n - 2f - p) < 6$. This is easily seen to follow by a simple calculation using (1).

So to summarize, we have shown that in any slot,

(1) each replica casts a notarization vote for at most 2 non-timeout blocks besides its first vote, and
(2) there are at most 5 distinct blocks for which a notarization certificate can be constructed.

This immediately gives us bounds on the message and storage complexity if the protocol per slot.

5.4 Complexity

Based on the concrete bounds in Section 5.3 (and the preliminary discussion in Section 4.2), it is easily seen that the message complexity per slot is $O(n^2)$. Based on the properties of erasure codes and Merkle trees discussed in Section 3.3.2, the communication complexity per slot is $O(\beta n + n^2 \log(n)\kappa + n^2\lambda)$, where β is a bound on the payload size, κ is the output length of the collision-resistant hash, and λ is a bound on the length of any signature share or certificate. Moreover, the communication is *balanced*, in that every replica, *including the leader*, transmits the same amount of data, up to a constant facrtor. It is also easily seen that each replica needs to store $O(\beta + n \log(n)\kappa + n\lambda)$ bits of data.

6 PROTOCOL VARIATIONS

Our protocol can be adapted to the setting of $n \geq 3f + 2p^* - 1$, where $p^* \geq 1$ [3], with the insight that if honest replicas vote for different blocks in the same slot, the leader has to be corrupt. The liveness analysis can leverage the fact that, in this case, honest replicas will observe at least $n - f$ votes from non-leader replicas, as is done in [35]. We plan to analyze a variation of Kudzu with this adaptation in an extended version of this paper.

A variation of DispersedSimplex [32] features segments of consecutive slots with the same leader. Such stable leader assignment is beneficial for the throughput of the protocol. The same technique can be applied to our protocol, and we plan to analyze a variation of Kudzu featuring stable leaders in an extended version of this paper.

REFERENCES

[1] Abraham, I., Gueta, G., Malkhi, D., Alvisi, L., Kotla, R., Martin, J.P.: Revisiting fast practical byzantine fault tolerance. arXiv preprint arXiv:1712.01367 (2017)

[2] Abraham, I., Gueta, G., Malkhi, D., Martin, J.P.: Revisiting fast practical byzantine fault tolerance: Thelma, velma, and zelma. arXiv preprint arXiv:1801.10022 (2018)

[3] Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Good-case latency of byzantine broadcast: A complete categorization. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. pp. 331–341 (2021)

[4] Alhaddad, N., Duan, S., Varia, M., Zhang, H.: Succinct erasure coding proof systems. Cryptology ePrint Archive, Paper 2021/1500 (2021), https://eprint.iacr.org/2021/1500

[5] Aublin, P.L., Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 bft protocols. ACM Transactions on Computer Systems (TOCS) **32**(4), 1–45 (2015)

[6] Blackshear, S., Chursin, A., Danezis, G., Kichidis, A., Kokoris-Kogias, L., Li, X., Logan, M., Menon, A., Nowacki, T., Sonnino, A., et al.: Sui lutris: A blockchain combining broadcast and consensus. Tech. rep., Technical Report. Mysten Labs. https://sonnino. com/papers/sui-lutris. pdf (2023)

[7] Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y. (ed.) Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2567, pp. 31–46. Springer (2003). https://doi.org/10.1007/3-540-36288-6_3, https://doi.org/10.1007/3-540-36288-6_3

[8] Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. Cryptology ePrint Archive, Paper 2018/483 (2018), https://eprint.iacr.org/2018/483

[9] Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2248, pp. 514–532. Springer (2001). https://doi.org/10.1007/3-540-45682-1_30, https://doi.org/10.1007/3-540-45682-1_30

[10] Brasileiro, F., Greve, F., Mostéfaoui, A., Raynal, M.: Consensus in one communication step. In: Parallel Computing Technologies: 6th International Conference, PaCT 2001 Novosibirsk, Russia, September 3–7, 2001 Proceedings 6. pp. 42–50. Springer (2001)

[11] Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus (2018), arXiv:1807.04938, http://arxiv.org/abs/1807.04938

[12] Buterin, V.: Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)

[13] Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Annual International Cryptology Conference. pp. 524–541. Springer (2001)

[14] Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: Fraigniaud, P. (ed.) Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3724, pp. 503–504. Springer (2005). https://doi.org/10.1007/11561927_42, https://doi.org/10.1007/11561927_42

[15] Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Seltzer, M.I., Leach, P.J. (eds.) Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999. pp. 173–186. USENIX Association (1999), https://dl.acm.org/citation.cfm?id=296824

[16] Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., Riche, T.: Upright cluster services. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. pp. 277–290 (2009)

[17] Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In: Bromberg, Y., Kermarrec, A., Kozyrakis, C. (eds.) EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022. pp. 34–50. ACM (2022). https://doi.org/10.1145/3492321.3519594, https://doi.org/10.1145/3492321.3519594, also at arXiv:2105.11827, http://arxiv.org/abs/2105.11827

[18] Dolev, S., Wang, Z.: SodsBC: Stream of distributed secrets for quantum-safe blockchain. In: 2020 IEEE International Conference on Blockchain (Blockchain). pp. 247–256. IEEE Computer Society, Los Alamitos, CA, USA (2020)

[19] Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988). https://doi.org/10.1145/42282.42283, http://doi.acm.org/10.1145/42282.42283

[20] Friedman, R., Mostefaoui, A., Raynal, M.: Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. IEEE Transactions on Dependable and Secure Computing **2**(1), 46–56 (2005)

[21] Guerraoui, R., Vukolić, M.: Refined quorum systems. In: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing. pp. 119–128 (2007)

[22] Gueta, G.G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M., Seredinschi, D.A., Tamir, O., Tomescu, A.: Sbft: A scalable and decentralized trust infrastructure. In: 2019 49th Annual IEEE/IFIP international conference on

dependable systems and networks (DSN). pp. 568–580. IEEE (2019)
[23] Kang, D., Gupta, S., Malkhi, D., Sadoghi, M.: Hotstuff-1: Linear consensus with one-phase speculation. arXiv preprint arXiv:2408.04728 (2024)
[24] Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is DAG. In: Miller, A., Censor-Hillel, K., Korhonen, J.H. (eds.) PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021. pp. 165–175. ACM (2021). https://doi.org/10.1145/3465084.3467905, https://doi.org/10.1145/3465084.3467905, also at arXiv:2102.08325, http://arxiv.org/abs/2102.08325
[25] Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative byzantine fault tolerance. In: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles. pp. 45–58 (2007)
[26] Kursawe, K.: Optimistic byzantine agreement. In: 21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings. pp. 262–267. IEEE (2002)
[27] Kuznetsov, P., Tonkikh, A., Zhang, Y.X.: Revisiting optimal resilience of fast byzantine consensus. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. p. 343–353. PODC'21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3465084.3467924, https://doi.org/10.1145/3465084.3467924
[28] Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems **4**(3), 382–401 (1982)
[29] Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-MVBA: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: Emek, Y., Cachin, C. (eds.) PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020. pp. 129–138. ACM (2020). https://doi.org/10.1145/3382734.3405707, https://doi.org/10.1145/3382734.3405707
[30] Martin, J.P., Alvisi, L.: Fast byzantine consensus. IEEE Transactions on Dependable and Secure Computing **3**(3), 202–215 (2006)
[31] Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 31–42. ACM (2016). https://doi.org/10.1145/2976749.2978399, also at https://eprint.iacr.org/2016/199
[32] Shoup, V.: Sing a Song of Simplex. In: Alistarh, D. (ed.) 38th International Symposium on Distributed Computing (DISC 2024). Leibniz International Proceedings in Informatics (LIPIcs), vol. 319, pp. 37:1–37:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2024). https://doi.org/10.4230/LIPIcs.DISC.2024.37, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.DISC.2024.37
[33] Song, Y.J., van Renesse, R.: Bosco: One-step byzantine asynchronous consensus. In: International Symposium on Distributed Computing. pp. 438–450. Springer (2008)
[34] Stathakopoulou, C., David, T., Pavlovic, M., Vukolić, M.: Mir-BFT: High-throughput robust bft for decentralized networks (2019), arXiv:1906.05552, http://arxiv.org/abs/1906.05552
[35] Vonlanthen, Y., Sliwinski, J., Albarello, M., Wattenhofer, R.: Banyan: Fast rotating leader bft. In: Proceedings of the 25th International Middleware Conference. pp. 494–507 (2024)
[36] Yakovenko, A.: Solana: A new architecture for a high performance blockchain v0.8.13. https://solana.com/solana-whitepaper.pdf (2018)
[37] Yang, L., Park, S.J., Alizadeh, M., Kannan, S., Tse, D.: DispersedLedger: High-throughput byzantine consensus on variable bandwidth networks (2021), arXiv:2110.04371, http://arxiv.org/abs/2110.04371
[38] Yang, L., Park, S.J., Alizadeh, M., Kannan, S., Tse, D.: {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). pp. 493–512 (2022)
[39] Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT consensus in the lens of blockchain (2018), arXiv:1803.05069, http://arxiv.org/abs/1803.05069