Red Belly: A Secure, Fair and Scalable Open Blockchain

Conference Paper · May 2021

DOI: 10.1109/SP40001.2021.00087

Tyler Crain
IRISA - Institut de Recherche en Informatique et Systèmes Aléatoires
24 PUBLICATIONS **288** CITATIONS
SEE PROFILE

Christopher Natoli
The University of Sydney
10 PUBLICATIONS **274** CITATIONS
SEE PROFILE

Vincent Gramoli
University of Sydney and EPFL
159 PUBLICATIONS **2,628** CITATIONS
SEE PROFILE

Red Belly: A Secure, Fair and Scalable Open Blockchain

Tyler Crain
University of Sydney
Australia

Christopher Natoli
University of Sydney
Australia

Vincent Gramoli
University of Sydney and CSIRO
Australia

Abstract: Blockchain has found applications to track ownership of digital assets. Yet, several blockchains were shown vulnerable to network attacks. It is thus crucial for companies to adopt secure blockchains before moving them to production. In this paper, we present *Red Belly Blockchain (RBBC)*, the first secure blockchain whose throughput scales to hundreds of geodistributed consensus participants. To this end, we drastically revisited Byzantine Fault Tolerant (BFT) blockchains through three contributions: *(i)* defining the *Set Byzantine Consensus* problem of agreeing on a *superblock* of all proposed blocks instead of a single block; *(ii)* adopting a fair leaderless design to offer *censorship-resistance* guaranteeing the commit of correctly requested transactions; *(iii)* introducing *sharded verification* to limit the number of signature verifications without hampering security. We evaluate RBBC on up to 1000 VMs of 3 different types, spread across 4 continents, and under attacks. Although its performance is affected by attacks, RBBC scales in that its throughput increases to hundreds of consensus nodes and achieves 30k TPS throughput and 3 second latency on 1000 VMs, hence improving by 3× both the latency and the throughput of its closest competitor.

I. Introduction

Unlike classic replicated state machines (RSM), blockchains [73] aim at offering a peer-to-peer model where many geodistributed participants replicate the system state and where even more requesters can check their balance and issue cryptographically signed transactions. While permissionless blockchains allow any nodes to participate in the consensus protocol and permissioned blockchains allow only a pre-determined set of nodes to participate, new blockchain designs will likely be *open* permissioned where permissioned nodes offer a Byzantine Fault Tolerant (BFT) consensus service to which permissionless clients have access [15]: Ethereum v2.0 gives permissions in exchange of a proof-of-stake while other blockchains are naturally building upon BFT consensus [50], [37], [63]. The limitation of these blockchains is that they cannot offer high throughput when deployed on hundreds of nodes: verifying all transactions is computationally intensive while agreeing on a block is communication intensive.

In this paper, we propose *Red Belly*[1] *Blockchain (RBBC)*, the first secure blockchain that scales to hundreds of geodistributed consensus nodes. As far as we know, previous blockchains either assume synchrony (a known bound on message delays) or their performance drops when the number of nodes increases. By contrast, RBBC achieves a strong form of *scalability* where throughput does not drop as the number of consensus nodes increases. Scaling to hundreds of consensus nodes is ideal for a decentralized representative governance where at least one consensus node can run in each of the 195 independent sovereign nations around the world to serve the requests of many more nodes. RBBC is secure in that it prevents double spending [73] by resolving conflicts and not forking—even with asynchrony—and is resilience optimal in that, among the n nodes executing each of its consecutive consensus instances, up to $t < n/3$ can be Byzantine [60]. The consensus protocol of RBBC is also time optimal [35] and was proved correct for any number of nodes using model checking [9]. As RBBC supports reconfiguration [87], the set of consensus nodes can be changed before being bribed.

To achieve scalability, RBBC offers a new balancing method that totally orders all transactions while assigning them to distinct groups of *proposer* and *verifier* nodes. *(i)* Its leaderless design balances the communication load on multiple proposers, hence avoiding the congestion and slowdown induced by the least responsive node. As opposed to classic Byzantine consensus protocols that rely on a leader to propose transactions, RBBC's multiple proposers combine distinct sets of transactions into a *superblock* to solve the new *Set Byzantine Consensus* problem and commit more transactions per consensus instance. *(ii)* Its verification sharding balances the computation load across verifiers. As opposed to existing blockchains where all n nodes typically verify every transaction, each of our transaction signatures is verified by between $t+1$ and $2t+1$ *verifiers*.

We conducted the most extensive experiment of a secure blockchain on a thousand virtual machines (VMs) spread over more than 10 countries in 4 continents, under normal conditions and under adversarial attacks. We implemented RBBC over a period of 4 years in 30k lines of code and compared it to the traditional leader-based PBFT [18] with well-known optimizations [10], [50] and the HoneyBadgerBFT [68], and observed that, only RBBC scales to hundreds of geodistributed VMs be they high-end (18 hyperthreaded cores) or low-end VMs (4 vCPUs). The absence of a leader without the need for

[1] "Red belly" stems from the name of the red-bellied black snake endemic to the Sydney region where this blockchain was designed and implemented.

System	deployment	network	throughput			latency	#nodes	#machines
			peak	max scale	under attack			
Elastico [66]	country-wide	emulated	20 KB/s	20 KB/s	N/A	800 sec	1,600	800
Algorand [37]	country-wide	emulated	208 KB/s	90 KB/s	$\sim$	22 sec	50,000	1000
Omniledger [55]	datacenter	emulated	205 MB/s	1.8 MB/s	N/A	14 sec	1,800	60
RapidChain [93]	datacenter	emulated	4.2 MB/s	3.8 MB/s	N/A	9 sec	4,000	32
Mir [83]	world-wide	real	140 MB/s	30 MB/s	0	5 sec	100	100
RBBC	world-wide	real	264 MB/s	12.3 MB/s	760 KB/s	3 sec	8,560	1000

TABLE I

Scalable blockchain experiments – the peak throughput of Elastico is obtained at 100 nodes with 14 1MB-sized blocks in 700 seconds [66] or at larger scale by producing 16 of them within 800 seconds [93]. Omniledger achieves 3500 TPS when tolerating 25% of adversarial power for 512-byte transactions but goes up to $4 \cdot 10^5$ TPS when the adversarial power is 1%. RapidChain peaks at 7384 TPS for 512-byte transactions but needs smaller blocks to achieve a 9-second latency, which leads to 7000 TPS [93]. The throughput of Algorand is 750 MB/h=208 KB/s or 327 MB/h=90 KB/s for a 22-second latency and the impact of attacks on its performance seems negligible [37]. The throughput of Mir is from 0, when a leader stalls after another, to 60,000 TPS, when all $n = 100$ proposers are correct, with 500-byte payload and no durability, and peaks at 40,000 TPS when $n = 4$ with 3500-byte payload [83]. The throughput of RBBC peaks at 660,000 TPS for 400-byte transactions with $n = 300$ (Fig.3) but varies from 1900 TPS under a 33% coalition attack (Fig.10) to 30,684 TPS at max scale, where its latency is 3 s (Table IV).

a common coin yields a 3-fold improvement over the latency and throughput of its closest competitor, HoneyBadgerBFT.

RBBC also guarantees a level of fairness. RBBC ensures *censorship-resistance* in that all correctly requested transactions are eventually committed, hence implying blockchain liveness [19] for correctly signed transactions, but offering extra guarantees to requesters. A first consequence of censorship-resistance is to mitigate a series of problems that plague other blockchains, like anomalies [74], unfairness [46], front-running [82], [27] or oligarchy [94], by ordering transactions with their age. A second consequence of this property is particularly appealing at the application level when transactions are well-formed (i.e., non-conflicting and valid): RBBC exchanges $O(n^3)$ bits to commit a single transaction, just like lightweight leader-based implementations [48], but without their increased latency.

We start by presenting the background (§II) and our goals and assumptions (§III). Then we present an overview of RBBC (§IV), its design and implementation (§V) and the reason why RBBC is a secure, fair and scalable blockchain (§VI). We evaluate RBBC world-wide on up to 1000 machines and under attacks, and compare it against other blockchains (§VII). We conclude (§VIII) and provide the full proofs (§A) and a disclosure (§B).

II. Background and Threat Model

Most previous works on Byzantine fault tolerant blockchains expose themselves to a series of threats summarized below.

A. Double spending

The motivation for solving the traditional consensus problem is to guarantee that, all replicas agree on a unique block at a given index [77]. The uniqueness of the block avoids forks that could otherwise allow an attacker to double spend its coins in two branches [80]. This problem is symptomatic of blockchains that offer probabilistic guarantees [73], [91], [66], [37], [55], [93]: although it helps with scalability as depicted in Table I, the probability that consensus fails grows with the number of blocks that need to be agreed upon [93]. A blockchain based on deterministic consensus ensures that consensus is reached among correct nodes if less than a third of all nodes are Byzantine. The traditional definition, however, unnecessarily limits the scalability of the blockchain [89], [14], [81]: most blockchains decide *at most one* of the proposed blocks [14], [81]. Indeed, the leader whose proposal is eventually decided needs to propose requests to many nodes, its network interface thus acting as a bottleneck [49], [41]. By contrast, RBBC solves the Set Byzantine Consensus problem by deciding up to $\Omega(n)$ proposals (Theorem 2) when $n > 3t$ and no forks are possible even when the communication is asynchronous.

B. Incorrectly signed transactions

To decide a superblock that combines all proposed blocks, one may think of solving a variant of the consensus problem to instead combine proposals, into a decision [8], [76], [22]. For example, Agreement on a Core Set (ACS) [8], Interactive Consistency (IC) [60] and Vector Consensus (VC) [76] all require at least $t + 1$ (either $n - t$ or $t + 1$ with $n > 3t$) proposed values to be decided. In blockchain, however, there may not even be $t + 1$ compatible proposed blocks: when among $2t + 1$ blocks one transaction per block is not correctly signed or transactions of distinct blocks *conflict* in that they are concurrent and withdraw assets from an account that has insufficient balance. This is why, we introduce the *Set Byzantine Consensus* problem (§III) that ensures that all correct nodes extract the correctly signed and non-conflicting transactions from the same set of proposals. This allows RBBC to combine proposed valid blocks into a *superblock* (§V-C) for its performance to increase with the system size.

C. Unfairness

The reason for discarding invalid transactions is to cope with Byzantine requests that may lead to anomalies [74], unfairness [46], front-running [82], [27] or oligarchy [94]: a correct requester could be unable to commit any of its transactions due to invalid transactions always being committed first. An influential consortium blockchains called Hyperledger Fabric [5] suffers from this censorship: as acknowledged by its authors a spamming attack could lead its orderer service to order only invalid transactions. Its optimized version, called FastFabric [39], mitigates this issue by being $\sim 7\times$ faster, however, none of these versions tolerate malicious failures (including denial-of-service attacks). Even its Byzantine-fault tolerant ordering service [81] suffers from this issue as it cannot detect whether a transaction is valid. RBBC favors older requested transactions to achieve censorship-resistance (Theorem 3).

D. Network attacks

Many blockchains assume synchrony where all messages must be delivered in less than a known bounded time [73], [91], [66], [52], [2], [54], [37], [44], [93], [61]. The drawback is the vulnerability to various attacks [45], [74], [75], [32], [33]. Elastico [66], RapidChain [93], OmniLedger [55] achieve scalability by sharding the consensus. This sharding and block-DAGs (e.g., [62]) are not to confuse with our *verification* sharding as RBBC orders all transactions at once. One obtains the world state of RBBC by accessing the last block, instead of having to traverse the tip of a DAG. A sharded blockchain [90] scales even further than what is presented in Table I, however, some of its transactions are not atomic as their withdrawal is decoupled from their credit. Other blockchains [53], [2], [55], [79] assume synchrony but use a partially synchronous consensus algorithm [31]. Ouroboros reached 247 TPS [51] and assumes synchrony [52] but its Praos version [28], which does not, has no evaluation. RBBC only assumes partial synchrony to tolerate unknown delays.

E. Adversarial schedulers

Recent blockchains avoid completely the synchrony assumptions by solving consensus probabilistically [68], [71], [72], [63]. Stellar [63] requires a probabilistic leader election. The HoneyBadger Byzantine Fault Tolerance (HBBFT) [68] is a blockchain that combines proposals by solving ACS and building upon an asynchronous binary Byzantine consensus algorithm [71] that does not terminate under an adversarial scheduler [85]. As we show in §VII-B5, HBBFT is too costly for our needs because each of its nodes creates $n-1$ erasure coded messages and $n-1$ signatures, and verifies $\Omega(n^2)$ signatures. BEAT [30] and Dumbo [65], [43] improve over HBBFT when transactions are respectively less than 10 bytes and 250 bytes but build upon the same consensus algorithm [71]. Some consensus alternatives relax this assumption but require more messages, which risks to increase the overhead [72]. RBBC does not assume a fair scheduler.

F. Faulty leaders

To avoid both the cost often associated with randomization (e.g., common coin) and synchrony, various systems [18], [57], [20], [86], [67], [6], [10], [92], [83] assume partial synchrony [31]. Unfortunately, they all rely on some leader and revert to a costly recovery mechanism in case of failure [21], [12], [7]. Tendermint [50] uses a variant of PBFT and cannot scale beyond tens of nodes [14]. SBFT [38] uses threshold signatures to reduce the communication complexity of PBFT. It is evaluated in a wide area network as a key-value store and as a blockchain system that peaks at 172 TPS. ByzCoin [53] relies on PBFT using multicast trees to reduce the number of messages to $O(n)$. Similar to Bitcoin-NG [34], it also relies on a leader. Unfortunately, the leaderless consensus algorithms that could remain uninterrupted despite single points of failure are incomplete [59] or impractical [12] while others [70], [69] cannot be easily extended with verification sharding to implement a blockchain.

G. Single point of slowdown

HotStuff [92] is a replicated state machine (RSM) that tries to reduce the leader load by sending only the digest of each request. Its implementation [48] exchanges asymptotically as many bits as RBBC per committed transaction but requires clients to send their transactions to all correct consensus participants. Mir [83] is a deduplicating total order broadcast protocol that builds upon our sharded verification [25]. Although not formally stated, it could potentially solve the Set Byzantine Consensus (SBC) problem (Def. 1) as it also leverages multiple proposers. The key difference is that it builds upon the PBFT leader-based consensus algorithm by combining n PBFT proposers (called 'leaders' in Mir) and one leader (called 'primary' in Mir). In Mir, verification sharding may fail due to a single faulty or slow replica, in which case it reverts to full verification. In RBBC, faulty or slow nodes do not stop verification sharding as verification priorities are assigned to replicas so that faster replicas simply verify on behalf of the faulty or slow replicas. The throughput of HotStuff and Mir drops to 0 when their leader is faulty or slow as was reported separately for HotStuff [88] and Mir [83, Fig.10]. As a full-fledged blockchain, RBBC ensures durability (§VI-2) and tolerates failures by always deciding proposals.

H. Human errors

Consensus algorithms are particularly complex, especially when they are monolithic [60], [18], [58]. As blockchains require consensus to handle conflicting transactions from different nodes [42], an erroneous consensus algorithm can lead to dramatic losses. Some algorithms suffer from known errors [1]. Even when formally specified [70], their specification might be too large to be machine-checked and may appear erroneous [84]. Such errors already affected blockchains, for example, when the randomized consensus at the heart of HoneyBadgerBFT was proved non-terminating [85]. An attempt to limit blockchain errors lies on theorem provers [4], [64], however, they check proofs but not algorithms. By contrast, the consensus algorithm of RBBC was formally proved safe and live for any parameters n and $t < n/3$ using complete model checking [9]. To achieve this, its complex (multivalue) consensus protocol decomposes into reliable broadcast [13] and binary consensus using Ben-Or et al.'s reduction [8]. The reliable broadcast has been verified using model checking [56] while the binary consensus was formally verified with model checking thanks to an additional decomposition [9]. Although these verifications depend on the correctness of the model checker, the compiler, etc., it considerably reduces human errors.

III. Goals and Assumptions

The goal is to implement a blockchain system whose performance scales with a number of consensus participants that treat (verify cryptographically and totally order) a large amount of transactions sent by requesters all over the world. The communication model is the classic resilience optimal model with partial synchrony [31] and $t < n/3$ [18].

A. Open permissioned system

We consider an "open" distributed system consisting of nodes that do not need any permission to join, called *replicas* and *requesters* ('requester' is equivalent to 'client' in the distributed computing literature but differs from the notion of 'client' of the Ethereum documentation). The replicas receive blocks from other nodes and maintain a copy of the current state of the blockchain whereas the requesters simply act as clients, requesting balances and issuing transactions. We call this an open "permissioned" system because nodes need some permission to play the roles of *proposers* and *verifiers*. Each proposer collects a set of requested transactions and proposes it periodically as a batch whereas the verifiers check the transaction signatures, a procedure called *verification*. This open permissioned model is appealing for assigning permissions based on proof-of-stake in Ethereum v2.0 or for revoking permissions upon misbehaviors in LLB [78].

B. Transaction model

Let T be the set of all possible transactions and let any transaction $tx \in T$ be a tuple of $\langle a, b, m, \sigma \rangle$ that represents a transaction with non-forgeable signature σ transferring amount m from account a to account b (implemented with UTXO as explained in §IV). We use SSL handshake with certificates listed within blocks for secure channels and new blockchain accounts create new key pairs that they use after they receive coins. Let a proposal $s \subset T$ be a set of transactions. Let $S = T^*$ be the set of possible proposals, or the set of possible sets of transactions. A transaction $\langle a, *, m, \sigma \rangle$ is *provisioned* if the balance of a is larger than m; it is *valid* if it is provisioned and σ is the signature of the owner of a. A set of transactions is valid if all its transactions are valid; it is *non-conflicting* if no subset of its transactions are cumulatively withdrawing more from any account than its balance. Note that the transactions could be multisigned as their execution is not sharded.

Initially, all nodes have a copy of a special block called the *genesis* block of the blockchain that indicates the initial balance of some addresses (i.e., accounts) and the identities of the permissioned nodes identified by their public keys. Note that because permissioned nodes are listed in blocks, they do not have to be "pre-selected" as in traditional consortium blockchains [15] but can instead change periodically to avoid bribery attacks [87].

C. Failure model

The failure model is Byzantine in that *faulty* nodes can fail arbitrarily [77]. We refer to non faulty nodes as *correct*. More specifically, among the n permissioned nodes there are at most $t < \frac{n}{3}$ faulty nodes. Among these n permissioned nodes, the set V of verifiers contains at least $t+1$ correct nodes and the set P of proposers contains at least one correct proposer, which is easily ensured with $|V| = |P| = n$. Note that any number of requesters and replicas that are not part of these permissioned nodes can be faulty. To ensure termination we assume that the communication is partially synchronous [31] in that there exists an unknown global stabilization time after which all messages sent are delivered in less than a fixed amount of time. In order to guarantee that the system is censorship-resistant (Theorem 3) despite Byzantine nodes, we make the following assumptions:

1) *sequential-transaction-requests:* a correct requester does not issue invalid transactions or two conflicting transactions;
2) *bounded-requesting-rate:* the rate at which transactions enter the *mempool* (memory pool) of proposers is lower than the rate at which transactions get proposed by proposers.

Note that Assumption (1) only applies to correct nodes in order to guarantee that their transaction will be eventually treated by the system; the system cannot guarantee that transactions issued by Byzantine requesters will be treated (as their transactions may be invalid). Assumption (2) precludes Denial-of-Service (DoS) attacks where correct proposers would receive too many requests to keep track of them within their bounded memory. We reduce the likelihood of a DoS attack by showing empirically that RBBC treat a large volume of transactions for a long period (§VII).

D. Goal

Our goal is to implement a censorship-resistance replicated state machine (RSM). By censorship-resistance, we mean that any transaction issued properly by a requester gets committed by the system, hence preventing censorship (§II). By replicated state machine, we refer to a way of totally ordering sets of transactions in the form of a blockchain, starting from the genesis block, so that all transactions are provisioned and linearizable [47]. To this end, we require to solve a variant of the BFT consensus problem to agree on an enumerable valid subset of the union of the proposed values as follows.

Definition 1 (Set Byzantine Consensus): Assuming that each correct node proposes an enumerable set of transactions as a proposal, the *Set Byzantine Consensus* (SBC) problem is for each of them to decide on a set in such a way that the following properties are satisfied:

1) SBC-Termination: every correct node eventually decides a set of transactions;
2) SBC-Agreement: no two correct nodes decide different sets of transactions;
3) SBC-Validity: a decided set of transactions is a valid non-conflicting subset of the union of the proposed sets;

4) SBC-Nontriviality: if all nodes are correct and propose an identical valid non-conflicting set of transactions, then this subset is the decided set.

The SBC-Termination and SBC-Agreement properties are common to many Byzantine consensus definition variants, while SBC-Validity is different: it states that transactions proposed by Byzantine proposers could be decided as long as they are valid and non-conflicting. SBC-Validity is inspired by the external validity property [17], [3] that requires a decision to be valid and the idea of deciding at least $t+1$ proposed values [60], [8], [29], however, SBC-Validity cannot result from any combination of these properties. For example, the union of strict subsets of all proposed values is a possible SBC decision. SBC-Nontriviality prevents a trivial algorithm that always outputs an empty set from solving the problem.

IV. Overview of RBBC

This section presents the architecture and the two main novelties of RBBC: its verification leverages the few computational resources when the system is small and its consensus leverages communication to commit more transactions when the system is large and bandwidth becomes limited.

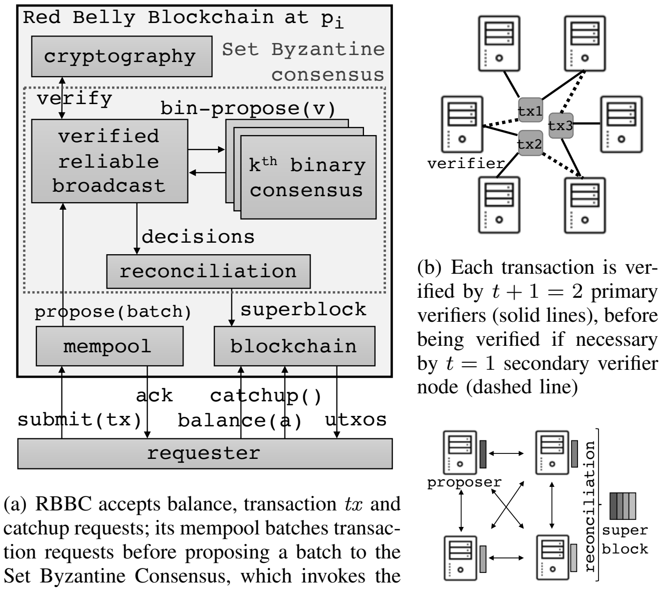
A. Architecture

Figure 1(a) depicts the architecture of RBBC that features a memory pool (or *mempool*), a Set Byzantine Consensus, a cryptography component, a reconciliation component and a blockchain that stores the superblocks. The set Byzantine consensus includes (i) a verified variant of the reliable broadcast, (ii) a reduction from the multivalue consensus problem to the binary consensus problem, (iii) a binary consensus and (iv) a reconciliation protocol to build a superblock from multiple sets of transactions. From time to time the $|P|$ proposers extract some transactions from their mempool that they propose to the multivalue consensus.

B. Reducing the computation at small scale

Verification is needed to guarantee that the *Unspent Transaction Output (UTXO)* transactions [73] are correctly signed. As verifications are CPU-intensive and notably affect performance (as we experiment in §VII-A2), the verification is sharded by letting different verifiers verify distinct transactions without reducing security. The expected result is twofold. First, it improves performance as it reduces each verifier's computational load. Second, it helps scaling by further reducing the per-verifier computational load as the number of verifiers increases. We confirm empirically in §VII-B4 that verification sharding divides the verification load by 3.

As t verifier nodes can be Byzantine, each transaction signature has to be checked by at least $t+1$ verifier nodes. If all the $t+1$ nodes are unanimous in that the signature check passes, then at least one correct node checked the signature successfully and it follows that the transaction is correctly signed. Given that t nodes may be Byzantine, a transaction may need to be verified by up to $2t+1$ times before $t+1$ equal responses are computed. As depicted in Figure 1(b), we map each transaction to $t+1$ *primary verifiers* that eagerly verify this transaction and t *secondary verifiers* that lazily verify this transaction if necessary, hence summing up to $2t+1$ verifiers.



(a) RBBC accepts balance, transaction tx and catchup requests; its mempool batches transaction requests before proposing a batch to the Set Byzantine Consensus, which invokes the verified reliable broadcast and multiple binary consensus instances; when all binary consensus instances have decided, a reconciliation is invoked to group proposals into a decided superblock that is stored in the blockchain

(b) Each transaction is verified by $t+1=2$ primary verifiers (solid lines), before being verified if necessary by $t=1$ secondary verifier node (dashed line)

(c) $n=|P|=4$ proposals depicted with grey rectangles are all combined in the decided superblock

Fig. 1. RBBC architecture (Fig. 1(a)), its verification sharding (Fig. 1(b)) and its superblock optimization (Fig. 1(c))

C. Leveraging bandwidth at larger scales

In a large scale environment, geodistributed proposers are likely to receive different sets of transactions coming from requesters located in their vicinity. Instead of selecting one of these sets as the next block and discarding the others, RBBC combines all the sets of transactions proposed by distinct proposers into a unique superblock to improve the performance (as we quantify in §VII-B5).

In particular, RBBC decides upon multiple proposed sets of transactions. To illustrate why this is key for scalability, consider that each of the n consensus participants have $O(1)$ transactions to propose. As opposed to blockchains based on traditional Byzantine consensus that will decide $O(1)$ transactions, RBBC can decide $\Omega(n)$ transactions. As the typical communication complexity of Byzantine consensus is $O(n^4)$ bits [18], [23], it results that $O(n^3)$ bits are needed per committed transaction in RBBC (cf. Theorem 2), instead of $O(n^4)$ and without suffering from a leader bottleneck. Some leader-based approaches limit the communication complexity by relying on threshold encryption [38], [3], [92], our goal is to avoid more cryptography to limit the CPU load.

To illustrate how RBBC achieves this optimization, consider Figure 1(c) that depicts $n = 4$ permissioned nodes that propose different sets of transactions but that decide a value that is actually a superblock containing the union of all sets of transactions that were proposed. It results from this optimization that the number of transactions decided grows linearly in n as long as each transaction is proposed by a constant number of correct proposers (Theorem 2). Note that some of these transactions may not be executable as they conflict, this is why we need a reconciliation procedure (§V-C).

D. Assigning roles to nodes

We now explain how node roles are assigned for each transaction using a deterministic function. For each consensus instance, we have an ordered set P of permissioned node identifiers where $t < |P| \leq n$, indicating the nodes that play the role of primary or secondary proposers for all transactions. Note that this determinism does not imply predictability as one can change proposers [37] directly, and such changes can also be decided deterministically and anonymously by the consensus nodes themselves with a recent voting protocol also based on DBFT [16]. Although $2t < |P| \leq n$ is necessary to achieve censorship-resistance a requester never needs to send requests to more than $t+1$ proposers.

For a requester to identify the proposers responsible to propose a given transaction tx to the consensus, it executes a deterministic function $\mu(a)$ that takes as input the source account a of tx and returns the identifier of a node $p_i \in P$, called the *primary proposer* of transaction tx. To guarantee that a transaction is proposed (despite a faulty primary proposer), between t and $n-1$ secondary proposers distinct from p_i are also selected deterministically. The number of proposers of each transaction tx is at least $t+1$ to guarantee that tx will be proposed by at least one correct proposer (Theorem 3). The number of proposers can be as large as n, however, fewer proposers lower latency whereas more proposers increase throughput, as we will experiment in §VII-B.

As each transaction must be verified between $t+1$ and $2t+1$ times, each proposer p_i is also mapped to a set of $t+1$ $primary\_verifiers_{p_i}$ and a set of t additional $secondary\_verifiers_{p_i}$. The $primary\_verifiers_{p_i}$ include p_i itself and verify upon reception the signatures of tx, so the basic design makes some nodes both proposers and verifiers. If the verification returns the same $t+1$ results, then it becomes clear whether the signature of tx is correct. If not, t additional verifications are needed to identify the majority of $t+1$ identical responses indicating whether the signature of tx is correct. This is why, the $secondary\_verifiers_{p_i}$ set includes nodes of P that are distinct from the $primary\_verifiers_{p_i}$ and that verify tx in case one or more of the primary verifiers are slow/faulty. One could take more verifiers but would waste CPU (§VII-A2).

V. Design and Implementation

In this section, we detail the design and implementation of RBBC. Requesters request the balance of an account or send a transaction to $t+1$ proposers. All communications are exchanged through SSL-encrypted channels. The following methods are exposed by the permissioned nodes through a JSON RPC to the requesters.

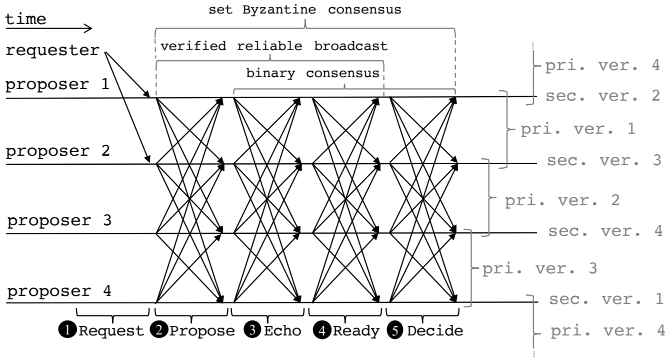


Fig. 2. The typical message pattern of the consensus protocol between each proposer i and the permissioned nodes that play the role of primary verifiers i (pri. ver. i) and secondary verifiers i (sec. ver. i) for proposer i

- submit(tx) runs at proposer nodes and takes as input a UTXO transaction tx of about 400 bytes from a requester. If tx is provisioned, does not exist already and does not conflict with any transaction of the mempool, then it is placed in the mempool and true is returned. A correct requester calls this method at $t+1$ proposers.
- balance(a) takes account a as an input and returns its UTXOs. A requester performs this operation by contacting different proposers until $t+1$ equal notifications are received. Not only does it allow small devices to securely consult the state without downloading the blockchain, but it also guarantees the integrity of the ledger.
- catchup() is a method for lagging or recovering replicas to get updated about the current index of the blockchain.

A. Normal consensus execution

Figure 2 depicts a normal consensus execution when $n = |P| = 4$ and $t = \lceil n/3 \rceil - 1 = 1$ where a single requester sends one request to $t+1$ proposers for simplicity (actually, many requesters typically request proposers in parallel) and each one of the proposers executes the following:

❶ **Request.** A requester computes $\mu(a)$ with the source account a of transaction tx to retrieve the mapped proposers and verifiers of tx and sends the request for transaction tx to the $t+1$ primary proposers and verifiers. Upon reception, tx is added to the mempool and is verified by the primary verifiers.

❷ **Propose.** Each proposer selects, from their mempool, transactions (i) for which it is the primary proposer and (ii) in decreasing order of their *age* or the number of blocks appended to the chain since these transactions arrived. This batch is proposed to the consensus by sending it in an INIT message to all other permissioned nodes. (Note that a proposer with an empty mempool starts the consensus with an empty proposal if it receives a non-empty proposal from another node to guarantee sufficiently many participants.)

❸ **Echo.** Upon reception, permissioned nodes broadcast a digest of each received proposals in ECHO messages.

- ❹ **Ready.** Upon reception of $n-t$ equal ECHO messages, verifiers for the received proposals verify them (if not already done in the request step) and send the result in a READY message (§V-B) to all proposers. Upon reception of $t+1$ equal READY messages, a node broadcasts the READY message if it has not done so already. (This step is represented by a single message exchange in Figure 2 due to the fast Ready-Decide optimization presented below.)
- ❺ **Decide.** Upon reception of $2t+1$ equal READY messages (§V-B) for a particular proposal, the nodes store this proposal in an array indexed by the sender id and input 1 to a corresponding binary consensus instance. This is repeated for other proposals until $|P|-t$ binary consensus instances decide 1, after what a reconciliation (§V-C) combines into a superblock every valid transaction of proposals for which a binary consensus output 1.

a) Good execution complexity and scalability: In good executions, RBBC differs from previous work by committing securely $\Omega(n)$ transactions within 4 message delays (❷–❺) thanks to the Ready-Decide optimization: In the *Ready step*, if the verifiers are fast enough to verify before the reception of $t+1$ equal READY messages, then they broadcast a READY message. As a result, all nodes receive $2t+1$ equal READY messages in a single message delay allowing them to enter the *Decide step* by inputting 1 to a binary consensus instance directly. At the end, the proposers pick the transactions from the $\Omega(n)$ proposals for which the binary consensus decided 1 and reconciliate them into a superblock. The complexity and throughput of RBBC are computed in §V-C and Theorem §2.

B. Verified all-to-all reliable broadcast

For proposers to exchange verified proposals we add a `secp256k1` *Elliptic Curve Digital Signature Algorithm (ECDSA)* verification to the reliable broadcast, which is originally a 3-step one-to-all communication abstraction exchanging INIT, ECHO and READY messages where any message delivered to a correct proposer gets eventually delivered to all correct proposers [13].

```
1:  verified-reliable-broadcast($v$):      ▷ verified variant of reliable broadcast at $p_i$
2:      broadcast(INIT, $v$)                                 ▷ broadcast value $v$ to all
3:  upon receiving a message (INIT, $v$) from $p_j$:
4:      broadcast(ECHO, $h(v), j$)                           ▷ echo the hash digest of $v$
5:  upon receiving $n-t$ (ECHO, $h(v), j$) msgs and not having sent READY:
6:      if $p_i \in primary\_verifiers(v)$ then $verif \leftarrow$ verify($v$)
7:      if $p_i \in secondary\_verifiers(v)$ then wait($\Delta$); $verif \leftarrow$ verify($v$)
8:      broadcast(READY, $verif, h(v), j$)                    ▷ piggyback verifications
9:  upon receiving $t+1$ (READY, $verif, h(v), j$) and did not send READY:
10:     stop-verify($v$)                          ▷ prevent unnecessary verifications
11:     broadcast(READY, $verif, h(v), j$)                    ▷ piggyback verifications
12: upon receiving $n-t$ (READY, $verif, h(v), j$) and not delivered from $j$:
13:     if is-verified($v, verif$) then deliver($v, j$)  ▷ deliver if sufficiently verified
```

Our verified variant of the reliable broadcast adds a verification function verify (lines 6 and 7) before the broadcast of the READY message. A proposer broadcasts an INIT message with a proposal v (line 2). Upon reception, its digest $h(v)$ is broadcast in an ECHO message (line 4); we use the SHA256 digest to save bandwidth as proposals can contain thousands of transactions (§VII-B). Upon reception of $n-t$ equal ECHO messages, the verification of the proposal starts first at the $primary\_verifiers(v)$ and later, if necessary, at the secondary ones. After the verification, a list $verif$ of integers indicating the indices of invalid transactions in the proposal is appended to the READY message, which is then broadcast (line 8) with the digest of the corresponding proposal. After receiving the same $verif$ field for $h(v)$ from $t+1$ distinct processes (line 9), a node knows which transactions of v are valid. Upon reception of $n-t$ equal READY messages, v is delivered if it contains valid transactions (line 13). The proof that the verified reliable broadcast ensures the properties of reliable broadcast for each valid value (and discards invalid values) relies on the fact that the sets Q of correct proposers and Q' of verifiers are such that $Q \cap Q' \geq t+1$, guaranteeing that the READY message with the same $verif$ will be sent at line 11 (cf. Theorem 4).

C. Agreeing on a superblock

We modify Ben-Or, Kelmer and Rabin's reduction (lines 14–23) of the multi-value consensus problem to the binary consensus problem [8] to solve the Set Byzantine Consensus (§III) by replacing the reliable broadcast by our verified reliable broadcast (§V-B) and invoking a reconciliation to decide a superblock of non-conflicting provisioned transactions (§III). Symbol → indicates that the array $props$ gets populated in the background by all concurrent reliable broadcasts (line 15). For each of the first proposals delivered at p_i by the verified reliable broadcasts (§V-B), p_i proposes 1 to a binary consensus instance (line 19). Proposer p_i proposes 0 to the remaining binary consensus instances (line 21) after a timer expires and $|P|-t$ consensus return 1. This timer (line 16) increases with the age of the oldest transaction of the mempool to potentially decide it.

```
14: propose($val$):          ▷ set Byzantine consensus at $p_i$ with $val$ a batch of txs
15:     verified-reliable-broadcast($val$) → $props$       ▷ exclude invalid txs §V-B
16:     start-timer(age of oldest tx in mempool)                ▷ give time to slow ones
17:     while $|\{k : bitmask[k] = 1\}| < |P| - t$ or $timer$ did not expire do
18:         for all $k$ such that $props[k]$ has been delivered          ▷ for all delivered
19:             $bitmask[k] \leftarrow$ bin-propose$_k(1)$ ▷ propose 1 to $k^{th}$ binary consensus
20:     for all $k$ such that $props[k]$ has not been delivered        ▷ for undelivered
21:         $bitmask[k] \leftarrow$ bin-propose$_k(0)$   ▷ propose 0 to $k^{th}$ binary consensus
22:     wait until $bitmask$ is full and $\forall \ell$, $bitmask[\ell] = 1 : props[\ell] \neq \varnothing$
23:     reconciliate($bitmask$ & $props$)                    ▷ combine into a superblock
```

All binary consensus instances proceed in parallel (their invocation is non-blocking). The decisions of these binary consensus instances constitute a $bitmask$ that is applied to the set of potentially decidable proposals (line 23). Although the array of verified proposals may differ across correct nodes, the bitmasks of all correct nodes are guaranteed to contain 1s (Lemma 5) and be identical due to the agreement properties of the binary consensus. Note that even though the proposal may not be known yet for some of these indices, it is guaranteed by the reliable broadcast to be eventually delivered at all correct proposers (§V-B). Each correct proposer waits until a proposal has been delivered at each of these indices (line 22), then each correct proposer obtains the same set of proposals after applying the $bitmask$. The unselected proposals are stored in the mempool for later.

1) Reconciliation: To fill the superblock, we reconciliate the decidable set of proposals (lines 24–29). Correct proposers extract deterministically the transactions by going through all proposals and through each of their transactions one-by-one, to add the non-conflicting provisioned ones to the superblock. With the UTXO model, one can easily ensure all transactions are provisioned and non-conflicting by implemeting conflict($tx, *$) that executes transaction tx and returns false if all the UTXOs tx consumes exist and otherwise returns true. For the sake of fairness (i.e., to not favor any particular proposer), correct proposers traverse the proposals from the index number ($k \mod n$) to index number ($k-1 \mod n$) where k is the index of the last superblock in the blockchain. This prevents the proposer with the lowest index number from having its proposed transactions added to the superblock with a higher priority than the transactions of other proposers. (We will show in §VII-B that the computation needed for reconciliation is negligible compared to the signature verification.)

```
24: reconciliate($props$):                          ▷ combine proposals, initially $superblock = \varnothing$
25:   for $i = 0..(n-1)$ do                          ▷ use the last block index $k$ for fairness
26:     for $tx \in props[(k+i) \bmod n]$ do        ▷ starting from the first tx of proposal
27:       for $ctx \in superblock$ do                ▷ for all committed transactions
28:         if $\neg$conflict($tx, ctx$) then $superblock \leftarrow superblock \cup \{tx\}$
29:   decide($superblock$)                           ▷ decide superblock of non-conflicting transactions
```

2) Binary consensus: To solve the binary consensus deterministically, we chose the binary consensus of DBFT [23] because it is resilience optimal, time optimal and was recently verified with model checking [85], [9]. Each replica refines an estimate value, initially its input value to the consensus, across consecutive rounds until it decides (line 45). It invokes broadcast primitives that deliver some values into a dedicated variable pointed out by $\rightarrow$ at lines 32, 36 and 39. The bv-broadcast (line 32) is a reliable broadcast for binary values [72]. (We optimize by piggybacking it for $r = 1$ with the verified-reliable-broadcast at line 15.) One replica per round acts as a coordinator by broadcasting its value c (line 36) that others prioritize (line 38) to help them converge to the same decision. Hence, RBBC is leaderless with multiple coordinators. The binary values are then forwarded in AUX messages (line 39) and each replica waits to receive a sufficiently represented set of these AUX values (lines 40–42). If only one value is sufficiently represented (line 43) and if it corresponds to the parity of the round, then it is decided (line 45). Otherwise, val is set to the parity of the round and another round starts.

3) Complexity: In the worst case, each bin-propose decides within $O(t)$ rounds after the network stabilizes and messages start being delivered in bounded time, which is optimal [35]. Hence, as the (multivalue) propose needs a constant additive factor of message delays, its time complexity is asymptotically optimal. Moreover, propose is resilience optimal as it tolerates any $t < n/3$ failures [60]. It has the same communication complexity $O(n^4)$ as PBFT [18] and DBFT [23] but decides up to n times more transactions than them in good executions: $O(n^3)$ bits exchanged per committed transaction batch. Recall that the communication complexity of PBFT is $O(n^4)$ bits because there are at most $t+1$ view-change rounds, the message in each round contains the state received from the previous view-change rounds, which is $O(t)$ bits, and is broadcast by n nodes, leading to $(t+1) \cdot O(t) \cdot (n-1) \cdot n = O(n^4)$.

```
30: bin-propose($val$):                              ▷ binary consensus at $p_i$ with $val \in \{0, 1\}$
31:   loop:                                          ▷ loop that starts with round $r = 1$
32:     (bv-broadcast(EST, $r$, $val$) $\rightarrow$ $cvals$)  ▷ reliable bcast if not done at l.15
33:     start-timer($r$)                             ▷ timeout increases with rounds
34:     if $i = r \bmod n$ then                      ▷ coordinator rebroadcasts
35:       wait until ($cvals = \{w\}$)               ▷ $cvals$ stores delivered values
36:       broadcast(COORD, $r$, $w$) $\rightarrow c$  ▷ coordinator broadcasts
37:     wait until ($cvals \neq \varnothing \wedge$ timer expired)  ▷ wait enough time
38:     if $c \in cvals$ then $e \leftarrow \{c\}$ else $e \leftarrow cvals$  ▷ prioritize coord value
39:     broadcast(AUX, $r$, $e$) $\rightarrow bvals$  ▷ broadcast these values
40:     wait until $\exists s \subseteq bvals$ where the two following conditions hold:
41:       • $s$ contains contents received from at least $n-t$ distinct nodes
42:       • $\forall v \in s, v \in cvals$           ▷ every value in $s$ is in $cvals$
43:     if $s = \{v\}$ then                          ▷ if there is only one value in $s$
44:       $val \leftarrow v$                         ▷ adopt this singleton value
45:       if $v = (r \bmod 2)$ and not decided yet then decide($v$)  ▷ decide
46:     else $val \leftarrow (r \bmod 2)$            ▷ otherwise, adopt the current parity bit
47:     if decided in round $r-2$ then exit()        ▷ help others in two last rounds
48:     $r \leftarrow r+1$                           ▷ increment the round number
```

VI. Correctness and Analysis

To explain how RBBC implements a secure, fair and scalable blockchain, we first show that it disallows double-spending by implementing an RSM that stores exclusively valid non-conflicting transactions to reliable storage (Theorem 1). Finally, we explain why its throughput scales with the number of nodes (Theorem 2) and show that RBBC offers censorship resistance (Theorem 3). The proofs are deferred to the appendix (§A).

1) Correctness: To avoid forks that could lead to double-spending, RBBC executes consensus instances in a totally ordered sequence and at the end of each instance the decided superblock orders all transactions it contains in the same order at all replicas. Note that to implement a blockchain system, RBBC offers stronger guarantees than a simple RSM, by for example discarding the incorrectly signed transactions eagerly (line 13) and the conflicting transactions lazily (lines 28).

Theorem 1 (Replicated State Machine): In RBBC, all correct replicas observe the same sequence of committed transactions, which are all valid and non-conflicting.

2) Durability: To ensure durability [11], the remaining transactions are stored in a block in an append only log on disk. For recovery, each node keeps a write-ahead log containing the messages it has broadcast during at least the previous two committed multivalued consensus instances, older messages being garbage collected. To ensure transactions are verified and committed quickly, the UTXO table is stored in memory. To minimize the size of the UTXO table, transactions should consume all UTXOs for their account. After a crash, nodes can reconstruct their UTXO table by parsing their blocks from disk. Nodes that need to recover messages from older consensus instances simply collect $t+1$ equal instances of the decided block.

3) Scalability: We explain the scalability of RBBC by showing that RBBC commits $\Omega(n)$ transactions per consensus instance in good executions where BFT algorithms (e.g., PBFT, DBFT, Tendermint, HotStuff) that do not solve SBC (Def. 1) commit $O(1)$ transactions. Note that we could have modified leader-based consensus algorithms (PBFT, Tendermint, HotStuff) for their leader to batch $\Omega(n)$ times more requests, however, this would have added up to the leader load. More dramatically, the leader would have sent $\Omega(n)$ bytes to $n-1$ nodes, which would have taken a quadratic amount of time in a WAN, where no Ethernet broadcast is available and the message authentication codes (MACs) optimization of PBFT can thus not be used. Instead, RBBC commits $\Omega(n)$ transactions as explained below.

Theorem 2 (Scalable throughput): Let m be the number of transactions proposed to the Set Byzantine Consensus by each proposer and let $n = |P|$ be the number of proposers. Assume that proposals are all reliably delivered before the algorithm times out (line 17) but that all $\lceil \frac{n}{3} \rceil - 1$ Byzantine proposers do not propose any transaction. The Set Byzantine Consensus commits between $2m$ (as n tends to infinity) and $\Omega(n)$ distinct transactions.

4) Deduplication: If a Byzantine client can approximate closely the age parameter and the timing of the system, then more transactions can be proposed twice and their duplicates may persist until the reconciliation phase, after which one of them is discarded (line 28) as they necessarily conflict. One way to reduce the ability of Byzantine to create duplicates is for each node to choose the age parameter for each transaction at random as a small constant. Another is to apply this randomization only for clients that are duplicating transactions. Note that some recent efforts [83] were devoted to eliminate duplicated transactions during the consensus execution by adopting a more conservative approach where each transaction can only be proposed by a single proposer at a time. If the proposer of this particular transaction fails, then an epoch change happens and another one is selected. Although the benefit is to eliminate duplicates during the consensus execution, the drawback is to increase linearly this transaction latency as up to t epoch changes may be necessary before proposing it correctly.

Theorem 3 (Censorship-Resistance): Every transaction sent by a correct requester is eventually included in a superblock.

Censorship-resistance differs from other notions of blockchain liveness, validity or resilience by offering guarantees to a correct requester [68], [19], [43] and without requiring its transaction to be sent to all replicas [36], [5]. It is important to note that censorship-resistance does not require all requesters to be correct, it simply ensures that if a requester follows the protocol to submit its transaction, then this transaction is guaranteed to be committed by the system. By contrast, a Byzantine requester not following the request protocol, does not have this guarantee.

VII. Experimental Evaluation

In this section, we show that RBBC scales up to hundreds of Amazon EC2 VMs running consensus located on different continents and replicating the blockchain state to up to 1000 machines in 14 separate regions. To this end, we compare the performance of (1) *RBBC* with its verification and its consensus as depicted in §V, *HBBFT* which corresponds to the original code of the HoneyBadgerBFT protocol as made available by its authors [68] and (3) *CONS1* that corresponds to a variant of RBBC with the classic 3-step leader-based BFT algorithm of PBFT taken from [26] with BFT-SMaRt optimizations [18], [50], [10].

We ran four types of experiments with parameters from Table II: *(i)* with a varying fault tolerance and verification in a geodistributed environment (§VII-A1); *(ii)* with all three blockchains on low-end machines (§VII-B); *(iii)* with up to 1000 replicas all updating their copy of all account balances (§VII-C); and finally *(iv)* with Byzantine failures (§VII-D).

1) Machine specifications: We ran the blockchains on all the 14 Amazon datacenters that we had at our disposal at the time of the experiment, North Virginia, Ohio, North California, Oregon, Canada, Ireland, Frankfurt, London, Tokyo, Seoul, Singapore, Sydney, Mumbai, São Paulo. Each pair of datacenters is separated by a specific delay and bandwidth listed in §VII-C. We tested three VM types: (1) *high-end* c4.8xlarge instances with an Intel Xeon E5-2666 v3 processor of 18 hyperthreaded cores, 60 GiB RAM and 10 Gbps network performance when run in the same datacenter where storage is backed by Amazon's Elastic Block Store (EBS) with 4 Gbps dedicated throughput; (2) *mid-end* c4.4xlarge instances with an Intel Xeon E5-2666 v3 processors with 16 vCPUs and 30 GiB RAM with "high" network performance (as defined by Amazon), 2 Gbps EBS dedicated throughput; (3) *low-end* c4.xlarge instances with an Intel Xeon E5-2666 v3 processor of 4 vCPUs, 7.5 GiB RAM and "moderate" network performance, and 750 Mbps EBS dedicated throughput. To limit the bottleneck effect on the leader of PBFT, we always place the leader in the most central (w.r.t. latency) region, Oregon. When not specified, proposals contain 10,000 transactions and t is set to $\lceil \frac{n}{3} \rceil - 1$.

2) Leader-based (CONS1) and randomized BFT (HBBFT): CONS1 is the classic 3-step leader-based Byzantine consensus implementation similar to PBFT [18], the Tendermint consensus [50], and including the concurrency optimizations of BFT-SMaRt [10]. To reduce network consumption CONS1 is implemented using digests in messages that follow the initial broadcast. Both CONS1 and HBBFT variants use a classic verification, as in traditional blockchain systems [73], [91], that takes place at every proposer upon delivery of the decided superblock from consensus. HBBFT uses a common coin [71] and reliable broadcast with erasure codes.

A. Peak scalability and leaderless fault tolerance

In a leaderless case, the first messages from $n-t$ replicas determine the performance, so a lower t can yield lower

notation	meaning	value	motivation
μ	compute primary proposer	$0 \leq id < n$	avoid conflicting transactions to different proposers
β	block size	$1 \leq \beta \leq 10,000$	adjust the block size impact on network delay
P	set of proposers	$t+1 \leq \lvert P \rvert \leq n$	fewer (resp. more) proposers commits faster (resp. more)
τ	transaction signature	ECDSA	require smaller keys and signatures than RSA for similar security

TABLE II
PARAMETERS USED IN THE EXPERIMENTS

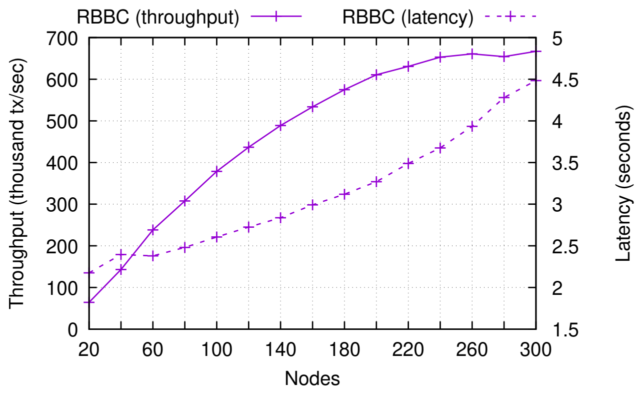


Fig. 3. The performance (latency and throughput) of RBBC in a single datacenter

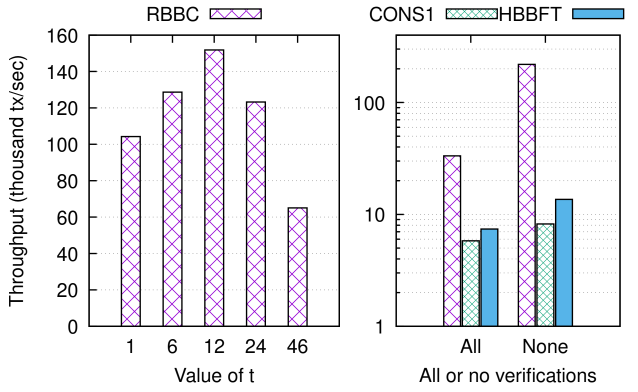


Fig. 4. Impact of fault tolerance on RBBC and verification on 3 blockchains with $n = 140$ geodistributed machines

performance. We stress test RBBC in a single datacenter with up to 300 high-end VMs and a fixed fault tolerance to measure how fast it could go in a consortium setting. To this end, we fix t to the largest possible fault tolerance with $n = 20$ nodes and increase the number of nodes from 20 to 300 permissioned nodes in steps of 20. The results, shown in Figure 3, indicate that the throughput scales to hundred of nodes with a practical latency: the throughput scales up to $n = 260$ nodes to reach 660,000 TPS while the latency remains lower than 4 seconds. At $n = 280$ throughput drops slightly. We discuss below the impact of varying t on performance.

1) Impact of fault tolerance without a leader: Next we evaluate the performance when running 10 high-end VMs in each of the 14 regions for a total of 140 machines. We varied t from the minimum to its maximum value ($46 < \frac{140}{3}$) with sharded verification as depicted in Figure 4 (left).

The peak throughput of 151,000 TPS is achieved with the fault-tolerance parameter $t = 12$. When $t \leq 6$, performance is limited by the $(t-1)^{th}$ slowest node as the consensus waits for a higher number of $n - t$ proposers. The peak throughput occurs while waiting for $n - t = 128$ nodes, probably because it avoids waiting for any node of the slowest region (Table III), São Paulo. When $t \geq 24$, performance tends to be limited and drops further as the fault tolerance t keeps increasing. We conjecture (and show below) that the $t + 1$ necessary cryptographic verifications induce a higher computational load as t increases. As mentioned in Section II, alternative leaderless Byzantine consensus algorithms lack details [59] or have exponential complexity, which would be impractical [12].

2) Impact of verification sharding: To verify the conjecture that more verifications slow performance down, we ran additional experiments and measured the performance with different numbers of verifications per transaction. As depicted in Figure 4 (right), we compared all three blockchains with all nodes verifying all transactions (all) and without any verification (no verification). The performance of all blockchains is higher without verification than with full verification. RBBC is the most affected, dropping from 219,000 TPS to 33,000 TPS while HBBFT and CONS1 throughputs drop less but from a lower peak. As we will show, there are factors other than verification, like the use of a leader and erasure codes §VII-B3, that have a larger impact on these algorithms, yet this confirms that our previous conjecture was correct.

B. Scaling throughput up to hundreds of low-end machines

We now experiment on up to 240 low-end VMs evenly spread on 5 datacenters in Europe (Ireland and Frankfurt) and the United States (Oregon, Northern California, and Ohio). Following up on our previous verification observations, we precisely measured the CPU usage with go `pprof` on microbenchmarks and confirmed that the workload could be CPU-bound. In particular, we measured that dedicating the 4 vCPUs of these low-end instances led to verify about 7800 serialized transactions per second with 97% of CPU time spent verifying signatures and 3% spent deserializing and updating the UTXO table.

1) RBBC vs. a leader-based BFT blockchain: Figure 5 shows the throughput and latency of RBBC with $t + 1$

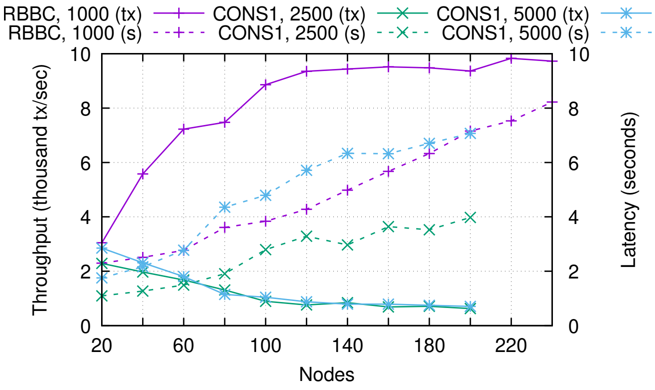


Fig. 5. The performance of CONS1 with batching and RBBC with $t+1$ proposer nodes; the number following the algorithm name represents the number of transactions in the proposals; solid lines represent throughput, dashed lines represent latency

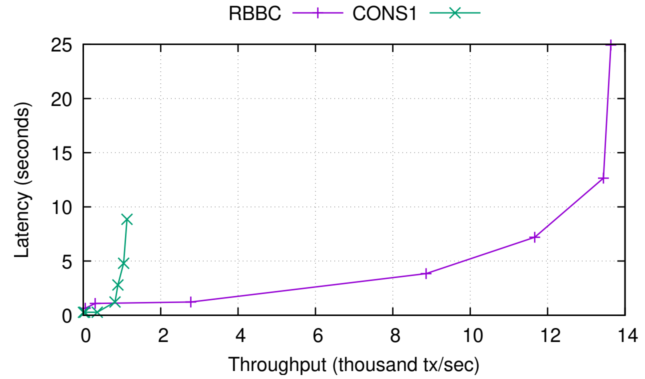


Fig. 6. Comparing throughput and latency of CONS1 and RBBC with $t+1$ proposer nodes on 100 geodistributed nodes; each point represents the number of transactions in the proposals, either 10, 100, 1000, 2500, 5000 or 10000

proposers and CONS1 with different sizes of proposals. As CONS1 is limited to a single proposer (its leader) while RBBC supports multiple proposers, we tested whether CONS1 performance would be better with batching more transactions per proposal than RBBC. With proposal size of 1000, RBBC throughput increases from 3000 TPS to 9000 TPS because of the additional resources and proposals of the growing number of nodes. It flattens out around 10,000 TPS while latency increases from 2 to 8 seconds. By contrast, CONS1 throughput decreases as the number of nodes increases, despite larger proposals.

2) Latency vs. throughput at 100 nodes: To better understand the difference in performance of RBBC compared to the leader-based approach, we depicted on Figure 6 the evolution of the latency as a function of throughput at a reasonable number of consensus nodes, $n = 100$ and different proposal sizes of 1 to 5000. We clearly see that the throughput of CONS1 reaches a limit of about 1100 TPS while RBBC approaches 14,000 TPS, which indicates a 12-fold speedup of RBBC over CONS1. CONS1 has a better minimum latency of 270 ms compared to 640 ms for RBBC for proposals of size 1 but CONS1 latency explodes rapidly. (HBBFT does not appear due to lower performance.)

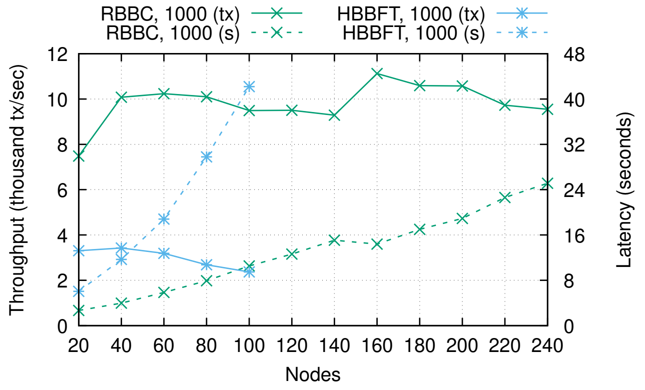


Fig. 7. The performance of HBBFT and RBBC with n proposer nodes. The number following the algorithm name represents the number of transactions in the proposals; solid lines represent throughput, dashed lines represent latency

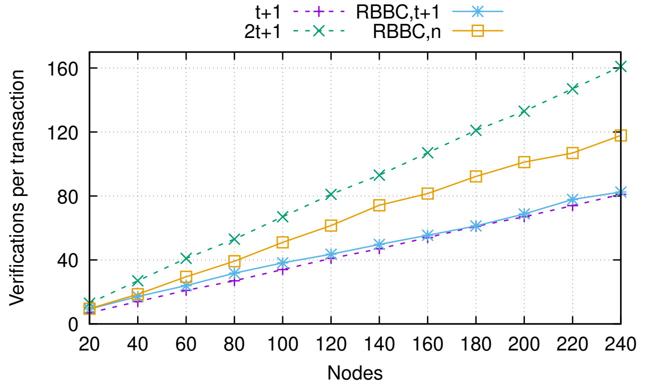


Fig. 8. The number of times a transaction is verified in RBBC with proposal size of 100 transactions, with either $t+1$ or n proposer nodes; the dashed lines $t+1$ and $2t+1$ represent the minimum and maximum number of possible verifications

3) RBBC vs. a randomized BFT blockchain: Figure 7 depicts the performance of RBBC and HBBFT with n proposers, with proposal sizes of 100 and 1000 transactions. With a larger portion of proposers the throughput and latency of RBBC increases faster. With n proposers, the throughput peaks at 11,124 TPS and latency reaches 25,100 ms with 240 nodes, while with $t+1$ proposers the throughput was lower and the latency was much lower. HBBFT performance degrades as the number of nodes increases: latency increases and throughput decreases (we omit latencies beyond $n = 100$ as they reach minutes). This is because each node broadcasts $n-1$ erasure coded messages with as many distinct signatures to distinct nodes that must be echoed, yielding $\Omega(n^2)$ verifications per node.

4) Reducing verification count helps scaling: To better measure the performance gain of verification sharding, we

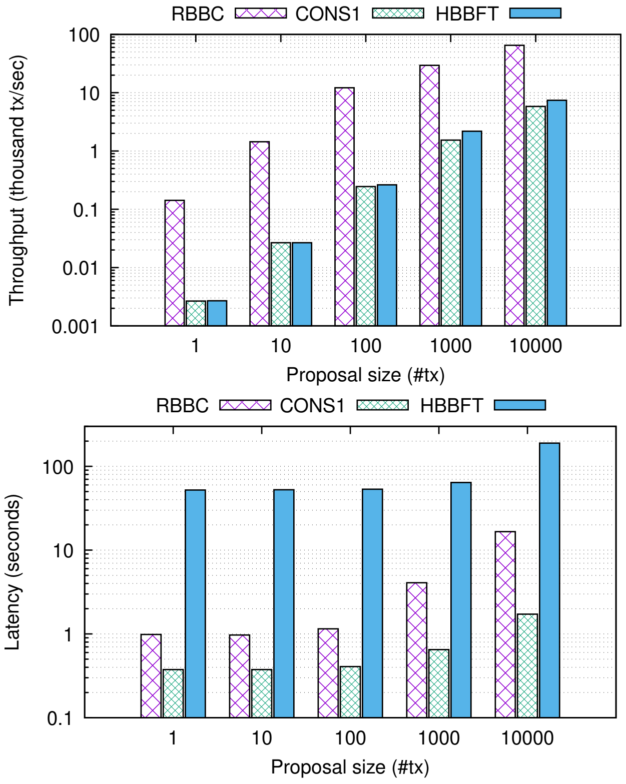


Fig. 9. Throughput and latency comparison of the blockchain solutions with $n = 140$ and $t = 46$, and proposal sizes of $1, 10, 100, 1000$ and 10000 transactions

recorded the average number of times a transaction was verified for $t+1$ and n proposer nodes. The results are shown in Figure 8 where we observe that with $t+1$ proposers the number of verifications stays close to the optimal, while with n proposers the number of verifications remains around the middle of $t+1$ and $2t+1$. This is likely due to the increased load on the system causing verifications to occur in different orders at different nodes. This tends to confirm that verification sharding is important for scalability.

5) Comparing the blockchains: Figure 9 explored the effect of deciding the unions of proposals when running the blockchain. CONS1 has the lowest latency because in all executions the leader acts correctly, allowing it to terminate in only 3 message delays, where RBBC requires 4 message delays. Due to its inherent concurrency, RBBC offers the best latency/throughput tradeoff: at 1000 ms latency, RBBC offers 12,100 TPS whereas at 1750 ms latency, CONS1 offers only 5800 TPS. Note that RBBC does not feature the classic costly view change [21], [12], [7], [92] and its latency remains similar despite faults (cf. §VII-D). Finally, HBBFT has the worst performance because its consensus [71] is randomized and it uses erasure codes: each node spends over 200 ms to compute 1000 transactions for each of the 140 proposals of this experiment. Again this confirms the important CPU load induced by the signature verifications.

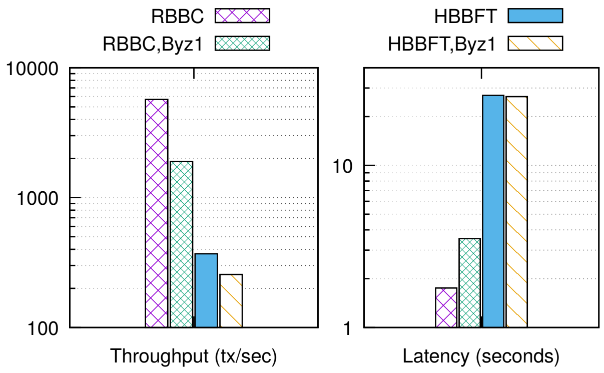


Fig. 10. Comparing throughput and latency of RBBC and HBBFT, with normal and Byzantine behaviors on 100 geodistributed nodes; all n nodes are making proposals of 100 transactions

C. Evaluation with 1000 VMs

Before spawning 1000 VMs to confirm RBBC scalability, we measured the variation of latencies and bandwidth between our 14 Amazon EC2 datacenters (cf. Table III). The minimum latency is 11 ms between London and Ireland, whereas the maximum latency is 332 ms observed between Sydney and São Paulo. Bandwidth between Ohio and Singapore is measured at approximately 64.9 Mbits/s (with variance between 6.5 Mbits/s and 20.4 Mbits/s).

To avoid wasting bandwidth, we segregated the roles: all 1000 VMs act as servers, keeping a local copy of the balances of all accounts. On these replicas, 10 clients per 840 low-end machines (60 VMs in each of 14 datacenters) send transactions and 160 high-end machines (40 machines in each of the Ireland, London, Ohio and Oregon datacenters) decide upon each superblock. Each of the 8,400 clients start with 100 UTXOs of size 64 bytes each (for a state database of size 51.27 MiB) and each proposal contains up to 1000 transactions. The resulting performance is depicted in Table IV. Interestingly, the throughput is only around 30,000 TPS but this is not due to the low capacity of RBBC but due to the difficulty of generating the workload: the replicas are located in 14 different datacenters and have to wait for owning a UTXO before they can request a transaction that consumes it. The asynchronous write latency measures the time a proposer acknowledges a transaction reception. Importantly, the transaction commit time (latency) remains about 3 seconds despite the large traffic.

D. Experiments under Byzantine attacks

We evaluate RBBC under 2 Byzantine attacks:

Byz1 The payload of the reliable broadcast messages is altered so that no proposal is delivered for reliable broadcast instances led by faulty proposers. To this end, the binary payloads of the binary consensus messages are flipped. The goal of this behavior is to reduce throughput and increase latency.

Byz2 The Byzantine proposers form a coalition in order to maximize the bandwidth cost of the reliable broadcast

	Tokyo	Seoul	Mum.	Singa.	Syd.	Cana.	Frank.	Ireland	Lond.	São P.	N.Virg	Ohio	N.Cal.	Oregon
Tokyo	0	551	129	240	161	106	74	66.4	59	55.4	90.1	96.2	129	132
Seoul	33	0	137	157	141	91.5	54	60.8	54.7	56.6	84.2	114	84.2	116
Mumbai	133	164	0	121	67	90.9	176	178	145	46.7	81.9	80.5	69.1	64.2
Singapore	69	100	67	0	90.9	83.2	90.7	86.1	90.4	40.8	59.5	64.9	80.5	77.3
Sydney	106	135	235	170	0	77.1	61.3	53.8	51.2	40.2	74.9	99.7	135	119
Canada	166	185	196	220	225	0	166	250	164	159	808	760	205	168
Frankfurt	244	275	112	178	292	102	0	477	823	92.9	222	220	144	85.7
Ireland	226	246	122	188	286	78	25	0	829	114	185	183	104	117
London	255	284	111	179	281	90	15	12	0	107	190	195	107	85.5
São Paulo	271	293	302	328	332	125	210	184	192	0	131	124	77.7	81.7
N. Virginia	162	209	182	238	205	15	89	85	76	122	0	827	232	186
Ohio	169	199	193	227	196	25	99	91	87	131	13	0	428	219
N. California	120	150	262	178	148	76	148	142	138	182	64	52	0	681
Oregon	105	135	235	163	162	66	164	141	158	183	76	71	22	0

TABLE III
HEATMAP OF THE BANDWIDTH (MBPS) IN THE TOP RIGHT TRIANGLE AND LATENCY (MS) IN THE BOTTOM LEFT TRIANGLE BETWEEN 14 AWS REGIONS

#Replicas	#Requesters	Valid-tx/sec	Async write latency(ms)	Latency(ms)	Valid-tx/superblock	Invalid-tx/superblock
1000	8400	30684	238	3103	95407	378

TABLE IV
PERFORMANCE OF RBBC WITH 1000 REPLICAS SPREAD IN 14 DATACENTERS

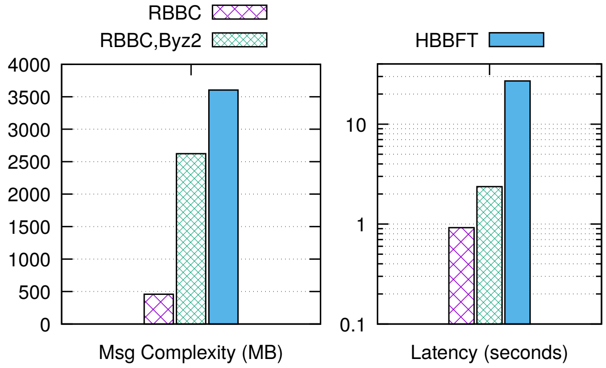


Fig. 11. Comparing bandwidth usage and latency of RBBFT and HBBFT with normal and Byzantine behaviors on 100 geodistributed nodes

using the digests described in §V-B. As a result, for any reliable broadcast initiated by a Byzantine proposer, $t+1$ correct proposers will deliver the full message while the remaining t will only deliver the digest of the message, meaning they will have to request the full message from $t+1$ different proposers from whom they receive ECHO messages.

Experiments are run with 100 low-end machines using the same 5 datacenters from US and Europe and with n proposers (as in §VII-B). Figure 10 shows the impact of Byz1 on performance with n proposers and proposal sizes of 100. RBBC throughput drops from 5700 TPS to 1900 TPS due to having t less proposals being accepted (the proposals sent by Byzantine proposers are invalid) and to the increase in latency. The latency increases due to the extra rounds needed to be executed by the binary consensus to terminate with 0. The throughput of HBBFT drops from 350 to 256 TPS due to fewer proposals but the latency decreases because with less proposals erasure codes require less computation.

Byz2 is designed against the verified reliable broadcast of §V-B, to delay the delivery of the message to t of the correct proposers, and increasing the bandwidth used. HBBFT avoids this problem by using erasure codes, but has a higher bandwidth usage in the correct case. Figure 11 shows its impact on bandwidth usage and latency for RBBC and HBBFT with n proposers and proposal sizes of 100. The bandwidth usage of RBBC increases from 538 MB to 2622 MB per multivalued consensus instance compared to HBBFT, which uses 3600 MB in all cases. Furthermore, the latency of RBBC increases from 920 ms to 2300 ms.

Regarding corruptions, as it is impossible to solve consensus when $t \geq n/3$ [60], one must either replace the permissioned nodes [87] before $n/3$ collude, or implement an eventually consistent alternative of the service [78].

VIII. CONCLUSION

Blockchains tend to adopt an open permissioned model where a subset of the nodes with some permissions (e.g., PoS) can decide upon the next block. RBBC is the first of these that does not need synchrony to scale to hundreds of geo-distributed permissioned nodes. To this end, it solves the Set Byzantine Consensus problem, adopts a leaderless design that offers censorship-resistance and introduces sharded verification. World-wide experiments demonstrate that it triples the performance of its closest competitor.

Acknowledgments

We wish to thank our shepherd, Ralf Kuesters, our PC contact person, Andrew Miller, the anonymous reviewers as well as Alysson Bessani, Adi Seredinschi, Manuel Vidigueira for their comments. This research is supported under Australian Research Council Future Fellowship funding scheme (project number 180100496) entitled “The Red Belly Blockchain: A Scalable Blockchain for Internet of Things”.

References

[1] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. Revisiting fast practical Byzantine fault tolerance. Technical Report 1712.01367, arXiv, 2017.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A blockchain protocol based on reconfigurable Byzantine consensus. In *OPODIS*, pages 25:1–25:19, 2017.

[3] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous Byzantine agreement. In *ACM PODC*, pages 337–346, 2019.

[4] Musab A. Alturki, Jing Chen, Victor Luchangco, Brandon M. Moore, Karl Palmskog, Lucas Peña, and Grigore Rosu. Towards a verified model of the Algorand consensus protocol in Coq. In *International Workshops on Formal Methods (FM)*, pages 362–367, 2019.

[5] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In *EuroSys*, pages 30:1–30:15, 2018.

[6] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 BFT protocols. *ACM Trans. Comput. Syst.*, 32(4):1–45, January 2015.

[7] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: Redundant Byzantine fault tolerance. In *IEEE ICDCS*, pages 297–306, July 2013.

[8] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *ACM PODC*, pages 183–192, 1994.

[9] Nathalie Bertrand, Vincent Gramoli, Igor Konnov, Marijana Lazic, Pierre Tholoniat, and Josef Widder. Compositional verification of Byzantine consensus. Technical Report 03158911, HAL, 2021.

[10] Alyson Bessani, João Sousa, and Eduardo E. P. Alchieri. State machine replication for the masses with BFT-SMaRt. In *IEEE/IFIP DSN*, pages 355–362, 2014.

[11] Alysson Bessani, Eduardo Alchieri, João Sousa, André Oliveira, and Fernando Pedone. From Byzantine replication to blockchain: Consensus is only the beginning. In *IEEE/IFIP DSN*, pages 424–436. IEEE, 2020.

[12] Fatemeh Borran and André Schiper. A leader-free Byzantine consensus algorithm. In *ICDCN*, pages 67–78, 2010.

[13] Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

[14] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains, 2016. MS Thesis.

[15] Vitalik Buterin. On public and private blockchains, Aug. 2015. https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/.

[16] Christian Cachin, Daniel Collins, Tyler Crain, and Vincent Gramoli. Anonymity preserving Byzantine vector consensus. In *ESORICS*, pages 133–152. Springer, 2020.

[17] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541. Springer-Verlag, 2001.

[18] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[19] Benjamin Y. Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *ACM AFT*, pages 1–11, 2020.

[20] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. In *SOSP*, pages 189–204, 2007.

[21] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *USENIX NSDI*, pages 153–168, 2009.

[22] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. (Leader/randomization/signature)-free Byzantine consensus for consortium blockchains. Technical Report 1702.03068v2, arXiv, 2017.

[23] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: Efficient leaderless Byzantine consensus and its applications to blockchains. In *IEEE NCA*, pages 1–8, 2018.

[24] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Blockchain system and method, 2018. International Patent Application No. PCT/AU2018/050642.

[25] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Evaluating the red belly blockchain. Technical Report 1812.11747, arXiv, 2018.

[26] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains - (A position paper). In *Financial Cryptography*, pages 106–125, 2016.

[27] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE S&P*, pages 910–927, 2020.

[28] Bernardo David, Peter Gazi, and Aggelos Kiayias ands Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT*, pages 66–98, 2018.

[29] Assia Doudou and André Schiper. Muteness detectors for consensus with Byzantine processes. In *ACM PODC*, page 315, 1998.

[30] Sisi Duan, Michael K. Reiter, and Haibin Zhang. BEAT: asynchronous BFT made practical. In *ACM CCS*, pages 2028–2041, 2018.

[31] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[32] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. Impact of man-in-the-middle attacks on Ethereum. In *IEEE SRDS*, pages 11–20, 2018.

[33] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The attack of the clones against proof-of-authority. In *NDSS*, 2020.

[34] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *USENIX NSDI*, 2016.

[35] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Proc. Letters*, 14:183–186, 1982.

[36] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, pages 281–310, 2015.

[37] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68, 2017.

[38] Guy Golan-Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: A scalable and decentralized trust infrastructure. In *IEEE/IFIP DSN*, pages 568–580, 2019.

[39] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. FastFabric: Scaling hyperledger fabric to 20 000 transactions per second. *Int. J. Netw. Manag.*, 30(5), 2020.

[40] Vincent Gramoli. Blockchain scalability and its foundations in distributed systems. Accessed: 2021-03-01, https://www.coursera.org/learn/blockchain-scalability.

[41] Vincent Gramoli, Len Bass, Alan Fekete, and Daniel Sun. Rollup: Non-disruptive rolling upgrade with fast consensus-based dynamic reconfigurations. *IEEE Trans. on Parallel and Distributed Systems*, 27(9), 2015.

[42] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *ACM PODC*, pages 307–316, 2019.

[43] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous BFT protocols. In *ACM CCS*, pages 803–818, 2020.

[44] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. Technical Report 1805.04548, arXiv, May 2018.

[45] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *USENIX Security*, pages 129–144, 2015.

[46] Maurice Herlihy and Mark Moir. Enhancing accountability and trust in distributed ledgers. Technical Report 1606.07490, arXiv, 2016.

[47] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

[48] libhotstuff. Accessed: 2021-03-01 https://github.com/hot-stuff/libhotstuff.

[49] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX ATC*, pages 11–11, 2010.

[50] Kwon Jae. Tendermint: Consensus without mining, 2014. https://tendermint.com/docs/tendermint.pdf.

[51] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889, 2016. https://eprint.iacr.org/2016/889.

[52] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, pages 357–388, 2017.

[53] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *USENIX Security*, pages 279–296, 2016.

[54] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger. Technical Report 2017/405, Cryptology ePrint, 2017.

[55] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE S&P*, pages 583–598, 2018.

[56] Igor Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL*, pages 719–734. ACM, 2017.

[57] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *SOSP*, pages 45–58, 2007.

[58] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative Byzantine fault tolerance. *SIGOPS Oper. Syst. Rev.*, 41(6):45–58, October 2007.

[59] Leslie Lamport. Brief announcement: Leaderless Byzantine Paxos. In *DISC*, pages 141–142, 2011.

[60] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[61] Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. FairLedger: A fair blockchain protocol for financial institutions. In *OPODIS*, pages 4:1–4:17, 2019.

[62] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *Financial Cryptography*, pages 528–547, 2015.

[63] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. Fast and secure global payments with Stellar. In *SOSP*, pages 80–96, 2019.

[64] Giuliano Losa and Mike Dodds. On the formal verification of the Stellar consensus protocol. In *2nd Workshop on Formal Methods for Blockchains, FMBC@CAV 2020*, pages 9:1–9:9, 2020.

[65] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-MVBA: Optimal multi-valued validated asynchronous Byzantine agreement, revisited. In *ACM PODC*, pages 129–138, 2020.

[66] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *ACM CCS*, pages 17–30, 2016.

[67] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. Towards low latency state machine replication for uncivil wide-area networks. In *In Workshop on Hot Topics in System Dependability*, 2009.

[68] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *ACM CCS*, pages 31–42, 2016.

[69] Zarko Milosevic, Martin Biely, and André Schiper. Bounded delay in Byzantine-tolerant state machine replication. In *IEEE SRDS*, pages 61–70, Sep. 2013.

[70] Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *SOSP*, pages 358–372, 2013.

[71] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous Byzantine consensus with $t > n/3$ and $O(n^2)$ messages. In *ACM PODC*, pages 2–9, 2014.

[72] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary Byzantine consensus with $t > n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *J. ACM*, 62(4):31:1–31:21, September 2015.

[73] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. http://www.bitcoin.org.

[74] Christopher Natoli and Vincent Gramoli. The blockchain anomaly. In *IEEE NCA*, pages 310–317, 2016.

[75] Christopher Natoli and Vincent Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *IEEE/IFIP DSN*, Jun 2017.

[76] Nuno F. Neves, Miguel Correia, and Paulo Verissimo. Solving vector consensus with a wormhole. *IEEE Trans. Parallel Distrib. Syst.*, 16(12):1120–1131, December 2005.

[77] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[78] Alejandro Ranchal-Pedrosa and Vincent Gramoli. Blockchain is dead, long live blockchain! accountable state machine replication for longlasting blockchain. Technical Report 2007.10541, arXiv, 2020.

[79] Team Rocket. Snowflake to Avalanche: A novel metastable consensus protocol family for cryptocurrencies, 2018. Unpublished manuscript.

[80] Meni Rosenfeld. Analysis of hashrate-based double spending. Technical Report 1402.2009, arXiv, 2014.

[81] João Sousa, Alysson Bessani, and Marko Vukolić. A Byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *IEEE/IFIP DSN*, pages 51–58, 2018.

[82] Michael Spain, Sean Foley, and Vincent Gramoli. The impact of ethereum throughput and fees on the transaction latency during ICOs. In *Tokenomics*, volume 71, pages 9:1–9:15. Schloss Dagstuhl, 2019.

[83] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. Mir-BFT: High-throughput BFT for blockchains. Technical Report 1906.05552v3, arXiv, Jan. 2021.

[84] Pierre Sutra. On the correctness of egalitarian Paxos. *Inf. Proc. Letters*, 156, 2020.

[85] Pierre Tholoniat and Vincent Gramoli. Formally verifying blockchain Byzantine fault tolerance. In *The 6th Workshop on Formal Reasoning in Distributed Algorithms (FRIDA'19)*, 2019. Available at https://arxiv.org/pdf/1909.07453.pdf.

[86] Giuliana Veronese, Miguel Correia, Alysson Bessani, Lau Cheuk Lung, and Paulo Verissimo. Minimal Byzantine fault tolerance. Technical Report TR-2009-15, DI-FCUL, June 2009.

[87] Guillaume Vizier and Vincent Gramoli. ComChain: A blockchain with Byzantine fault-tolerant reconfiguration. *Concurr. Comput. Pract. Exp.*, 32(12), 2020.

[88] Gauthier Voron and Vincent Gramoli. Dispel: Byzantine SMR with distributed pipelining. Technical Report 1912.10367, arXiv, Dec. 2019.

[89] Marko Vukolic. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *IFIP WG 11.4 International Workshop on Open Problems in Network Security*, pages 112–125, 2015.

[90] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *USENIX NSDI*, pages 95–112, 2019.

[91] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.

[92] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *ACM PODC*, pages 347–356, 2019.

[93] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. RapidChain: Scaling blockchain via full sharding. In *ACM CCS*, pages 931–948, 2018.

[94] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without Byzantine oligarchy. In *OSDI'20*, pages 633–649, 2020.

Appendix

A. *Proofs of a Scalable Censorship-Resistant RSM*

In this section, we show that RBBC is a censorship-resistant replicated state machine (RSM) that totally orders correct transactions and we give bounds on its theoretical throughput.

1) Proof that RBBC implements an RSM: We show that our verified reliable broadcast ensures the properties of the reliable broadcast for valid values and discards invalid values, then we prove that our consensus protocol solves the Set Byzantine Consensus problem before showing that RBBC implements an RSM.

Theorem 4: The Verified Reliable Broadcast (lines 1–13) ensures the properties of the Reliable Broadcast for all valid values and does not deliver an invalid value at any correct proposer.

Proof. We proceed by showing the properties of reliable broadcast for a valid value: if a valid value is delivered then it was broadcast (validity), a correct proposer delivers at most one value from any given proposer (unicity), if a correct proposer broadcasts a valid value v then v is delivered at all correct proposers (termination1) and if a correct proposer delivers a valid value v, then all correct proposers deliver v (termination2). It is easy to ensure validity and unicity, so let us focus on termination1 and termination2. Consider that a valid value v is broadcast by some proposer p_i. There are two cases to consider, either p_i is correct or Byzantine.

1) **Proposer p_i is correct.** Proposer p_i broadcasts INIT to all proposers, hence each proposer broadcasts ECHO to all, and all correct proposers eventually receive ECHO messages with v from $n-t$ distinct correct proposers. These correct proposers, say Q, are thus ready to start verifying value v. As there are $t+1$ primary verifiers and t secondary verifiers, there are up to $2t+1$ verifiers, say Q', that will verify value v if not stopped at line 10. If $t+1$ verifiers broadcast READY messages with the same verification outcome verif, then we know that all correct proposers will then retransmit READY with this verif, which will guarantee that all correct proposers will receive $n-t$ messages $\langle \text{READY}, \mathit{verif}, h(v), p_i \rangle$ and will thus deliver v (line 13). It thus remains to show that $t+1$ verifiers will broadcast READY messages with the same verification outcome verif. Note that $|Q \cap Q'| \geq t+1$, which means that among the correct proposers Q, $t+1$ of them verify v and obtain the same verification outcome verif that they broadcast.
2) **Proposer p_i is Byzantine.** First, if proposer p_i broadcasts INIT successfully to all $n-t$ correct proposers, in which case, they all broadcast ECHO with value v to all proposers and the case is identical to case (1), where all correct proposers deliver v (line 13). In the case where proposer p_i does not broadcasts INIT to $t+1$ correct proposers because it broadcasts to less proposers, then not enough proposers will receive INIT for ECHO to be received by sufficiently many proposers at line 5 and v will not be delivered. Third, if proposer p_i broadcasts INIT to $t+1 \leq \ell < n-t$ proposers, then it depends on the behaviors of the other Byzantine proposers, if sufficiently many of them send ECHO messages to $t+1$ verifiers or if they help verifying correctly, then v will be delivered at all correct proposers, otherwise, it will not be delivered at any correct proposer.

To show that no invalid values can be delivered at any correct proposer, consider that v is invalid so there cannot be $t+1$ distinct verifiers whose verif is identifying v as valid. As a result, if line 9 is enabled with $t+1$ identical messages $\langle \text{READY}, \mathit{verif}, h(v), p_i \rangle$ from distinct proposers then we know that verif is necessarily identifying v as invalid and the precondition to deliver v at line 13 will not be satisfied. □

Lemma 1: In the Byzantine binary consensus (lines 31–48), if at the beginning of a round r, all correct proposers have the same estimate val, they never change their estimate value thereafter.

Proof. Let us assume that all correct processes (which are at least $n-t > t+1$) have the same estimate val when they start round r. Hence, they all bv-broadcast the same message $\text{EST}(val)$ either at line 15 or within the reliable broadcast at line 32. It follows from the properties of the reliable broadcast [13] and bv-broadcast [72] that each correct process p_i is such that $cvals_i = \{v\}$ at line 38, and consequently can broadcast only $\text{ECHO}(\{v\})$ at line 39. Considering any correct process p_i, it then follows from the predicate of line 42 (s_i contains only v), the predicate of line 43 (s_i is a singleton), and the assignment of line 45, that val_i keeps the value v. □

The next lemma states that if the value s in the same round of two correct replicas are singletons then they are identical.

Lemma 2: Let p_i and p_j be two correct proposers. In the Byzantine binary consensus (lines 31–48), if $s_i = \{v\}$ and $s_j = \{w\}$ in the same round, then $v = w$.

Proof. Let p_i be a correct proposer such that $s_i = \{v\}$. It follows from line 42 that p_i received the same message $\text{AUX}(\{v\})$ from $(n-t)$ different processes, i.e., from at least $(n-2t)$ different correct processes. As $n-2t \geq t+1$, this means that p_i received the message $\text{AUX}(\{v\})$ from a set Q_i including at least $(t+1)$ different correct proposers.

Let p_j be a correct proposer such that $s_j = \{w\}$. Hence, p_j received $\text{AUX}(\{w\})$ from a set Q_j of at least $(n-t)$ different proposers. As $(n-t)+(t+1) > n$, it follows that $Q_i \cap Q_j \neq \varnothing$. Let $p_k \in Q_i \cap Q_j$. As $p_k \in Q_i$, it is a correct proposer. Hence, at line 39, p_k sent the same AUX message to p_i and p_j, and we consequently have $v = w$. □

Lemma 3 (Binary Consensus Validity): In the Byzantine binary consensus (lines 31–48), the value decided by a correct proposer was proposed by a correct proposer.

Proof. Let us consider the round $r = 1$. Due to the property of the bv-broadcast executed at line 15 or piggybacked within the reliable broadcast at line 32, it follows that the sets $cvals_i$ contains only values proposed by correct proposers. Consequently, the correct proposers broadcast, at line 39 AUX messages containing sets with values proposed only by correct proposers. It then follows from the predicate of line 42 ($s_i \subseteq cvals_i$), and the reliable and bv-broadcast properties, that the set s_i of each correct proposer contains only values proposed by correct proposers. Hence, the assignment of val_i (be it at line 44 or 46) provides it with a value proposed by a correct proposer. The same reasoning applies to rounds $r = 2$, $r = 3$, etc., which concludes the proof of the lemma. □

Lemma 4 (Binary Consensus Agreement): In the Byzantine binary consensus (lines 31–48), no two correct replicas decide different values.

Proof. Let r be the first round during which a correct proposer decides, let p_i be a correct proposer that decides in round r (line 45), and let v be the value it decides. Hence, we have $s_i^r = \{v\}$ where $v = (r \bmod 2)$.

If another correct replica p_j decides during round r, we have $s_j^r = \{w\}$, and, due to Lemma 2, we have $w = v$. Hence, all correct proposers that decide in round r, decide v. Moreover, each correct proposer that decides in round r has previously assigned $v = (r \bmod 2)$ to its local estimate s_i.

Let p_j be a correct proposer that does not decide in round r. As $s_i^r = \{v\}$, and p_j does not decide in round r, it follows from Lemma 2 that we cannot have $s_j^r = \{1 - v\}$, and consequently $s_j^r = \{0, 1\}$. Hence, in round r, p_j executes line 44, where it assigns the value $(r \bmod 2) = v$ to its local estimate val_j.

It follows that all correct proposers start round $(r+1)$ with the same local estimate $v = r \bmod 2$. Due to Lemma 1, they keep this estimate value forever. Hence, no different value can be decided in a future round by a correct proposer that has not decided during round r, which concludes the proof of the lemma. □

Lemma 5: During the RBBC consensus (lines 14–23) execution, at least one binary consensus instance decides 1.

Proof. By Theorem 4, we know that all correct proposers eventually populate their proposal array with at least one common value. Due to the reduction, all correct proposers will thus have input 1 for the corresponding binary consensus instance. By the validity (Lemma 3) and termination [22] properties of the binary consensus, the decided value for this binary consensus instance has to be 1. □

Lemma 6: If a Byzantine binary consensus instance at index i decides 1, then the verified reliable broadcast (lines 1–13) at index i reliably delivers a value at a correct proposer.

Proof. A correct proposer does not propose 1 to a binary consensus instance at index i without reliably delivering a proposal at index i. The result follows from Theorem 4 and the validity of the binary consensus (Lemma 3). □

Theorem 5 (Set Byzantine Consensus): The RBBC consensus (lines 14–23) solves the Set Byzantine Consensus.

Proof. By Lemma 5, at one index, a binary consensus instance terminates with 1. By Lemma 4, we know that all correct proposers have set 1 to the same indices of their *bitmask*. For each of these indices k there is a proposal $props[k]$ that will be delivered at every correct proposer by Lemma 6. As a result, all correct proposers invoke function reconciliate with the same argument at line 23. By examination of the code at lines 24–29, all correct proposers thus put in their *superblock* the same subset of valid and non-conflicting transactions hence guaranteeing all properties of the SBC problem (§III). □

Theorem 1 (Replicated State Machine): In RBBC, all correct replicas observe the same sequence of committed transactions, which are all valid and non-conflicting.

Proof. In RBBC, all permissioned nodes run the consensus algorithm either because they receive messages from proposers or because they propose themselves (§V-A[❷]). Each node starts by running a single instance of this consensus algorithm for the block at index 1 (after the genesis block). A proposer can start a new consensus instance for a block at index $j > 1$ only after the consensus instance at index $j-1$ has terminated. As Theorem 5 shows that consensus guarantees agreement there is a single block decided per index of the blockchain at correct permissioned nodes. It results that blocks are totally ordered through their index number: whenever a block is decided, it is ordered after all previously decided blocks. Given that in each block the transactions do not conflict and are ordered through the same deterministic strategy employed by all correct permissioned nodes (lines 24–29), transactions are replicated by $t+1$ correct permissioned nodes in the same total order. By examination of the code, all committed transactions are valid and non-conflicting because RBBC discards the incorrectly signed transactions eagerly (line 13) and the non-provisioned and conflicting transactions lazily (lines 28). □

Theorem 2 (Scalable throughput): Let m be the number of transactions proposed to the Set Byzantine Consensus by each proposer and let $n = |P|$ be the number of proposers. Assume that proposals are all reliably delivered before the algorithm times out (line 17) but that all $\lceil \frac{n}{3} \rceil - 1$ Byzantine proposers do not propose any transaction. The Set Byzantine Consensus commits between $2m$ (as n tends to infinity) and $\Omega(n)$ distinct transactions.

Proof. Note that correct requesters must send their transaction to $t+1$ proposers to guarantee that it will be committed. As a result, the same transaction could appear in the proposal of different proposers.

- Consider the worst case scenario where all the duplicated transactions of all requesters are proposed to the same consensus instance. As a result there are at most $A = m \cdot (\lfloor \frac{2n}{3} \rfloor + 1)$ transactions and each transaction is duplicated $B = \lfloor \frac{n}{3} \rfloor + 1$ times. We thus end up having A/B transactions committed such that $\lim_{n \to \infty} A/B = 2m$.
- Consider the best case scenario where each requester transaction is sent to 1 correct proposer and the t Byzantine proposers. As the Byzantine proposers do not propose anything, there are no duplicates, and each of the $\lfloor \frac{2n}{3} \rfloor + 1$ correct proposers propose m distinct transactions, leading to $m \cdot \lfloor \frac{2n}{3} \rfloor + 1 = \Omega(n)$ transactions committed per consensus instance.

It follows that the minimum number of transactions that can be committed per consensus instance is $2m$ as n tends to infinity and the maximum number is $\Omega(n)$. □

2) *Proof of Censorship Resistance:* We show that RBBC is censorship-resistant where $2t < |P| \leq n$ in that every transaction sent by a correct requester to only $t+1$ proposers is eventually committed.

Lemma 7: Every transaction sent by a correct requester is eventually proposed by a correct proposer.

Proof. We know by (§V-A[❶]) that a correct requester sends its transaction to $t+1$ proposers. It follows that at least one correct proposer receives this transaction. By the *bounded-requesting-rate* assumption (§III), we know that this correct proposer will add it to its mempool and thus propose (line 14) it to the Set Byzantine Consensus. □

Theorem 3 (Censorship-Resistance): Every transaction sent by a correct requester is eventually included in a superblock. **Proof.** By Lemma 7, we know that the transaction tx of a correct requester is proposed by a correct proposer, say in the k^{th} proposal. If during the current consensus instance this k^{th} proposal is included in the superblock, then tx is committed immediately yielding the result. So let us consider that the k^{th} proposal is not included, this can happen if the corresponding binary consensus returned 0 because $bitmask[k] = 0$. As the proposer is correct, its proposal will be eventually rb-delivered by all $|P| - t$ correct proposers. By the *bounded-requesting-rate* assumption and because tx is not committed, these $|P| - t$ proposers record tx in their mempool with a birth date corresponding to their local clock.

At each multivalue consensus instance, one of the correct permissioned nodes proposes transactions that get included in the agreed upon superblock and thus committed. This is due to the pigeonhole principle applied to the $bitmask$ of $|P|$ indices among which $|P| - t > t$ correspond to the identifiers of correct nodes and $|P| - t$ indices map to 1s (line 17). There is no guarantee that in a given multivalue consensus instance any correct proposer proposes tx, but as transactions are proposed in decreasing order of their age, we know that it will be proposed among the $(|P| \cdot T + 1)$ next transactions were T is the maximum number of transactions in the mempool of each of the up to $|P|$ correct proposers when it inserts tx in its mempool. Due to the same pigeonhole argument as before and because of the sequential-transaction-requests assumption (§III), transaction tx will be committed. □

B. Disclosure

Some of the techniques proposed in this paper appear in a patent application [24] and are taught in a massive open online course [40].