THE DISTRIBUTED COMPUTING COLUMN

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

In this issue of the distributed computing column, we return to the topic of blockchains. The column provides an overview of the Red Belly Blockchain, an innovative new blockchain developed in Australia (and named after the locally prevalent red-bellied black snake). Red Belly Blockchain has served as a testbed for several cutting edge approaches to designing a blockchain, including ideas from distributed computing, game theory, and formal verification. Some examples described in this paper include:

- Red Belly blockchain is (provably) *accountable*, providing a cryptographically secure "proof-of-fraud" when users act maliciously and disrupt the system.
- The blockchain contains several scalability innovations. For example, it takes an interesting approach to sharding (and "superblocks"), using a leaderless ("democratic") approach to dynamically create new shards as needed.
- The consensus component of the Red Belly blockchain has been formally verified with model checking tools, providing stronger guarantees of correctness.

The article also discusses new optimizations in handling smart contracts, challenges in governence, and trade-offs between mutability, accountability, and privacy. The article provides a window into how recent academic research translates into a large-scale, real-world blockchain design. Overall, it provides interesting insights in the challenges of integrating new ideas into blockchain design. Enjoy this new distributed computing column!

Redbelly Blockchain: a Combination of Recent Advances*

Redbelly Network

Abstract

Redbelly Blockchain builds upon recent scientific advances in the context of distributed computing, game theory and formal verification to apply blockchains to the real world. In this paper, we present how Redbelly Blockchain combines these results to remedy vulnerabilities that affect modern blockchains. In particular, Redbelly Blockchain offers accountability by generating a *Proof-of-Fraud*, an undeniable proof of misbehavior, automatically. The architecture of Redbelly Blockchain is decoupled into a consensus component and a language virtual machine component, called SEVM, to achieve resilience optimality against failures and attacks. For greater security, its consensus component does not assume pure synchrony, is formally verified with model checking and solves the consensus problem deterministically. To run a large ecosystem of dApps efficiently, the SEVM supports Solidity bytecode without the unnecessary redundant validations of traditional designs. Redbelly Blockchain is dynamic as it features a built-in reconfiguration smart contract triggered by a representative governance. Finally, it is designed for mobile devices to interact securely without downloading the blockchain.

1 Distributed Ledger Architecture

The blockchain service is provided by blockchain *nodes*, or machines, offering consensus execution and storage service. Redbelly Blockchain is *open* in that any node has the possibility of providing the service, however, Redbelly Blockchain also aims at not wasting node resources by offering too many redundant services. Hence, instead of incentivising all nodes to execute the same tasks, Redbelly Blockchain allows different nodes to offer different services at different times. To eventually provide the service, nodes must first authenticate themselves, just like nodes must first offer a proof-of-work in classic blockchains [27] to produce valid blocks. One can thus compare the absence of proof-of-fraud in Redbelly Blockchain to the creation of proof-of-work, but without the carbon footprint associated with resolving cryptopuzzles.

More precisely, the architecture of Redbelly Blockchain is decoupled in different components to achieve resilience optimality. It is known that consensus cannot be solved in the general model, when nodes communicate over the Internet and in the presence of arbitrary (byzantine) failures, without $n > 3f$ nodes [26], where f is the number of byzantine nodes. Interestingly, however, this threshold does not apply to provide secure data retrieval: to achieve data integrity and tamper-proofness one simply needs $n > 2f$: despite f byzantine nodes, one is guaranteed to identify the correct copy as the only copy received from $f + 1$ distinct nodes. Hence, decoupling storage/SEVM nodes from consensus nodes allows to better tune the resource provisioning necessary to achieve security.

Figure 1 presents the architecture of Redbelly Blockchain. The SEVM component is web3 compliant, executes *decentralised applications (dApps)* and reliably stores the blockchain. The web3 compliance comes from its `web3.js` server that accepts valid requests to transfer assets, upload or

*Contact author: Vincent Gramoli `vincent.gramoli@redbelly.network`

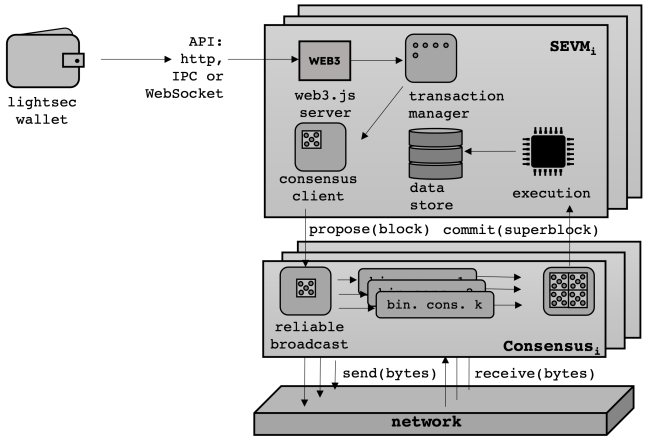


Figure 1: Redbelly Blockchain architecture and the life cycle of a transaction

execute a smart contract. Redbelly Blockchain supports smart contracts written in Solidity bytecode to facilitate the integration of the most used dApps. The transaction manager batches these transactions into blocks that are proposed to the consensus service. The consensus groups as many proposed blocks as possible into a committed *superblock* before the SEVM validates its requests, executes them and reliably stores the results.

The consensus component (§2) features the Democratic Byzantine Fault Tolerant (DBFT) consensus algorithm [11] that solves consensus deterministically without the need to assume pure communication *synchrony* or that there exists a known upper bound on the message delays [14]. (Instead, we assume that this bound is unknown, a property referred to as partial synchrony.) Note that this is appealing to cope with network attacks that could otherwise lead to double spending [28, 16]. To avoid network bottlenecks, DBFT starts with a leaderless all-to-all reliable broadcast where each consensus node shares its proposal with the rest of the system (§2.2). Then DBFT spawns as many binary consensus instances as proposals to decide whether each of these proposals gets included in the decided superblock. As the binary consensus is probably the most elaborate and sensitive part of Redbelly Blockchain it has been formally verified with model checking (§2.1). The decided superblock is then sent back to the SEVM for execution and storage.

Redbelly Blockchain supports a large ecosystem of dApps by offering the *Scalable EVM (SEVM)* (§5). Like for the Ethereum Virtual Machine (EVM), the SEVM allows Redbelly Blockchain to run like a state machine replication whose commands are expressed in a Turing complete programming language. In contrast with Ethereum [36] though, Redbelly Blockchain leverages this capability by utilizing smart contract output as an input to itself. Since the smart contract execution is deterministic and agreed upon by all nodes, it allows Redbelly Blockchain to upgrade itself without hard-forking (§6.1), to change its governance and mitigate the risks of bribery (§6) and for a subset of nodes to spawn their own shard or close it on-demand at runtime (§7.2).

Last but not least, Redbelly Blockchain interfaces the real-world (§8). This is why it comes with a lightweight secure wallet, called *lightsec* (§8.1) and interacts with oracles (§8.2). In particular, the lightsec wallet differs from classic wallets in that it is both (i) lightweight as it can run in handheld devices without having to download a blockchain history, and (ii) secure as it can retrieve correct information, hence its name. The transaction fees are maintained low thanks to the high capacity throughput of Redbelly Blockchain. The oracle offers the necessary identification mechanism to let real users govern Redbelly Blockchain and inform the blockchain about the success of traditional payments.

1.1 Blockchain Abstract Representation

The distributed architecture described above implements a blockchain, a simple *linked list* abstraction chaining blocks, one block to another, each containing a sequence of cryptographically signed transactions. We call it a linked list (and not a directed acyclic graph) as it does not have forks: consensus upon a block at a given index is reached before this block gets appended. We say that a transaction is *final* as soon as it is included in a block of Redbelly Blockchain.

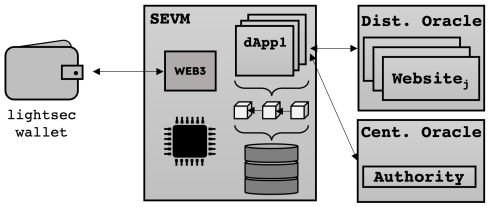


Figure 2: Redbelly Blockchain interactions with oracles

Figure 2 depicts the blockchain abstraction as maintained in the reliable storage of a single SEVM node. Thanks to the consensus protocol depicted in the distributed architecture (Figure 1), among two chains of blocks maintained locally by two honest SEVM nodes, either one is guaranteed to be the prefix of the other or the two are guaranteed to be identical. This property will be listed as a requirement of the blockchain problem (Def. 3). As all SEVM nodes end up having the same blocks anyway, we simply focus here on a single SEVM node, its interaction with oracles and the wallet for simplicity. The dApps are stored in the blockchain (either because these are built-in dApps or because some client uploaded them) and their functions can be invoked remotely by the clients. The dApps can gather real-world information from distributed (Dist.) or centralized (Cent.) oracles as we will discuss in §8.2.

2 Scalable and Verified Consensus

The byzantine consensus problem [26] is for a set of nodes to agree on a unique value despite arbitrary behaving nodes, called *byzantine*. This problem must be solved to design a secure blockchain that guarantees the uniqueness of the block to be appended at its next available index [1].

Redbelly Blockchain features the Democratic Byzantine Fault Tolerance (DBFT) consensus algorithm [11]. By contrast with most consensus algorithms, it is democratic in that it does not elect a leader node that tries to impose its value to the rest of the nodes, which would otherwise limit scalability as described in §2.2. In addition, this consensus algorithm is probably the first blockchain consensus algorithm to be formally verified with model checking [4] as we explain in §2.1, which reduces drastically the risks of human errors, that are otherwise common [33]. Finally, it does not assume that the bound on the delay of messages is known (it only assumes partial synchrony but not synchrony [14]) and it is resilient optimal as it tolerates $f < n/3$ arbitrary (or byzantine) failures.

The consensus problem is generally defined as the problem for *honest* (non-byzantine) nodes of ensuring the conjunction of three properties despite the presence of byzantine nodes.

Definition 1 (Byzantine Consensus)**.** *Assuming that each honest (non-byzantine) node proposes a value, the* Byzantine Consensus *(BC) problem is for each of them to decide on a value in such a way that all the following properties are satisfied:*

- *BC-Termination: every honest node eventually decides a value;*
- *BC-Agreement: no two honest nodes decide on different values;*

- *BC-Validity: the value decided is one of the proposed values.*

To understand the importance of solving consensus to design a blockchain, consider that the blockchain is in its initial state consisting of a genesis block, at index 0. Let an attacker, say Mallory, issue two conflicting transactions, tx_1 where Mallory transfers all her coins to Alice and tx_2 where Mallory transfers all her coins to Bob. Let us consider what can happen if the BC-Agreement property is violated in that distinct nodes of the blockchain believe they can append distinct blocks, one containing tx_1 and the other containing tx_2, at index 1 of the blockchain. This can lead to a *double spending*, where the same coins are spent twice. By contrast, when consensus is reached then all nodes agree on a unique block to be appended and it is simple for each individual node to go through this block and executes the transactions it contains one-by-one unless a transaction conflicts (e.g., lack sufficient funds to execute).

Algorithm 1 Binary consensus algorithm at replica p_i

```
1:  bin-propose(val):                                   ▷ binary consensus at p_i with val ∈ {0, 1}
2:    loop:                                             ▷ loop that starts with round r = 1
3:      (bv-broadcast(EST, r, val) → cvals)             ▷ reliable bcast that is ommitted in some optimization
4:      start-timer(r)                                  ▷ timeout increases with rounds
5:      if i = r mod n then                             ▷ coordinator rebroadcasts
6:        wait until (cvals = {w})                      ▷ cvals stores delivered values
7:        broadcast(COORD, r, w) → c                    ▷ coordinator broadcasts
8:      wait until (cvals ≠ ∅ ∧ timer expired)          ▷ wait enough time
9:      if c ∈ cvals then e ← {c} else e ← cvals        ▷ prioritize coord value
10:     broadcast(AUX, r, e) → bvals                    ▷ broadcast these values
11:     wait until ∃s ⊆ bvals where the two following conditions hold:   ▷ wait to deliver more values until
12:       • s contains contents received from at least n − t distinct nodes   ▷ sufficiently many copies are delivered
13:       • ∀v ∈ s, v ∈ cvals                           ▷ and every value in s is in cvals
14:     if s = {v} then                                 ▷ if there is only one value in s
15:       val ← v                                       ▷ adopt this singleton value
16:       if v = (r mod 2) and not decided yet then decide(v)   ▷ decide only once
17:     else val ← (r mod 2)                            ▷ otherwise, adopt the current parity bit
18:     if decided in round r − 2 then exit()           ▷ help others in two last rounds
19:     r ← r + 1                                       ▷ increment the round number
```

2.1 Formal Verification

As the problem of consensus is particularly difficult to solve, and human errors are frequent [33], it is important to check mathematically the properties of a security system, a process we call formal verification. We formally verified the binary consensus at the heart of DBFT (Alg. 1) using a model checker [4]. This binary consensus algorithm is represented as the bin.cons.1, ..., bin.cons.k blocks on Figure 1 because there is one instance of this binary consensus algorithm to decide whether each proposed block should be included in the committed superblock. All honest participants pass their input value *val* to the loop of Alg. 1 where they exchange values and refine their estimate until they decide a value (line 16). Because verification of liveness properties consists of assuming fairness and showing that a property holds in every fair execution, we relax the partial synchrony assumption [14] of DBFT and replace it by a fairness assumption. Relaxing partial synchrony allows to simplify the pseudocode by ignoring timers (lines 4 and 8 of Alg. 1) and the weak coordinator steps (lines 5–7 of Alg. 1). The fairness assumption states that in any infinite sequence of iterations of the protocol loop, there exists one iteration where, at all honest nodes, a binary value broadcast (or bv-broadcast) instances deliver the same bit first. Note that this fairness could be violated if byzantine nodes were controlling the network, but no solutions to the consensus problem would exist in this case anyway [6].

Algorithm	Verification time
bv-broadcast	48.87 s
Naive consensus	>3 days
Consensus	22.54 s

Table 1: Although none of the properties of the naive blockchain consensus could be verified within a day of execution of the model checker, it takes about ~4 s to verify each property on the simplified automaton of the blockchain consensus. Overall it takes less than 70 seconds to verify both the binary value broadcast and the blockchain consensus protocols.

To formally verify the consensus algorithm we holistically verified that the model of the pseudocode verifies the properties of Def. 1, expressed in linear temporal logic (LTL), for any system size n and fault tolerance $f < n/3$ as detailed elsewhere [3]. To this end, we rely on the parameterized model checker, ByMC [25], convert the pseudocode of the consensus algorithm into a threshold automaton, and deployed it on a Message Passing Interface (MPI) cluster of 4 AMD Opteron 6276 16-core CPU with 64 cores at 2300 MHz and 64 GB of memory. As the naive threshold automaton (Naive consensus) is too large to be formally verified within 3 days of execution on our MPI cluster as indicated in Table 1, we first verified the properties of the binary value broadcast (bv-broadcast) primitive (line 3 of Alg. 1) using ByMC before simplifying the naive threshold automaton using the formally proven properties of the bv-broadcast primitive. Both the safety and liveness properties of the resulting threshold automata (bv-broadcast and Consensus) could be verified in 1 minute and 11 seconds (cf. Table 1).

2.2 Bypassing the Leader Bottleneck

The name Democratic BFT consensus algorithm stems from the fact that this algorithm does not need a leader that tries to impose its block to the rest of the system. Instead, during an execution of DBFT, every node can propose its own block to the consensus and the decided, so called, superblock (cf. §3.2) can include any of the proposed blocks. One advantage is that, in DBFT, there cannot be a misbehaving leader that prevents the convergence towards agreement. In particular, there can be as many different weak coordinators (line 5 of Alg. 1) as there are concurrent binary consensus instances, and even a single weak coordinator cannot prevent for sure the convergence of the binary consensus to which it participates. As opposed to the nodes of traditional blockchains that "compete" to append their own block, the nodes of Redbelly Blockchain "collaborate" to append a combined superblock.

Another advantage of this collaboration is the scalability induced by having every participant exchanging their proposals in parallel over a wide area network. Consider, as an example, that we want to append a new block (or superblock) of b bits with a blockchain system with a traditional leader-based consensus algorithm running on n nodes that have limited bandwidth resources. In particular, each node i is equipped with a download capacity of d_i and an upload capacity of u_i, both expressed in bits per unit of time. Without loss of generality, let the leader be node 1 with download and upload capacities d_1 and u_1, respectively. The time τ it will take the leader to send the block to all nodes (we consider that the leader sends to itself for simplicity as it does not alter our conclusion) is the maximum between these two:

- The time for the leader to upload $n \cdot b$ bits, which is $\frac{n \cdot b}{u_1}$ units of time.
- The time to download b bits for the node that has the lowest download rate among all other nodes, which is $\frac{b}{min(d_i : 1 \leq i \leq n)}$ units of time.

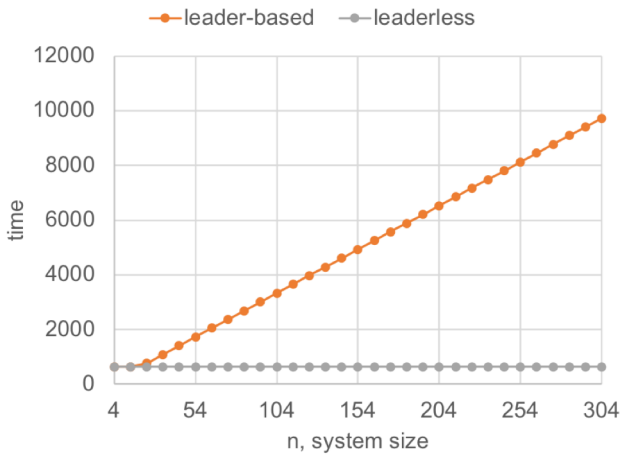


Figure 3: The time it takes for propagating a block in a leader-based consensus algorithm typically increases with the number n of nodes whereas the time to propagate a block in a leaderless consensus algorithm can be independent of n

Figure 3 depicts the time to propagate a block in both our simplistic leaderless and leader-based consensus algorithms as a function of n. The time needed for the leader to propagate the block to all nodes is $\tau = max\left(\frac{n \cdot b}{u_1}, \frac{b}{min(d_i : 1 \leq i \leq n)}\right)$ and we can conclude that this time is $\Omega(n)$ units of time. As n increases, we can see that the time it takes for the leader-based consensus algorithm to propagate the leader block increases with the number of nodes in the system. By contrast, the time it takes if the algorithm is leaderless to propagate a block of b bits to all the nodes is the maximum between the time for the node with the lowest upload rate to upload b/n bits, which is $\frac{b/n}{min(u_i : 1 \leq i \leq n)}$, and the time for the node with the lowest download rate to download b bits, which is $\frac{b}{min(d_i : 1 \leq i \leq n)}$. Provided that the difference in download rates among n nodes is independent of n, the maximum is thus a constant independent of n because each node simply needs to propagate b/n bits. This explains in part why DBFT helps Redbelly Blockchain scale [12].

3 Block Structure

As mentioned previously, the structure of the Redbelly Blockchain is a linked list, hence this can be viewed as a "non-forkable" chain of superblocks.

3.1 A non-forkable chain

As opposed to classic blockchains, Redbelly Blockchain does not fork to mitigate the risks of double spending. The main distinction with classic blockchains is that Redbelly Blockchain solves consensus first, before appending the unique agreed upon superblock as we already explained (§1.1). This guarantees a total order on the blocks that allows anyone to retrieve the current state of accounts by consulting the latest block: we do not talk about "confirmation" as a transaction is either pending or committed. This is in contrast with classic blockchains where an appended block must be sufficiently confirmed (or followed by sufficiently many blocks) for the probability of its transactions aborting to be sufficiently low.

Even during network attacks that can delay the message propagation [15], in common executions, two conflicting transactions cannot be inserted into two branches. We refer to these "common executions", as we will detail in §4, as executions in which an overwhelming number $f \geq n/3$ of

faults do not occur at the same time. In the scenario where conflicting transactions are included in the same block, then the first of these transactions will get executed while the second will not, as the conflict will be detected locally by the SEVM node. This tolerance to network partition can be seen as the result of favoring consistency over availability given that not both consistency and availability can be achieved in this case [21]. The key motivations for favoring consistency is for Redbelly Blockchain to support secure applications: no transactions that appeared committed can later be aborted.

In §4, we will explain how to cope with an overwhelming number of faults. In §7.2, we will explain how we can extend Redbelly Blockchain to spawn additional chains to complement the mainchain, however, none of these chains fork because they inherit the Redbelly Blockchain design.

3.2 Superblocks to scale throughput

Appending superblocks to the blockchain allows us to increase the performance (i.e., throughput) as the number of nodes running the consensus grows. This optimization originally proposed in [22], consists of resolving a different notion of consensus, called the Set Byzantine Consensus [12] as defined below.

Definition 2 (Set Byzantine Consensus)**.** *Assuming that each honest node proposes a set of transactions, 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:*

- *SBC-Termination: every honest node eventually decides a set of transactions;*
- *SBC-Agreement: no two honest nodes decide on different sets of transactions;*
- *SBC-Validity: a decided set of transactions is a non-conflicting set of valid transactions taken from the union of the proposed sets;*
- *SBC-Nontriviality: if all nodes are honest and propose a common valid non-conflicting set of transactions, then this set is the decided set.*

Thanks to this new definition, Redbelly Blockchain does not need to decide one block maximum in each iteration of the consensus (as classic blockchains do). Instead, Redbelly Blockchain decides a number of blocks that can grow linearly with the number of nodes in the system. This is key to the scalability of the consensus protocol. There are two important remarks regarding this definition. First, note that it refers to a decided set, although the transactions should be executed in the same order at every node. This can easily be ensured by executing transactions in the order of their hash or nonce. Second, the superblock optimization differs from batching more proposals at the leader, as batching could exacerbate the bottleneck effect of the leader (§2.2).

4 Accountability with PoF

Redbelly Blockchain applies *accountability* [23], the ability to make distributed participants responsible for their actions, to the partially synchronous setting [9]. To this end, it generates undeniable proofs-of-fraud (PoFs) as indicated in §4.2. By requiring participants to deposit assets prior to participating, we can exploit proofs-of-fraud to slash malicious players and compensate the victims, in the unlucky case of an overwhelming majority of participants as we will explain in §4.1.

PoF can be compared to PoW or PoS to cope with Sybil attacks: a block is considered valid in Redbelly Blockchain only if it was produced by a participant that would leave some undeniable PoF in case it tries to attack the system. This is why a valid block is one that is produced by a user who

provided the necessary information required by Redbelly Blockchain. This information is used to identify the user uniquely, which prevents a malicious attacker from impersonating other users to conduct a Sybil attack.

4.1 Strengthening fault tolerance

Accountability allows us to reward only the good behaviors that contribute to the system. Such good behaviors include staking (providing liquidity to the system for a period of time), using storage resources to keep track of the blockchain history, or exploiting network and CPU resources in order to contribute in reaching a consensus. Redbelly Blockchain will require consensus participants to deposit some stake before they start executing the consensus. They will gain a reward based on their contribution to the consensus. Our reward is equally divided among the consensus participants for their contributions but consensus participants will change over time as we will explain in §6.1. If a consensus participant misbehaves, then it will not receive its reward and will be excluded from the set of consensus participants, preventing it from being rewarded in the future.

Accountability is particularly effective in "uncommon" situations, where a coalition of malicious users is of size f sufficiently large to lead honest consensus nodes to a disagreement. As consensus is impossible in the general setting when $n \leq 3f$ where n is the number of consensus nodes [26], we know that this can happen. Fortunately, even when $f \geq n/3$ and as long as $f < 2n/3$ and less than $n/3$ nodes are inactive forever, Redbelly Blockchain can recover. When the disagreement occurs we can upper-bound the number a of branches given the number f of malicious participants, as was shown in [29]. A recent work has even demonstrated that not more than $2n/3$ honest participants is necessary to solve consensus when considering that some participants are rational [30]. For this reason, we can also lower-bound a deposit that consensus nodes need to escrow in order to reimburse any fooled users.

In particular, we require each consensus node to deposit some amount d during the time it participates to the consensus protocol. This deposit amount depends on various parameters. In particular, our recent result [29] expresses the ratio b of the deposit over the value G of funds that is being stolen. Consider a colluding majority of size $n/3 \leq f < 5n/9$, a probability of attack success $\rho = 0.5$, deposits held for $m = 10$ blocks and $G = \$1M$ manipulated funds. As $f < 5n/9$, we have $a \leq 3$ branches, thus $b = 1/500$ is sufficient. As $D = G/500 = 2,000$, for $n = 100$, each node needs to deposit $3bG/n = \$60$ for Redbelly Blockchain to recover.

4.2 Proof-of-Fraud

Our solution to the accountability problem is to build undeniable proofs-of-fraud at runtime. More specifically, our implementation enforces all nodes to sign key messages: a honest node will simply ignore key messages that are not properly signed by their senders. These 'key' messages are those that can influence the decision of other participants during the execution of any binary consensus protocol (Alg. 1) or the reliable broadcast (§1) that precedes these binary consensus executions [29].

In particular, it was shown that the only way for honest nodes to disagree while executing DBFT is for a coalition of malicious nodes to hack one of the broadcast primitives so as to equivocate, by sending different messages to different honest nodes [9]. Provided that these messages are signed, upon reception of these messages, honest nodes simply have to cross-check their received messages to detect a misbehavior. An undeniable proof-of-fraud is thus built by a honest node using the concatenation of two equivocating messages from the same sender. These proofs-of-fraud are rapidly communicated to other honest nodes to stop rewarding malicious nodes.

Note that the original accountable consensus technique, called Polygraph [9], is an extension of our consensus algorithm DBFT [11], which already offered scalable results in geo-distributed experiments [29]. Since then, a more generic transformation has been proposed [10]. It only requires

an additional round of communication involving an additive $O(n^2)$ communication complexity when threshold signatures can be used.

5 Language Virtual Machine

In this section, we present the Scalable EVM (SEVM), an optimized way of validating transactions and executing smart contracts.

5.1 Smart contracts

Redbelly Blockchain is compatible with a large ecosystem of DApps by offering the possibility to execute the same bytecode supported by the Ethereum Virtual Machine (EVM) [36]. The motivation for not developing a new DApp language is simple: the shortage in resources and the high demand of programmers skilled in blockchain raised the cost of programming decentalized finance, which makes it difficult to grow a new ecosystem of DApps.

5.2 Validation reduction

We built upon the EVM to develop what is called the *SEVM*, standing for the Scalable EVM, as detailed in [32]. It inherits the gas mechanism of Ethereum, mitigating denial-of-service attacks, but differs from the EVM to achieve 10,000 TPS on 100 nodes with 33 failures tolerated. With our consensus protocol (§2), the overhead is no longer the consensus but the EVM execution when trying to execute smart contract functions. The key aspect to reduce existing smart-contract blockchain overheads is thus to reduce the unnecessary validation overhead common to existing blockchains (e.g., Ethereum, Libra/Diem) where all validators validate each transaction twice (upon transaction reception and upon block reception), hence validating globally each transaction $2n$ times, where n is the number of SEVM nodes. Redbelly Blockchain simply needs to validate each transaction $n + 1/n$ times, which tends to halving the validation overhead as the system size grows towards infinity, $n \to \infty$. The security is not hampered because all nodes, upon reception of the decided block, validate every transaction, anyway.

Go Ethereum, or `geth` for short, is probably the mostly deployed EVM implementation. In order to check that a request (or transaction) is valid, all of the `geth` *servers* (i.e., miners) must validate each transaction eagerly and lazily, hence we distinguish the two following validations:

- **Eager validation:** This validation occurs upon reception of a new client transaction to check various parmeters of this transaction (gas, balance, signature, size). If the transaction is valid, it is propagated to other servers.

- **Lazy validation:** This validation occurs before transactions are executed in a decided block and simply checks the nonce and the gas. The lazy validation is thus less time consuming in `geth` than the eager validation.

Note that this is an overconservative strategy because each to-be-executed transaction of `geth` is validated twice by each validator/miner. In particular, there is no need for all validators to validate all transactions twice. In Redbelly Blockchain, only a constant number of nodes execute the eager validation, but without re-propagating the transaction to all servers. If the transaction is decided by the consensus algorithm, anyway, all servers will execute the lazy validation before executing the transaction. Note that not forwarding the request still ensures the best effort property of classic blockchains, namely that a transaction needs to be received by a honest node to be eventually

committed. The advantage is that it reduces the number of validations per nodes from $V^n_{eth} = 2$ in Ethereum to $V^n_{rbb} = 1 + \frac{k}{n}$ in Redbelly Blockchain. At large scale, when $n \to \infty$, we thus obtain:

$$\begin{cases} \lim_{n\to\infty} V^n_{rbb} &= \lim_{n\to\infty} \left(1 + \frac{k}{n}\right) = 1, \\ \lim_{n\to\infty} V^n_{eth} &= 2. \end{cases}$$

Hence, the number of validations per node in Redbelly Blockchain tends to become half the number of validations per node in existing blockchains (e.g., Ethereum, Libra/Diem [18]) as the system enlarges.

6 Governance

In this section, we explain the internals of the Redbelly Blockchain governance that decides upon variations in the protocol, the incentives or the governance membership. The governance is handled by the consensus nodes and the SEVM nodes. Note that each of these nodes may reside on different machines and be administered by different entities.

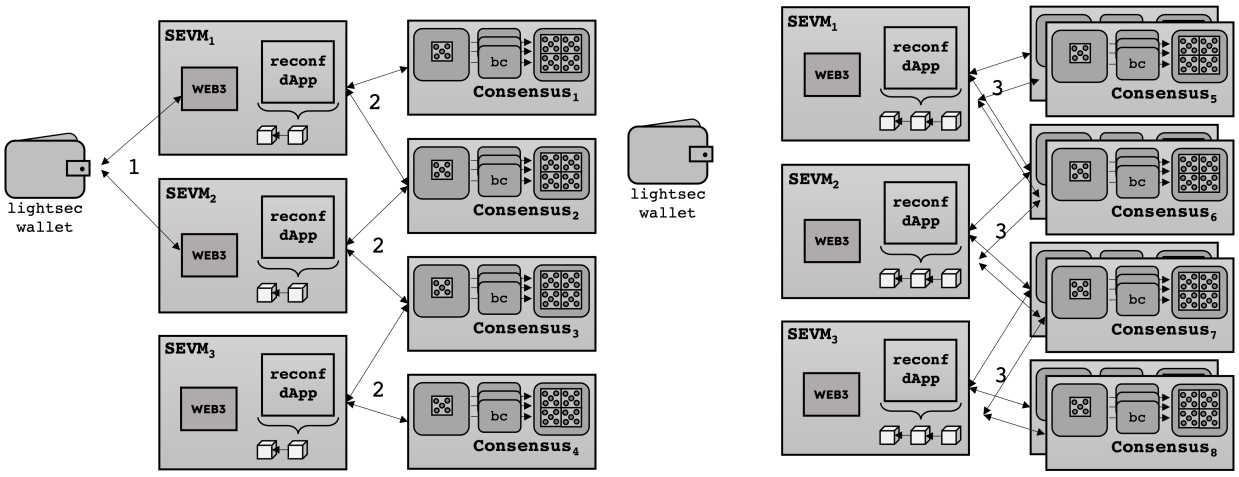
6.1 Reconfiguration to mitigate bribery

The nodes that govern are called *governors*. They must change from time to time to reduce the risks of bribery attacks, under the assumption of a slowly-adaptive adversary (so that it takes more time to bribe more nodes). Just like in other blockchains, nodes are incentivized to become governors by obtaining a reward for offering the blockchain service (e.g., execution, consensus, storage). Before governing, a node must first express its interest in governing and satisfy a series of requirements (personal information, resource allocation). If these requirements are met, the node becomes a *candidate*.

Redbelly Blockchain features a novel smart contract based reconfiguration process to change the governance every k blocks. The difficulty to reconfigure distributed systems is that all nodes must agree on the new configuration, a coordination process that often requires multiple consecutive consensus executions: one to decide to add new nodes to the system, another to discard the old nodes. Thanks to Redbelly Blockchain each of these consensus are reached simply through a *built-in smart contract* function invocation. As each execution of these invocations is deterministic and replicated, we are guaranteed that all honest nodes will be informed of this reconfiguration. This type of smart contracts is called "built-in", because they are already part of Redbelly Blockchain at the time it is started: all users of Redbelly Blockchain can thus observe the power delegated to the governance at the time they start using Redbelly Blockchain.

Figure 4 depicts the smart contract based reconfiguration process where SEVM nodes maintain a copy of the reconfiguration smart contract. For the sake of simplicity, it illustrates how to replace consensus nodes, but this reconfiguration can be used to replace SEVM nodes as well as changing the current version of the software. On the left hand side (Fig.4(a)), the client sends a request that invokes a function of the reconfiguration smart contract. This request is validated and passed onto the current consensus nodes ($consensus_1$, $consensus_2$, $consensus_3$ and $consensus_4$). The consensus nodes agree to encapsulate this request in the next superblock and pass this superblock to the SEVM for storage and execution. Upon execution (Fig.4(b)), the reconfiguration smart contract function replaces the current consensus nodes by new ones ($consensus_5$, $consensus_6$, $consensus_7$ and $consensus_8$). Hence the governors responsible of running the consensus have been replaced.

The software upgrade also relies on a built-in smart contract function, however, it takes, as arguments, the current version number, the new version code (e.g., in the form of a URL and its RSA256 hash representation) and the index *idx* of the chain at which it should start being used.



(a) The client sends a function invocation that gets agreed upon by the consensus nodes

(b) The function is executed on each SEVM node, hence replacing the current consensus nodes by new ones

Figure 4: Smart contract based reconfiguration

When $2n/3 + 1$ of the governors have invoked the function provided that this happens before the chain reaches index *idx*, then the new version is being downloaded, its hash is checked and in case of success, the version is being used when the chain reaches *idx*. (No other version besides the current one can be used until index *idx* is reached.)

6.2 Elections with non-dictatorship

In order to avoid that an attacker acts as a dictator by imposing its chosen governance to the rest of the system, we build upon results from the social choice theory. One way of selecting new governors, is to let existing governors proportionally elect the next set of governors. Black [5] was the first to define proportionality or that elected members represent "all shades of political opinion" of a society.

Dummett [13] introduced *fully proportional representation* to account for ordinal ballots, containing multiple preferences: given a set of n voters aiming at electing a committee of k governors, if there exist $0 < \ell \leq k$ and a group of $\ell \cdot q_H$ voters who all rank the same ℓ candidates at the top of their preference orders, then these ℓ candidates are all elected. However, it builds upon Hare's quota q_H, which is vulnerable to strategic voting, whereby a majority of voters can elect a minority of seats [24]. This problem was solved with the introduction of Droop's quota q_D as the smallest quota such that no more candidates can be elected than there are seats to fill [34].

Woodall [37] replaces Hare's quota with Droop's quota $q = \lfloor \frac{n}{k+1} \rfloor$ and defines the *Droop proportionality criterion* as a variant of the fully proportional representation property: if for some whole numbers j and s satisfying $0 < j \leq s$, more than $j \cdot q_D$ of voters put the same s candidates (not necessarily in the same order) at the top of their preference list, then at least j of those s candidates should be elected.

This desirable "proportionality" property can be achieved using the *Single Transferable Vote (STV)* algorithm with Hare's quota $q_H = \frac{n}{k}$. In STV, candidates are added one by one to the winning committee and removed from the ballots if they obtain a quota q of votes. STV is used to elect the Australian senate and is known to ensure fully proportional representation. Unfortunately, this protocol is synchronous [14] in that its quotas generally rely on the number of votes n received within a maximum voting period. If some of these n voters are byzantine and do not respond, then the protocol could not terminate without synchrony.

As one cannot predict the time it will take to deliver any message without synchrony, one cannot distinguish a slow voter from a byzantine one. Considering n as the number of governors or potential

voters among which up to f can be bribed or byzantine, our protocol can only wait for at most $n-f$ votes to progress without assuming synchrony. Waiting for $n-f$ votes prevents us from guaranteeing that the aforementioned quotas can be reached. We thus define a new quota called the *byzantine quota* $q_B = \lfloor \frac{n-f}{k+1} \rfloor$ such that $f < n/3$ and reduce, to $n-f$, the number of needed votes to start the election. Of course, up to f of these $n-f$ ballots may be cast by byzantine nodes, however, Redbelly Blockchain guarantees that no adversary controlling up to f byzantine nodes can act as a dictator in always imposing its decision.

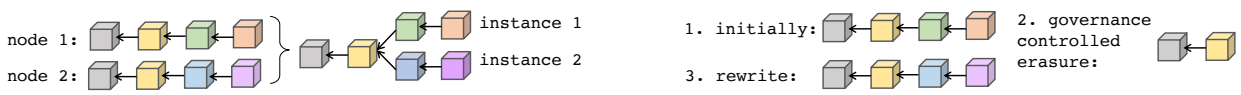
Based on q_B, we propose a smart contract election that expects $n-f$ votes to be cast to run a byzantine fault tolerant version of STV that satisfies proportionality and non-dictatorship without assuming synchrony.

7 Mutability, Auditability, Privacy

In this section, we discuss tradeoffs between auditability and privacy on the one hand, and mutability and immutability on the other hand (§7.1). We also present the two key techniques to offer privacy. First, Redbelly Blockchain offers the possibility to a subset of users to spawn a new shard as a dynamic variant (§7.2) of the Eth2 topology [19]: by invoking a *mainchain* built-in smart contract, some of the users can spawn a new *shardchain*. This allows users to perform transactions that are not directly visible from the mainchain or from other shards and without inducing a negative impact on performance. In addition, to enforce a stronger form of privacy, we will exploit the privatization of fungible and non-fungible tokens (§7.3).

7.1 Mutability with a Built-in Contract

On the one hand, immutability is a particularly appealing property for maintaining a distributed ledger. If transactions can be erased, then the data integrity is at risk. On the other hand, blockchain can be used for storing sensitive data whose immutability is questionable. The General Data Protection Regulation (GDPR) imposed in Europe since 2018 permits personal data to be rectified, withdrawn of permission, and erased. Similarly, the California Consumer Privacy Act (CCPA) and the United States Fair Credit Reporting Act (FCRA) enforces the possibility to remove objectionable data from the blockchain. Even mainstream blockchains are mutable: after the DAO hack that affected Ethereum, the history of transactions was replaced through a hard fork. Unfortunately, this created confusion and led to the use of two blockchains, Ethereum Classic and Ethereum. This is why, Redbelly Blockchain guarantees immutability by preventing any adversary from tampering with data but offers a built-in smart contract rewrite function to the governance.



(a) Rewriting in a classic blockchain creates new instances

(b) Rewriting in Redbelly Blockchain does not fork

Figure 5: To avoid forks while offering mutability, Redbelly Blockchain enforces that any rework of the history is controlled by the governance

Figure 5 compares a rewrite in a classic blockchain (Fig 5(a)) to a rewrite in Redbelly Blockchain (Fig. 5(b)). In classic blockchains, miners may legitimately decide to run the same software version they have been running before, whereas other users may follow the recommendation to upgrade their software. This typically creates multiple instances sharing the same genesis block but followed

by diverging transaction histories. The original instance is usually called "classic" so that the new version retains the original name of the blockchain, which can add to the user's confusion.

By contrast and as depicted in Figure 5(b), in Redbelly Blockchain, all users initially agree to run a reconfigurable blockchain, as we already explained in Section 6.1. This guarantees that all users are ready to upgrade to a new version if the governance majoritarily vote to do so. Typically, the governors cast their vote by passing a rewrite parameter to a built-in contract, a smart contract that was already present when the blockchain was launched, as defined in §6.1. This rewrite parameter is a pair $\langle state, start\text{-}index \rangle$ where the *state* is the hash of the global state of the blockchain, also called "state root" and the *start-index* is the index of the first block of the blockchain suffix to erase. If a quorum of $2f + 1$ governors vote for the same *start-index* within the time it takes to create k blocks (meaning that all their *state* parameters belong to at most k consecutive blocks, then the erasure of the suffix takes place at each honest SEVM node. From then on, all blocks get appended in place of the erased suffix.

7.2 Dynamic Sharding

Our blockchain design allows participants to operate individually in a dedicated shard or *shardchain* without presenting all their transactions to the rest of the system. This benefits privacy and performance by reducing the congestion on the default chain called the *mainchain*. It can also allow users to control the sovereignty of data by allocating the machines hosting these data in a specific jurisdiction. While this sharding may look similar to the beacon chain and the shard chains of Eth2 [19], it differs by being dynamic: one can adjust the number and size of shards at runtime as was recently proposed [31].

Redbelly Blockchain exploits smart contract outputs to configure itself at the lower level. Because our consensus protocol is both deterministic and formally verified (§2.1), it guarantees that all participants agree on a consistent total order on the smart contract upload, invocations and transactions issued to Redbelly Blockchain. As a result, and because smart contract functions are also deterministic, the execution of the sequences of commands at all honest participants results in the same outcome or state.

A built-in smart contract features a `spawnShard()` function that allows a set of participants to create a shard by providing the resources where to run the shard as well as depositing some assets that will be locked on the mainchain to be used in the shard. Similarly to the reconfiguration (§6.1), these participants will invoke the `spawnShard()` function with an index *idx* parameter and a number k of consecutive blocks, so that the shard will be spawned only if more than a single participant has invoked this function between the block at index *idx* and the block at index $idx + k$ of Redbelly Blockchain.

To close the shard, a `closeShard()` function will trigger the closing of a shard s when the blockchain depth reaches d only if a quorum of $\lfloor 2n/3 \rfloor + 1$ participants (or governors) of shard s have invoked this function within a certain range of indices *idx* to $idx + k < d$. To cope with non-termination due to windows of asynchrony, after failure to close a shard, the participants can simply retry with a longer offset k.

7.3 Stronger Forms of Privacy

Our solution will offer privacy of the transaction data (amount, asset, recipient) through the use of Zero-Knowledge Proof or Verifiable Secret Sharing. To this end, a privacy layer on top of Smart Redbelly that will consist of a shield smart contract, like the one used in Nightfall, will make the use of ERC20 (fungible) and ERC721 (non–fungible) tokens private. It allows the user to (i) create token commitments that are anonymous representations of the ECR20/721 tokens through the process of minting, (ii) to retrieve the token associated with a token commitment through the process of burning

and (iii) to transfer confidentially these commitments. This approach requires the user to create a proof off-chain and to store some private information needed to generate the proof locally. The user will then interact with the shield contract by sending their proof, which will then be verified. Upon successful verification, the token commitment will be stored in the commitment Merkle trie until it is spent, in which case it will be stored in a nullifier data structure.

8 Real World Interactions

In this section, we present the interactions between the wallet and the blockchain service and between the blockchain service and the oracle, and we explain how security is strengthened.

Redbelly Blockchain is dedicated to allow users to provide real-world services to one another. The key is to support the smart contracts already supported by Ethereum so as to minimise the efforts of porting existing contracts. We foresee the deployment of smart contracts encapsulating legal clauses. In order to tie these services to the real world, we will have to use oracles. As Redbelly Blockchain is secure, it is important that the oracle interactions do not introduce single point of failure vulnerabilities. To this end, Redbelly Blockchain will communicate with oracles whose implementation is distributed (tolerating isolated failures) or accountable (offering an insurance covering potential failures).

Accountability is instrumental in compensating the error committed by a centralized source. As an example a human error led Chainlink to report a price anomaly on their XAG/USD price feeds requesting gold price (XAU) instead of silver price (XAG) [7]. This was exploited by traders to generate ~US$36,000 in profit at the expense of Synthetix stackers. Distribution is instrumental in reducing such risks. For static information, the SEVM node requests $f + 1$ identical inputs from distributed sources to tolerate f failures. For dynamic information (e.g., stock value) or localised information (e.g., temperatures), the SEVM node extracts the median value among a group of $2f + 1$ distributed sources as we explain in §8.2.

8.1 Lightweight Secure Wallet

The lightsec wallet is both (i) lightweight in that it can run in handheld devices, and (ii) secure in that it can retrieve correct information. This is in contrast with traditional solutions where the device interacting with the blockchain must either trust the node it communicates with, which defeats the purpose of the blockchain; or download the blockchain itself, a task impossible for small devices that do not have sufficient storage space (recall that the Ethereum blockchain already exceeds 600 GB).

The lightsec wallet approach relies on the observation that to retrieve parts of the current state (e.g., the correct balance of a blockchain account), one simply needs to fetch $f + 1$ identical copies of this balance [32]. This is made possible by involving the participation of at most $2f + 1$ SEVM nodes that maintain a local copy of the blockchain and their current state or by requesting a threshold signature corresponding to such a quorum of nodes.

An interesting aspect of the lightsec client, is that it does not need to send its transaction to more than a single blockchain node to achieve the same liveness guarantee as Ethereum because it is anyway well-known that the blockchain cannot ensure that all transactions will be committed as we briefly mentioned in §5.2. This is why the blockchain problem is often defined with the liveness and uniformity properties [20, 8]. But since we require a blockchain to store only valid transactions, we also require to solve the additional validity property of [12] to solve this problem.

Definition 3 (The Blockchain Problem)**.** *The* blockchain problem *is to ensure that a distributed set of nodes maintain a sequence of transaction blocks such that the three following properties hold:*

- Liveness: *if an honest node receives a transaction, then this transaction will eventually be reliably stored in the block sequence of all honest nodes.*

- Uniformity: *the two chains of blocks maintained locally by two honest nodes are either identical or one is a prefix of the other.*

- Validity: *each block appended to the blockchain of each honest node is a set of valid transactions (non-conflicting well-formed transactions that are correctly signed by its issuer).*

Interestingly, the liveness property does not guarantee that a client transaction is included in the blockchain: if a wallet sends its transaction request exclusively to byzantine nodes then byzantine nodes may decide to ignore it. Hence, the lightsec wallet combined with Redbelly Blockchain guarantees this liveness property: it could be the case that a wallet has to send its transactions multiple times before it gets committed (just like in classic blockchains).

8.2 Oracles

An oracle is critical to feed major dApps with trustworthy off-chain information[1]. They can offer identification or payment services by indicating whether a bank payment is successful or whether the identity of a user has been verified.

Typically, a distributed oracle is a software running on some set of computers that gather real-world information from multiple websites in order to relay it to the blockchain (an oracle network [17] is an example of such a distribution). This information is typically used by dApps to trigger an action based on an external event. For example, an authentication system may accept some of the users based on its identity document. This acceptance is a piece of information from the real-world that an oracle must input to the blockchain for the dApp to authenticate the user. Typical examples include an API provider directly inputting this information to a smart contract [2] or offering an authentication proof to the end-user [35]. This input information can have such a large impact on the execution of the blockchain and the transfers of high value assets that it is important that it remains correct.

One cannot trust a single computer inputting this information, as the owner of the computer may easily get bribed by a bidder to steal the reward by pretending that their prediction was correct. This is why an oracle should either be accountable or distributed. One way to gather the information is by consulting online information on some online service. However, if the only service experiences a transient bug, then the blockchain assets are at risk. This is the reason why the distributed oracle (Dist. Oracle) will fetch the information from distinct sources (e.g., different websites) as depicted in Figure 2. To tolerate the failure of f online services announcing some real information, it is sufficient to gather $f + 1$ identical copies of the same information: this will guarantee that this information is correct. The worst case situation thus consists of contacting $2f + 1$ to retrieving the correct information.

Yet, there exist scenarios where the outcome is fluctuating: For example, a stock value on the New York Stock Exchange can change over time and no pair of sources could provide the exact same result due to message delays. In this case, the oracle will extract the median among $2f + 1$ requests to guarantee that no coalition of f byzantine machines can skew the outcome towards one end or the other. Another type of Oracle could be centralised and accountable (cf. Cent. Oracle in Figure 2) and under the control of an authority, like the SEC, that dictates how the regulation evolves over time and whose role is to guarantee that regulation is correctly followed at any time. In this case, we may consider the SEC information to be authoritative and render such a service accountable, removing the need to cross-check multiple sources.

[1] https://www.defipulse.com/.

9 Conclusion

Redbelly Blockchain is an innovative technology to interface the real world through a novel proof-of-fraud design. It ensures accountability of its participants and aims at complying with regulation. It builds upon recent research advances in the context of security [12], distributed computing [9], formal verification [4] and game theory [30]. Two of its key novelties is that it is deterministic, due to its consensus algorithm, and dynamic, due to its built-in smart contracts, that allow the governance to reconfigure it at runtime.

References

[1] Emmanuelle Anceaume, Antonella Del Pozzo, Romaric Ludinard, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Blockchain abstract data type. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, page 349–358, 2019.

[2] Burak Benligiray, Saša Milić, and Heikki Vänttinen. API3: Decentralized APIs for Web 3.0. Accessed: 2022-04-28 - https://drive.google.com/file/d/1GzkLKc6DYxImgeDhoKLA4wHGlE0eGGgo/view.

[3] Nathalie Bertrand, Vincent Gramoli, Igor Konnov, Marijana Lazić, Pierre Tholoniat, and Josef Widder. Compositional verification of Byzantine consensus. Technical Report hal-03158911, HAL, June 2021.

[4] Nathalie Bertrand, Vincent Gramoli, Igor Konnov, Marijana Lazić, Pierre Tholoniat, and Josef Widder. Brief announcement: Holistic verification of blockchain consensus. In *Proceedings of the 41st ACM Symposium on Principles of Distributed Computing (PODC)*, 2022.

[5] Duncan Black. *The Theory of Committees and Elections*. Cambridge University Press, 1958.

[6] Zohir Bouzid, Achour Mostfaoui, and Michel Raynal. Minimal synchrony for Byzantine consensus. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, page 461–470, 2015.

[7] Chainlink. Improving and decentralizing chainlink's feature release and network upgrade process, 2020. Accessed: 2022-03-31, https://blog.chain.link/improving-and-decentralizing-chainlinks-feature-release-and-network-upgrade-process/.

[8] Benjamin Y. Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 1–11, 2020.

[9] Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable Byzantine agreement. In *Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS)*, Jul 2021.

[10] Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. As easy as abc: Optimal (A)ccountable (B)yzantine (C)onsensus is easy! In *Proceedings of the 36th International Parallel and Distributed Processing Symposium (IPDPS)*, 2022.

[11] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: Efficient leaderless Byzantine consensus and its applications to blockchains. In *Proceedings of the 17th IEEE International Symposium on Network Computing and Applications (NCA)*, 2018.

[12] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red belly: a secure, fair and scalable open blockchain. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P)*, May 2021.

[13] Michael Dummett. *Voting Procedures*. Oxford University Press, 1984.

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

[15] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. Impact of man-in-the-middle attacks on ethereum. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 11–20, 2018.

[16] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The attack of the clones against proof-of-authority. In *27th Annual Network and Distributed System Security Symposium (NDSS)*, 2020. Presented at the Community Ethereum Development Conference in 2019.

[17] Steve Ellis, Ari Juels, and Sergey Nazarov. Chainlink: A decentralized oracle network, 2017. Accessed: 2022-04-28 - https://research.chain.link/whitepaper-v1.pdf.

[18] Amsden et al. The libra blockchain, 2020. Accessed on 2022-04-27, https://diem-developers-components.netlify.app/papers/the-diem-blockchain/2020-05-26.pdf.

[19] The eth2 upgrades. Accessed: 2022-03-28, https://ethereum.org/en/eth2/.

[20] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *34th Annu. Int. Conf. the Theory and Applications of Crypto. Techniques*, pages 281–310, 2015.

[21] Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[22] Vincent Gramoli. The red belly blockchain: Bft is back but is it the same? In *Workshop on Blockchain Technology and Theory collocated with DISC*, 2017. Invited Talk.

[23] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical Accountability for Distributed Systems. *SOSP'07*, 2007.

[24] Jonathan Lundell & David Hill. To advance the understanding of preferential voting system - notes on the droop quota. *Voting matters*, 2007.

[25] Igor Konnov, Marijana Lazić, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *Symposium on Principles of Programming Languages (POPL)*, pages 719–734. ACM, 2017.

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

[27] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. Accessed: 2022-05-03 - https://bitcoin.org/bitcoin.pdf.

[28] Christopher Natoli and Vincent Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017.

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

[30] Alejandro Ranchal-Pedrosa and Vincent Gramoli. TRAP: The bait of rational players to solve Byzantine consensus. In *Proceedings of the 17th ACM ASIA Conference on Computer and Communications Security (AsiaCCS)*, 2022.

[31] Deepal Tennakoon and Vincent Gramoli. Dynamic blockchain sharding. In *Proceedings of the 5th International Symposium on Foundations and Applications of Blockchain (FAB)*, volume 101. OASIcs, 2022.

[32] Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. CollaChain: A BFT collaborative middleware for decentralized applications. Technical Report 2203.12323, arXiv, 2022.

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

[34] Nicolaus Tideman. The single transferable vote. *Journal of Economic Perspectives*, 9(1):27–38, March 1995.

[35] Verite. Verifying verifiable credentials. Accessed: 2022-04-28 - https://docs.centre.io/verite/patterns/verification-flow.

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

[37] Douglas Woodall. Properties of preferential election rules. In *Voting Matters*, 1994.