Deconstructing the Smart Redbelly Blockchain

Deepal Tennakoon*† Vincent Gramoli*‡
*University of Sydney
{deepal.tennakoon, vincent.gramoli}@sydney.edu.au
†iGreenData – A Synechron Company ‡Redbelly Network

***Abstract*—Typical blockchain nodes replicate the execution of requests, which include native transfers and smart contract executions. With the shift of the Web towards Web3, modern blockchains suffer from congestion which prevents them to handle user requests. Recent studies showed that modern blockchains perform poorly or lose requests of realistic Decentralized Application (DApp) workloads, impairing the shift towards Web3.**

In this paper, we present Smart Redbelly Blockchain (SRBB) which handles realistic DApp workloads. SRBB improves blockchain performance with (1) Transaction Validation and Propagation Reduction, (2) caching optimizations and (3) fast block execution. SRBB outperforms Algorand, Avalanche, Diem, Ethereum, Quorum and Solana when deployed over 5 continents and under the realistic workloads of NASDAQ, Uber and FIFA using the DIABLO benchmark suite. Next, we decouple SRBB that improves the peak throughput of SRBB for the NASDAQ workload by 33% and reduces its latency by 20%.

***Index Terms*—Blockchain, Performance, Decoupling, Security**

I. INTRODUCTION

The Redbelly Blockchain [13] led to unprecendented performance: First, Redbelly scales to a geo-distributed network of a thousand machines because it combines the block proposals of all validators into a *superblock* instead of imposing the block of a single validator to the system [22]. Second, Redbelly prevents double spending despite arbitrary network delays by featuring a formally verified leaderless consensus protocol known as Democratic Byzantine Fault Tolerance (DBFT) [8]. Although this made blockchain consensus protocols scalable [24], Redbelly was limited by the Unspent Transaction Output (UTXO) [27]. More specifically, since the UTXO model uses multiple transaction outputs (i.e., UTXOs) from previous transactions as inputs of new transactions, the tracking of multiple UTXOs becomes complex when implementing smart contract logic [35].

In this paper, we present the provably secure *Smart Redbelly Blockchain* (SRBB) that copes with this problem. As its name indicates, SRBB builds upon the Redbelly Blockchain innovations, including its superblock and its formally verified consensus protocol, but extends it to support Solidity "Smart" contracts, hence offering compliance with the largest ecosystem of *Decentralised Applications (DApps)* that was originally developed for Ethereum [36]. Combining the performance and security of the Redbelly Blockchain with DApps makes SRBB an ideal blockchain to trade high value real world assets on Web3. One may think that the combination of Redbelly and the Ethereum Virtual Machine (EVM) is sufficient to combine the security and performance with the expressiveness needed by Web3, however, there are two limitations:

Decoupled blockchains	What is decoupled	Purpose
Hyperledger Fabric [7]	(1) Ordering and execution (2) Submits transactions in two rounds for ordering and execution.	Modularity
PRISM [33]	Decouples blocks into separate chains	Performance
SRBB-dec (our decoupling)	Consensus and execution	Performance

TABLE I: Blockchains with decoupled architectures.

1) **Redundant validations.** In modern blockchains [36], [21], [37], [11], [19], [1], including Ethereum, validators compete to get their block appended. To guarantee that a transaction will be included, each validator has to validate and propagate every transaction individually (and within a block), which incurs unnecessary overhead. To cope with this problem, SRBB combines proposed blocks into an agreed superblock and avoids having each validator propagate and validate every single transaction before inclusion in a block.
2) **Resource limitation.** Redbelly benefits from some memory optimisations that are possible with the UTXO model to obtain high throughput. In particular, it consumes as many UTXO as possible to keep the UTXO table in memory. By contrast, the EVM consumes more memory, CPU and storage I/O to execute bytecode and exchange messages. To minimize resource usage, we optimise the EVM to produce SRBB VM and decoupled the role of executing bytecode (SRBB VM) from the role of sending messages (consensus) in a variant of SRBB we call SRBB-dec.

These challenges are the reasons to extend substantially Redbelly and the EVM.

First, SRBB solves the aforementioned **Redundant Validations** problem by a transaction validation and propagation reduction mechanism. More specifically, SRBB does not propagate every transaction to all validators like in Ethereum. As a result, with n validators each transaction is validated only by the sole validator that receives it individually and then it is validated by each of the n validators right before the block that contains it gets executed by each validator. In Section VII we explain that SRBB does not impact a client's wait time for a transaction to be committed within a block despite not propagating transactions initially. Second, we solve the **Resource Limitation** problem in two steps. Initially we optimize the EVM to produce SRBB VM. SRBB consisting of SRBB VM and the previously mentioned transaction validation and propagation reduction mechanism, outperforms

Algorand [21], Avalanche [28], Diem [6], Ethereum [36], Quorum [11] and Solana [37] for three DIABLO benchmarks (NASDAQ, Uber and FIFA) [23]). Next, and as depicted in Table I, we decouple the node roles in SRBB to SRBB VM and consensus to produce SRBB-dec [31] such that each validator is split into a node running the consensus protocol and a node running the SRBB VM. This decoupling differs from Hyperledger Fabric [7] and PRISM [33] and improves the peak throughput of realistic DApp workloads (Section VI).

The remainder of this article is organized as follows. Section II presents the background and the problem. Section III presents Smart Redbelly Blockchain that addresses the redundant validation and resource limitation problem. We then extend Smart Redbelly Blockchain to SRBB-dec in Section IV to further address the resource limitation problem through decoupling. Sections V and VI present the evaluations of SRBB and SRBB-dec respectively. In Section VII we discuss the potential drawbacks of SRBB and SRBB-dec. In Section VIII we present our related work and in Section IX we conclude.

II. BACKGROUND

A. Nodes, blocks and transactions

We consider a distributed set of *nodes* that act either as a *client*, sending requests, or a *validator*, offering the blockchain service. (Note the distinction with the commonly used `Geth` clients that can also act as validators.) Validators that follow the blockchain protocol are called *correct* whereas those that deviate are called *Byzantine*. A client owns a private key and stores its digital assets in *accounts*, each comprising an *address* or the public key associated with the client's private key and a *balance* representing the amount on this account.

A *transaction* is a request that can be of three types: native transfer, smart contract upload, and smart contract invocation. A native transfer has a sender address a, a receiver address b, and an amount being transferred from a to b. Every transaction is associated with a nonce, a counter indicating the number of transactions ever sent from the same address. Two transactions *conflict* if they access the same data (e.g., smart contract variable) and one transaction is a write request [5].

A *block* is a batch of transactions. The validators execute blocks consisting of transactions submitted by clients. Upon executing each transaction in a block, validators update the balances of the sender and receiver addresses according to the amount of assets specified in the transaction. This is known as updating the blockchain state. The validators finally append the executed blocks to a chain of blocks known as the *blockchain*.

The notion of *gas* is used to reward validators based on their efforts to execute transactions. Each transaction, based on the type and amount of operations it contains, is associated with a *gas value*. The validator are rewarded for executing a transaction with its *transaction fee*, expressed as the product of the gas value and the current *gas price* [36] that usually depends on the network congestion.

B. Properties of a blockchain system

A blockchain system must ensure liveness and safety as previously stated by Garay et al. [20] as well as the classic validity property [15]:

Definition 1 (Blockchain System): A distributed set of validators implement a *blockchain system* if they maintain a sequence of transaction blocks such that the following properties hold:

- *Liveness:* if a correct validator receives a valid transaction, then this transaction is eventually reliably stored in the block sequence of all correct validators.
- *Safety:* the two chains of blocks maintained locally by two correct validators are either identical or one is a prefix of the other.
- *Validity:* each block appended to the blockchain of each correct validator is a set of valid and non-conflicting transactions.

The safety property does not require correct validators to share the same chain because one validator may already have received the latest block before another. Note that, as in classic definitions of liveness [20], [10], our liveness property does not guarantee a client transaction is included in the blockchain: if a client sends its transactions exclusively to Byzantine validators then these Byzantine validators may ignore it.

C. The redundant validation problem

The problem of many blockchains, like Algorand [21], Ethereum [36], Polkadot [19], Quorum [11], Solana [37] and Tezos [1] to name a few, is that their validators try to validate each transaction twice in an eager and a lazy validations.

Eager validation. An eager validation occurs when a validator receives a transaction either from another validator or a client. A validator then answers the following question regarding a transaction:

1) Is the transaction properly signed?
2) Does the sender account have sufficient funds to cover the specified payments to a receiver?
3) Is the transaction out of order (i.e., is the transaction sequence number too low or too high compared to the last received transaction from the same sender)?
4) Is there sufficient gas or is the transaction fee large enough to execute the transaction?
5) Is the transaction oversized?

If the eager validation of a transaction succeeds, the validator pushes the transaction to a pending queue in the transaction pool making eligible for inclusion ina block. The validator then propagates the valid transaction to downstream peers (i.e., its connected validators) that have not yet seen the transaction, eventually propagating the valid transaction throughout the network and having it eagerly validated at every validator. If the eager validation fails, validators drop the invalid transaction.

Lazy validation. Lazy validation occurs before the transactions in a block are executed and checks the sequence number of the transaction, whether the sender account has sufficient balance to send funds to the receiver, and the availability of gas to execute the transaction. Thus, lazy validation is less time-consuming than eager validation because the validity checks are comparatively less. Typically, lazy validation excludes signature and size checks.

Imposing a single block vs. combining multiple blocks. The crux of the problem is that, in these blockchains (e.g., Algorand [21], Ethereum [36], Polkadot [19], Quorum [11], Solana [37] and Tezos [1]), every validator is incentivised to impose its block to the rest of the system. To guarantee that a transaction gets included in the imposed block, the protocol requires this transaction to be propagated to as many validators as possible. This increases the chance that the transactions gets written to the blockchain.

Without the initial transaction propagation, a client's transaction may not be included in a block for some time if the validator directly receiving the transaction from the client is not selected to propose a block for an immediate consensus round. In contrast, even when the initial transaction propagation is removed on SRBB, a client's transaction can be immediately included in a block as SRBB's consensus decides all the blocks proposed by distinct validators into one larger superblock.

In the rest of this paper, we will show how to implement a blockchain system (Definition 1) without the initial propagation and eager validation of transactions (Section III-F). The idea is to reach a consensus on a superblock that combines the blocks of multiple validators so that transactions do not need to be propagated outside blocks. In our evaluation (Section V), we show the congestion impacts of this problem through transaction losses and performance degradation (throughput drop and latency increase) in modern blockchains under realistic DApp workloads.

D. The resource limitation problem

Decentralized applications, or DApps for short, are increasingly popular at allowing users to trade services transparently, peer-to-peer and without transferring ownership. In Q1 2023 DappRadar saw a 9% Q/Q growth rate of decentralized applications (DApps) totaling 17,564 DApps across all chains compared to Q4 2022.[1] These DApps are, however, plagued by the low throughput of the Ethereum blockchain capping at ∼15 transactions per second (TPS) [12]. Even without forcing miners to resolve a crypto-puzzle to obtain a proof-of-work, the proof-of-stake version of Ethereum, called "The Merge", does not increase the throughput and the proof-of-authority alternative of Ethereum delivers ∼80 TPS [26]. Ethereum already experienced congestion due to DApps [16] and its capacity is inherently too low to support a marketplace demand of 83 TPS [18]. Similar problems affect blockchains with different virtual machines as well. In particular, Solana was suspected to suffer DoS attacks because Solana is leader-based and its leader was overwhelmed by requests that slowed down the system[2]. It is thus crucial to better manage resources to offer multiple DApps through Web3.

III. Smart Redbelly Blockchain (SRBB)

In this section, we present SRBB, a permissionless blockchain that prevents the redundant validation and propagation of transactions. SRBB integrates a novel *Transaction Validation and Propagateion Reduction (TVPR)* to prevent the redundant validation and propagation of transactions. In addition, SRBB is compatible with the largest ecosystem of DApps, optimally resilient against Byzantine failures, and supports real DApp workloads like NASDAQ, Uber, and FIFA (Section V).

First, we present our assumptions followed by the TVPR. Second, we present the transaction life cycle of SRBB followed by the SRBB Virtual Machine (VM) implementation. Finally, we prove SRBB implements a Blockchain System (Def 1).

A. Assumptions

a) Partially Synchronous Model: Our network consists of a set of n validators that are well-connected. As consensus cannot be solved in the asynchronous setting, we assume *partially synchronous* communications similar to prior work [17]. In practice, we cope with partially synchronous communications by increasing timeouts [15].

b) Byzantine Model: We assume that out of n SRBB validators, at most f are *Byzantine* where $f < n/3$ (consensus is unsolvable in this model if $f \geq n/3$). The maximum number of Byzantine validators is thus $t = \lceil n/3 \rceil - 1$ such that $f \leq t$. Byzantine validators can act arbitrarily like proposing conflicting blocks, and invalid transactions. In general, any behavior that deviates from the blockchain protocol is considered a Byzantine behavior.

c) Threat Model: Several blockchains [25], [38], [4], [30] reconfigure their committee of n validators every epoch (i.e., a pre-specified unit of time) to prevent a majority of the committee from being bribed by a slowly-adaptive adversary. A slowly-adaptive adversary is defined as a malicious entity that can bribe validators progressively (not instantaneously) and only between epochs (not during an epoch) such that the total corrupted validators is f where $f < n/3$ at any time [38]. As n validators cannot reach consensus if $f \geq n/3$ validators are corrupted through bribery, the assumption of a slowly-adaptive adversary prevents consensus disagreements and double spending attacks. Therefore, we assume that the adversary is slowly-adaptive similar to prior works [25], [38].

B. Membership and committee reconfiguration

SRBB is a permissionless blockchain where (1) any node can join or leave the network and (2) any node has a chance to become a validator based on a periodic election process if they stake a certain amount of tokens. This membership approach of periodically rotating validators prevents an exclusive set of nodes from always being validators, providing a permissionless environment similar to Algorand [21]. An SRBB node can be (1) a client that sends transactions and reads the state of the blockchain (2) a validator that participates in consensus, executes transactions, and keeps a full state of the ledger to service clients, or (3) a *candidate* that is an willing to become a validator.

Initially, SRBB is bootstrapped with an initial set of validators pre-defined in the genesis block. Any candidate wants to be a validator in the future and must first express interest by depositing tokens (e.g., a pre-defined sum of native

[1] https://www.alchemy.com/blog/web3-developer-report-q1-2023.

[2] https://cryptopotato.com/solana-network-suffers-another-reported-ddos-attack.

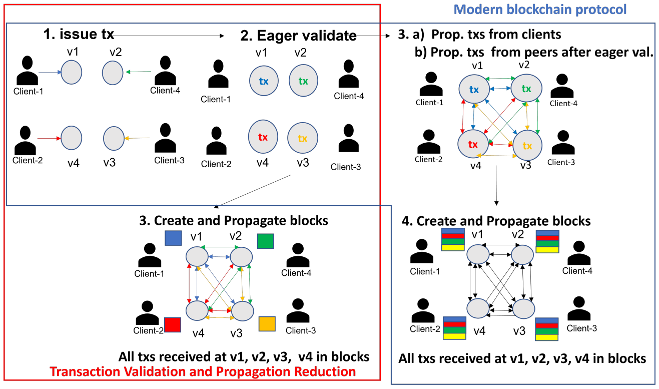

Modern blockchain protocol
1. issue tx
2. Eager validate
3. a) Prop. txs from clients
b) Prop. txs from peers after eager val.
v1
v2
v3
v4
Client-1
Client-2
Client-3
Client-4
tx
3. Create and Propagate blocks
4. Create and Propagate blocks
All txs received at v1, v2, v3, v4 in blocks
Transaction Validation and Propagation Reduction
All txs received at v1, v2, v3, v4 in blocks

Fig. 1: The modern blockchain protocol and TVPR modification are represented graphically at a high level.

coins) in a reconfiguration smart contract. SRBB periodically elects a committee of validators from the set of candidates. During this election, the current set of validators elect the next set of validators using the reconfiguration smart contract. The details of the validators of a committee are registered in the smart contract after an election outcome and each SRBB validator gets to know of other validators in the committee through an event emitted by the reconfiguration smart contract. This periodic reconfiguration of SRBB validators makes it hard for a slowly-adaptive adversary to bribe sufficiently many validators to control the election.

The requirement to deposit tokens to be a candidate mitigates the risk of Sybil attacks. In particular, it makes it costly for a single user to control a large number of candidates. The reconfiguration can offer proportionality and non-dictatorship in the governance as can be found in social choice theory. Detailing the steps of this reconfiguration process is out of the scope of this paper and can be found elsewhere [30].

C. TVPR (Transaction Validation and Propagation Reduction)

In TVPR, instead of validators eagerly validating and propagating transactions received from other validators individually, validators only eagerly validate transactions received directly from clients. These validated transactions are then included in blocks and broadcast to all validators. In other words, we remove step (3) in Figure 1 of the classic blockchain protocols where validators individually propagate transactions among themselves. To this end, we remove line 10 of Alg. 1 to prevent validators from propagating transactions outside blocks. This way, not all validators eagerly validate each transaction. As a result, in SRBB only one validator (the one receiving the transaction for the first time) eagerly validates it – other validators simply validate it only after receiving it within a block. This is in contrast with classic blockchains whose validators all eagerly validate every single transaction before the transaction gets included in a block. In Section V-B, we present the throughput improvement, latency reduction and transaction loss reductions of integrating TVPR to SRBB when compared to other modern blockchains. The potential drawbacks of TVPR are deferred to the discussion (Section VII).

Algorithm 1 The Smart Redbelly Blockchain protocol

```
1:  State:
2:    $blockchain$, an array of blocks, initially:
3:      $blockchain[0] = \text{genesis-block}$
4:    $\mathbb{P}$ is the txpool pending queue

5:  receive($t$):                    ▷ $t$ received from neighbors or directly from clients
6:    if eager-validate($t$) then              ▷ if eager validation succeeds
7:      if $t \notin blockchain$ and $t \notin \mathbb{P}$ then      ▷ $t$ not present
8:        $\mathbb{P} \leftarrow \mathbb{P} \cup t$          ▷ add to txpool pending queue
9:        if TTL of $t$ not exceeded then
10:         propagate($t$)                  ▷ SRBB omit this classic step
11: propose():
12:   $b_i \leftarrow$ create-block-with($\mathbb{Q}$)   ▷ create a block $b_i$ from $\mathbb{Q} \subset \mathbb{P}$
13:   $blockQueue \leftarrow blockQueue$.append($\mathsf{b_i}$)  ▷ append to queue
14:   $\mathbb{P} \leftarrow \mathbb{P} \setminus \mathbb{Q}$        ▷ remove $txs$ from txpool to create $b$
15:   $b_i \leftarrow blockQueue[0]$          ▷ get first element from block queue
16:   propagate($b_i$)                ▷ propagate block via rb-broadcast

17: Upon reception of $\mathbb{B}$ for index $k$ s.t. $\mathbb{B} \leftarrow \bigcup_{i=1}^{j} b_i$:▷ delivered blocks
18:   $\mathbb{B}^* \leftarrow$ consensus($\mathbb{B}$)   ▷ exec. cons. and decide $\mathbb{B}^*$ s.t. $\mathbb{B}^* \subseteq \mathbb{B}$
19:   for all $b_i \in \mathbb{B}^*$ starting from $i = 1$ do
20:     for $t \in b_i$ do
21:       $err \leftarrow$ execute($t$)
22:       if $err \neq$ null then
23:         $b_i \leftarrow b_i - t$              ▷ remove invalid $t$ from $b_i$
24:     if $b_i \neq$ null then                ▷ $b_i$ has transactions
25:       $blockchain[k] = b_i$            ▷ insert to permanent chain
26:       $k$++
27:   $blockQueue$.remove($\mathbb{B}*$)  ▷ remove decided blocks from queue

28: execute($t$):
29:   $err \leftarrow$ lazy-validate($t$)                ▷ lazy validation
30:   if $err \neq$ null then
31:     return $err$
32:   $S_r, err \leftarrow$ ApplyTransaction($t, S_i$)      ▷ apply $t$ on state $S_i$
33:   if $err \neq$ null then
34:     return $err$
35:   else
36:     return null
```

D. Transaction life cycle of SRBB

We now present the transaction life cycle of SRBB. An SRBB node consists of the SRBB VM and SRBB consensus. The SRBB VM is built upon `Geth` and integrates with TVPR. SRBB consensus uses the DBFT consensus protocol [13] combined with the superblock optimization of RBBC [15]. We omit the details of the consensus protocol, as the DBFT algorithm [13] and its integration into the Redbelly blockchain design [15] can be found in the literature. A transaction submitted by a client to SRBB goes through the stages below:

Reception. The client creates a properly signed transaction and sends it to at least one SRBB validator where the transaction is eagerly validated (Alg. 1, line 6). If the eager validation fails, the transaction is discarded. Otherwise, the transaction is kept in a pending queue in the transaction pool (Alg. 1, line 8). Once the size of the transaction pool pending queue reaches a threshold (pre-defined in the configuration), a validator creates a block b_i and adds it to a block queue (Alg. 1, line 13). Subsequently, the transactions used to create the block are removed from the transaction pool pending queue $\mathbb{P}$. Next, the validator fetches the first block from the block queue (Alg. 1, line 15) and propagates it to all validators (Alg. 1, line 16). Note that, unlike other blockchains, SRBB does not propagate transactions individually. Instead, SRBB simply includes transactions in blocks and propagates blocks to the network, hence implementing our TVPR solution. For each index of the blockchain, every correct SRBB validator propagates a block b_i s.t. i is the ID of the SRBB validator and $i \in \mathbb{Z}^+$. These blocks are propagated via reliable broadcast [9].

Consensus. Upon receiving a set of blocks $\mathbb{B} = \bigcup_{i=1}^{j} b_i$ from validators $j \leq n$, a validator starts executing the DBFT consensus protocol [13], [8] (Alg. 1, line 18) as outlined in Alg. 2, and decides the next superblock $\mathbb{B}^*$. Upon deciding the superblock, the validator sends it to the SRBB VM for execution (Alg. 1, lines 19-26). The decided blocks in the superblock are removed from the block queue (Alg. 1, line 27). All undecided blocks are kept in the block queue to be included in future consensus rounds. The consensus proceeds to the next round after a superblock is decided, and starts propagating a block again from the front of the block queue via reliable broadcast, if not already propagated (Alg. 1, lines 15-16). Note that the only similarity between SRBB and RBBC [15] is this consensus protocol. RBBC does not support the execution of smart contracts or DApps and does not feature TVPR.

Commit. An SRBB VM, upon receiving the superblock $\mathbb{B}*$, takes a block b_i at a time from $\mathbb{B}*$, iterates through its transactions (Alg. 1, line 20), and attempts to execute them (Alg. 1, line 21). In the execution process, first, the SRBB VM lazily validates the transaction (Alg. 1, line 29). If a transaction's lazy validation succeeds, the SRBB VM attempts to apply the transaction to the current blockchain state (Alg. 1, line 32). A state transition for executing a transaction only occurs if the transaction is valid and non-conflicting. Since lazy validation is not as strict as eager validation (Section II), a transaction may pass the lazy validation but still be invalid. The SRBB VM like modern blockchains handles such cases by throwing an error without transitioning to another state (Alg. 1, line 34). More specifically, during transaction execution, the SRBB VM rechecks other validity criteria not present in lazy validation (e.g., transaction signature verification, transaction size limit check), and throws an exception if invalidity is found.

If either the lazy validation fails or applying the transaction fails, then the transaction is deemed invalid. In this scenario, the SRBB VM discards the invalid transaction from the block b_i (Alg. 1, line 23) and moves on to the next transaction in the block. Later, b_i is appended to the blockchain (Alg. 1, line 25) and the SRBB VM moves to the next block in the superblock. The SRBB VM follows the same procedure to process the subsequent blocks in the superblock until all the valid blocks are written to the blockchain.

Algorithm 2 The SRBB consensus protocol

```
 1: $\mathbb{B} \leftarrow \{b_1, b_2, ...b_i\}$    ▷ reliably delivered blocks
 2: $blocks \leftarrow \emptyset$ ▷ blocks delivered from rbbroadcast are stored here in line 8
 3: $index \leftarrow 0$    ▷ consensus round
 4: $blockQueue \leftarrow \emptyset$    ▷ pending blocks to propose to consensus
 5: consensus($\mathbb{B}$):
 6:     $decCount \leftarrow 0$, $decBlocks \leftarrow 0$
 7:     for all $block \in \mathbb{B}$ do
 8:         $blocks[i] \leftarrow block$ ▷ add rb-broadcasted blocks to list s.t. $i$ is ID of sender
 9:         $decBlocks[i] \leftarrow$ b-cons-propose($i$, true)    ▷ props. to binary cons.
10:     wait until $\exists i$ : b-cons-decide(i) is $true$    ▷ till binary cons. decided
11:     for $j$ from 0 to $n$ do
12:         if $blocks[j] == \emptyset$ then
13:             $decBlocks[j] \leftarrow$ b-cons-propose($j$, $false$)
14:         if $decBlocks[j] ==$ true then $decCount \leftarrow decCount + 1$
15:     wait until $decCount == n$
16:     $superblock \leftarrow \emptyset$
17:     for $i$ from 0 to $n$ do
18:         if $decBlocks[i] ==$ true then
19:             $superblock \leftarrow superblock \cup blocks[i]$ ▷ combine decided blocks
20:     return $superblock$
```

E. SRBB VM implementation

The SRBB VM results from optimizing the `Geth` EVM and integrating TVPR to prevent redundant eager validation and propagation of transactions. We now present all these optimizations.

TVPR implementation. We integrated TVPR into the EVM by disabling the initial individual transaction propagation among validators. This way the first SRBB node receiving transactions from clients, eagerly validates and includes the transactions in blocks before propagating them, in blocks, to the network. One might think that SRBB allows the execution of invalid transactions because a transaction is eagerly validated by only one SRBB validator and then lazily validated at each SRBB validator prior to being executed, and because the lazy validation is not as strict as the eager validation. Note that the reduction of transaction validations does not cause the execution of invalid transactions in SRBB. Instead in the case of invalid transactions, the SRBB execution throws an exception. More precisely, a transaction is valid only if (i) the transaction is properly signed, (ii) its size does not exceed a limit, (iii) its nonce is the next sequence number, (iv) its gas cost is covered by the sender balance, (v) its transferred amount is covered by the sender balance. The lazy validation checks (iii), (iv), (v) whereas the execution checks (i) and (ii). The `Geth` implementation[3] which SRBB builds upon, raises an ErrInvalidSig exception if (i) is not satisfied. The original VM, even in its original release, raises an overflow or exceptions if (ii) is not satisfied.[4]. Thus, even if an invalid transaction is lazily validated due to the fewer number of checks, the SRBB VM's execution will perform additional checks to ensure that no invalid transactions are executed.

In order to implement TVPR, we had to change the intricate parts of the original `Geth` implementation, which required a

[3] L635 of https://github.com/ethereum/go-ethereum/blob/master/core/types/transaction.go of commit c4a6621

[4] L187-219 of https://github.com/ethereum/go-ethereum/blob/master/core/vm/interpreter.go, and https://github.com/ethereum/go-ethereum/blob/master/core/vm/errors.go.

deep dive into the implementation details of Ethereum. In total, we changed 161 lines of code. These changes included: (1) disabling the event that notifies successful eager validation of each transaction to the function that broadcasts transactions individually and (2) disabling functions that handle individual transactions received from peers. Our implementation of SRBB VM that encapsulates TVPR can be found at [29]. Note that since we developed SRBB VM, the official implementation of `Geth` has changed. To the best of our knowledge, none of these changes have impacted the relevance or the guarantees of TVPR. This is because after reducing eager validations, if an invalid transaction bypasses lazy validation, the execution attempts to execute the invalid transaction and throw an exception if any remaining invalidity is found, without transitioning to another state.

SRBB VM support for fast-paced consecutive blocks. Since our consensus is fast, it creates and delivers superblocks at high frequency to the SRBB VM. As `Geth` does not expect to receive blocks at such a high frequency, it raises an exception outlining that consecutive block timestamps are identical, which never happens in a normal execution of Ethereum. This equality arose because `Geth` encodes the timestamp of each block as `uint64`, not leaving enough space for encoding time with sufficient precision. `Geth` typically reports an error when consecutive timestamps are identical, due to a strict check that compares the parent block timestamp to the current block timestamp in `go-ethereum/consensus/ethash/consensus.go`: `header.Time < parent.Time`. We changed the original check to `header.Time <= parent`, which allowed for fast-paced executions of consecutive blocks.

Caching optimizations. We cached the block body that contains the transactions committed on the blockchain. A transaction is considered final once the client receives the receipt of the transaction from the blockchain. When retrieving the receipt of transactions initially, the EVM fetches transactions from the block body in the LevelDB. These transactions are then used to retrieve the transaction receipts. We cache the block body to speed up the return of the transaction receipt to the client, and hence, speed up transaction finality. We also cached the transaction receipts. Once a transaction receipt is fetched by a client, the EVM caches them to help fast retrieval of those receipts if queried again. However, this does not speed up the first time the receipt is retrieved. Therefore, we decided to cache receipts after transaction execution prior to writing receipts to the key value data store.

F. Proofs of correctness of SRBB

Here we present the proofs of correctness of SRBB. First, we prove that SRBB is a Blockchain System (Section 1) as defined in Def. 1.

Theorem 1: SRBB satisfies the properties of a Blockchain System.

Proof: We prove that each property of Def. 1 is preserved:

1) Liveness: As a result of removing line 10 of Alg. 1 in SRBB, transactions are no longer propagated individually to the network among validator peers and eagerly validated at each SRBB validator. However, correct validators still create blocks including valid transactions, and propagate them to peers (Alg. 1, line 16). Thus, every correct SRBB validator receives a set of blocks $\mathbb{B}$ propagated by correct SRBB validators at index k (Alg. 1, line 17). An SRBB VM decides a set of blocks $\mathbb{B}^*$ s.t. $\mathbb{B}^* \subseteq \mathbb{B}$ (a superblock) and stores the valid transactions in these decided blocks on the blockchain (Alg. 1, lines 18-25). While the decided blocks are removed from the block queue (Alg. 1, line 27), undecided blocks are kept in the SRBB block queue to be included in future blocks. These transactions are eventually re-included in a future decided block by correct SRBB validators and stored in the blockchain. Thus, every transaction received by a correct SRBB validator is eventually stored in the block sequence of all correct SRBB validators.
2) Safety: The preservation of safety is proved by contradiction. If none of the two chains of blocks maintained locally by any two correct SRBB validators $v1$ and $v2$ is a prefix of one another, it means the superblock $\mathbb{B}^*$ decided at index k (Alg. 1, line 18) of $v1$ and the superblock $\mathbb{B}^{*\prime}$ decided at index k of $v2$ are different. This results in two different transaction executions (Alg. 1, line 21) for $v1$ and $v2$. However, this is a contradiction because any two correct SRBB validators $v1$ and $v2$ should decide on the same superblock at index k due to consensus guarantees of DBFT [13] (Alg. 1, line 18). Thus, any two validators $v1$ and $v2$ should store the same block b at index k of the chain (Alg. 1, line 25). Therefore, any two correct validators should maintain locally an identical chain of blocks or a chain where one is a prefix of another (i.e., because blocks do not get decided, executed, and stored at the same time in all SRBB validators) resulting in the same execution. Thus, through proof by contradiction, SRBB achieves safety.
3) Validity: Due to the consensus protocol, all correct SRBB validators decide on the same superblock $\mathbb{B}^*$ at index k (Alg. 1, line 18). If each block b_i in the superblock $\mathbb{B}^*$ has a valid set of transactions, it is appended to the blockchain (Alg. 1, lines 20-25). Thus, each block appended to the blockchain of each correct SRBB validator is a set of valid non-conflicting transactions.

IV. SRBB-DEC: DECOUPLED SRBB

In this section, we present SRBB-dec, a version of SRBB that decouples the consensus (Consensus System) from the SRBB VM (Replicated State Machine) to better allocate resources across disjoint machines.

Just as before, we assume that there are n SRBB-dec participants such that at most f are *Byzantine* where $f < n/3$. However, we consider that each participant runs two machines, one dedicated to run the SRBB VM role and another to run the consensus role. Hence, in the remainder we consider $m = 2n$ nodes in the system.

A. Architecture

The layered architecture of SRBB-dec is depicted in Figure 2 with an SRBB VM node at the top, and a consensus node at the bottom. The communication between the consensus node and the SRBB VM node is event-based and implemented with gRPC. Although the event-based communication adds some communication overhead, the execution of the consensus node and the SRBB VM of SRBB-dec on separate machines offers greater modularity and, as we will show in Section VI, better performance than SRBB.

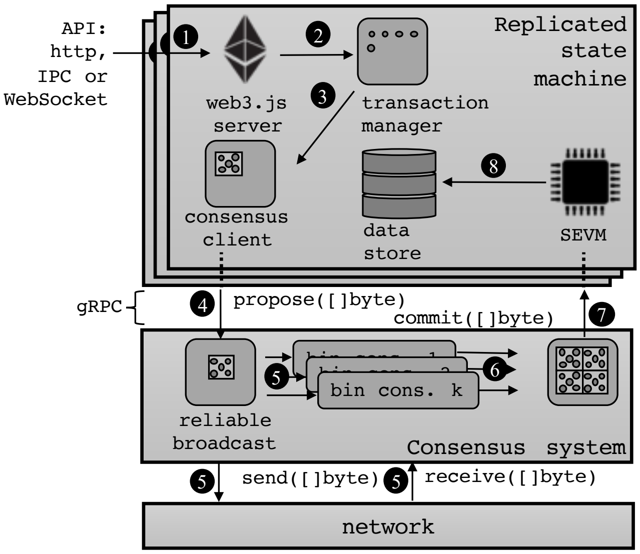


Fig. 2: The architecture of SRBB-dec. ❶ A client sends a transaction to some Smart Redbelly Blockchain VM node (SRBB VM), ❷ at each replica, the web3.js server eagerly validates transactions and sends them to the transaction pool that ❸ sends a block to the consensus client. ❹ The consensus client proposes it to the consensus protocol. Upon reception of a new block from the consensus client, ❺ the consensus server in the consensus node propagates it through the network using reliable broadcast [14]. Remote consensus nodes start participating in the same instance (if not done yet) upon reliably delivering this proposed block. ❻ When the consensus outputs some acceptable blocks, all of these blocks are combined into a superblock ❼. The VM client sends this superblock to the SRBB VM (SEVM) by invoking commit, the VM server upon receiving the superblock sends the block to be executed and stores it in the data store ❽.

B. The transaction life cyle of SRBB-dec

Since SRBB-dec is the decoupled version of SRBB, there are slight variations in their architecture and their transaction lifecycle. The SRBB VM and consensus are decoupled in SRBB-dec across two machines, thus communications occur between the consensus instance and the state machine instance. For these RPC calls, we added some components to SRBB-dec's system architecture. The lifecycle of a transaction in SRBB-dec goes through the stages below:

Reception. The client creates a properly signed transaction and sends it to at least one SRBB VM node. Once a request containing the signed transaction is received ❶ by the JSON RPC server of SRBB VM, the eager validation starts. If the validation fails, the transaction is discarded. If the validation succeeds, the transaction is added to the transaction pool ❷. Unlike modern blockchains where the transaction is propagated to all validators increasing the number of eager validations, SRBB-dec includes transactions in blocks directly from the transaction pool. This is because SRBB-dec includes TVPR from SRBB. Once a threshold of transactions has been received, the SRBB VM serializes blocks created from the transaction pool and sends them to the consensus client ❸.

Consensus. Once the consensus client receives a proposed block, it sends the corresponding byte array to the consensus system by invoking the propose([]byte) method using gRPC ❹. The consensus server upon receiving this byte array starts a new instance of consensus using the new block if it is not currently part of another consensus instance and reliably broadcasts the block to the network of consensus nodes ❺. Otherwise, it adds the new block to the block queue waiting for the current consensus instance to terminate. The consensus execution then invokes a *binary* consensus instance for each reliably delivered block. The output of the binary consensus instance indicates the indices of acceptable blocks ❻ as detailed before in Alg. 2, line 18. The consensus system creates a superblock with all acceptable blocks (Alg. 2, line 19) and sends it to the VM client ❼. The VM client sends the superblock to the SRBB VM via gRPC by invoking the commit([]byte) method.

Commit. When the superblock is received by the VM server on SRBB VM, the superblock is first deserialized using JSON unmarshalling. Subsequently, the superblock is passed on to the execution module of the SRBB VM ❼. The execution of transactions in SRBB-dec is identical to SRBB. The SRBB VM lazily validates and executes transactions by iterating through every block in the superblock and every transaction in each block (Section III-D). Finally, the SRBB VM appends each block to the ledger in the datastore ❽.

V. Evaluation of Smart Redbelly Blockchain

In this section, we evaluate SRBB and its optimizations and compare it to 6 other modern blockchains when deployed across on 200 machines spread over 5 continents.

We evaluated SRBB using the Diablo blockchain benchmark suite [23] that evaluates blockchains against specified workloads by sending pre-signed transactions. We used the realistic DApp workloads of NASDAQ, Uber, and FIFA that span 3, 2 and 3 minutes respectively (https://github.com/lebdron/minion/tree/aec). NASDAQ (peak request rate of 19800 TPS, avg. request rate of 168 TPS) uses a real trace of Apple, Amazon, Facebook, Microsoft, and Google stock trades executed on a DApp, Uber (peak request rate of 900 TPS, avg. request rate of 852 TPS) uses a real trace from the mobility service Uber executed on a DApp and FIFA (peak request rate - 5305 TPS, avg. request rate - 3483 TPS) uses a real workload from the soccer world cup executed on a DApp. For the DApp workload experiments, we used 200 validators spanning 10 AWS regions and 5 continents, namely: Bahrain, Cape Town, Milan, Mumbai, N. Virginia, Ohio, Oregon, Stockholm, Sydney, and Tokyo. For all DApp benchmarks, we used the same AWS c5.2xlarge EC2

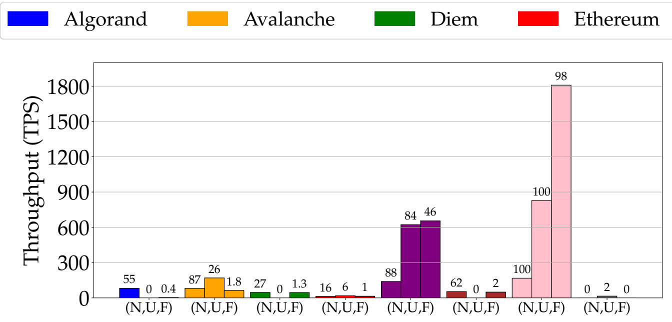


Fig. 3: Throughput (y-axis) and commit percentage (top of the bar) for NASDAQ (N), Uber (U) and FIFA (F) workloads,

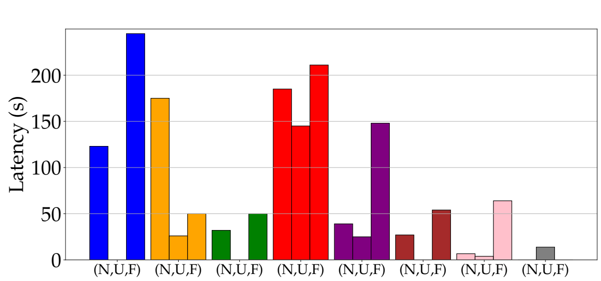


Fig. 4: Latency (y-axis) for NASDAQ, Uber and FIFA workloads (i.e., (N,U,F) is NASDAQ, Uber and FIFA)

instances (8 vCPUs, 16 GB RAM – equivalent to a modern-day PC) as DIABLO [23].

A. Rationale for machine selection

The c5.2xlarge AWS instance type was consistently used throughout all benchmarks for two reasons. First, to make all results comparable within our paper and with DIABLO [23]. Second, to make the evaluations encompass a wide range of blockchains as some blockchains require specific CPU and memory requirements (e.g., Solana) that cannot be met with smaller AWS instances.

In summary, all experimental parameters were the same as the ones used in DIABLO [23] for realistic DApp evaluation. Similar to the DIABLO DApp evaluations [23], all workloads were evaluated with one full experimental iteration.

Throughout this section, our evaluation focuses on the throughput, latency and transaction loss of blockchains which are indicators of blockchain congestion. The throughput is the number of transactions committed per second as observed by the client. The latency of a transaction is the difference between the transaction send time and the transaction commit time (i.e., the time that a client receives sufficiently many confirmation ACKs for a transaction sent) as seen by the sending client. The transaction loss is the number of dropped transactions. With more congestion, a blockchain's throughput drops, and latency and transaction losses increase.

In summary, SRBB reached a maximum average throughput of ~2000 TPS for realistic DApp workloads. SRBB outperformed (i.e., higher throughput, fewer transaction losses) 6 modern blockchains for the realistic DApp workloads of NASDAQ, Uber, and FIFA [23]. Moreover, SRBB was the only of these blockchains to not lose transactions for the realistic DApp workloads of NASDAQ and Uber, and commit over 98% of transactions for the demanding workload of FIFA. The higher throughput, lower latency, and fewer transaction losses of SRBB compared to modern blockchains indicate SRBB reduces blockchain congestion.

B. Comparison with other blockchains

Figures 3 and 4 depict the performance of 6 modern blockchains (i.e., Algorand, Avalanche, Libra-Diem, Ethereum Proof-of-Authority, Quorum IBFT, Solana) compared to SRBB and EVM+DBFT which is a naive smart contract supported version of RBBC that combines the Ethereum VM with the superblock optimized DBFT consensus but does not have TVPR and RPM. The evaluation of EVM+DBFT is included to show that the performance benefits of SRBB come from TVPR and not from the prior works of the superblock optimization and the DBFT consensus of RBBC.

We used the real DApp workloads of NASDAQ, Uber, and FIFA used in the DIABLO blockchain benchmarking suite [23] for our evaluation. Evaluating all blockchains in existence against SRBB is not realistic. We used the 6 modern blockchains evaluated in DIABLO [23] for comparison against SRBB because DIABLO reported a thorough evaluation of these blockchains under realistic DApp workloads [23]. To make the comparison fair, we used the same configuration parameters as used in DIABLO [23] when evaluating SRBB.

Note, some blockchains did not yield an average latency or throughput value for certain workloads (e.g., 0 TPS and 0 s latency) because the transaction costs exceeded their budget [23] or they crashed due to the high load.

Figure 3 presents the average throughput in the y-axis and transaction commit percentage as a value at the top of the bar for the NASDAQ, Uber and FIFA workloads (i.e., depicted by (N,U,F) in Figure 3 for brevity). SRBB is the only blockchain to commit 100% of the transactions for the NASDAQ and Uber workloads. SRBB also commits 98% of transactions for the demanding FIFA workload where no other blockchain commits more than 47% of transactions. SRBB reaches average throughputs of 167 TPS, 828 TPS, and 1808 TPS for the NASDAQ, Uber and FIFA workloads respectively, which are the highest average throughputs for all evaluated blockchains. Figure 4 shows the average latencies for the (N,U,F) workloads. SRBB yields the least average latency among all evaluated blockchains for both the NASDAQ and Uber workloads with average latencies of 6.6 and 3.9 seconds respectively. For the FIFA workload, SRBB yields an average latency of 64 seconds. This slightly higher average latency of SRBB over Avalanche, Diem and Solana in the FIFA workload is due to SRBB committing 98% of the transactions as opposed to only 2% or fewer transaction commits in the other blockchains. All 6 modern blockchains yield throughputs lower than 900 TPS and latencies higher than 20 seconds. These performances are much lower compared to their claimed performances [23]. This indicates a major performance degradation.

Importantly, SRBB multiplies the average throughput by

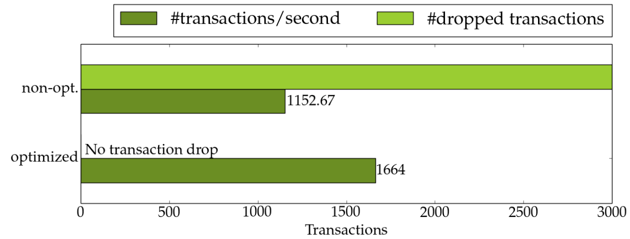


Fig. 5: The Optimized handling of the superblock in SRBB and non-optimized handling of the superblock

55×, divides the latency by 3.5×, and reduces transaction losses considerably compared to EVM+DBFT. This shows that TVPR is responsible for higher performance and reduction in transaction losses.

C. *Superblock handling at the SRBB VM*

At each index of the blockchain, the SRBB VM typically executes many more transactions per consensus instance than Ethereum. This is due to a single consensus instance outputting a superblock which contains potentially as many blocks as SRBB validators. Due to this excessive execution of transactions, the SRBB VM consumes high CPU. Typically, high CPU usage slows down the processing of transactions in the SRBB VM which results in the increase of pending transactions stored in-memory. Once a threshold of pending transactions are reached, we observe transaction drops due to the transaction pending queue overflowing. Also, when the memory usage is high, the SRBB VM flushes tries to disk to save memory, which also consumes a lot of I/O. This was evident in our superblock implementation. We observed that processing a superblock with a total of 15,000 transactions, would lead to losing transactions requests even on a reasonably-provisioned AWS instances featuring 16 GB RAM and 4 vCPUs (Figure 5). As a solution we made the SRBB VM fully process one proposed block of the superblock at a time allowing it to alternate frequently between CPU-intensive (verifying signatures and performing transaction executions) and memory-intensive (state writes) and IO-intensive (transaction writes) tasks. Due to this optimized handling of the superblock, we will see in the next section that our SRBB-dec does not experience bottlenecks as the number of nodes increases (cf. Figure 8).

VI. Evaluation of SRBB-dec

In this section we compare SRBB-dec with SRBB and present the scalability of SRBB-dec on demanding workloads.

A. *SRBB-dec vs SRBB*

Below we compare the performance of SRBB-dec and SRBB. We deployed 200 c5.2xlarge (8 vCPUs, 16 GiB of memory) geo-distributed machines spanning the 10 AWS regions on 5 continents used in the previous section, namely: Bahrain, Cape Town, Milan, Mumbai, São Paulo, Ohio, Oregon, Stockholm, Sydney, and Tokyo. The difference with the previous section is that these 200 machines of SRBB-dec consisted of 100 consensus nodes and 100 SRBB VM nodes. Due to the nature of the SRBB-dec, instead of deploying 20 SRBB nodes per region as in the previous section, we deployed 10 consensus nodes and 10 SRBB VM nodes per region, hence mimicking 100 participants with two machines each. Note that the total numbers of machines used to run both blockchains were kept equal to ensure as much resources were used for both to obtain a fair comparison. We elaborate more on the fairness of this evaluation as a part of our discussion section (Section VII).

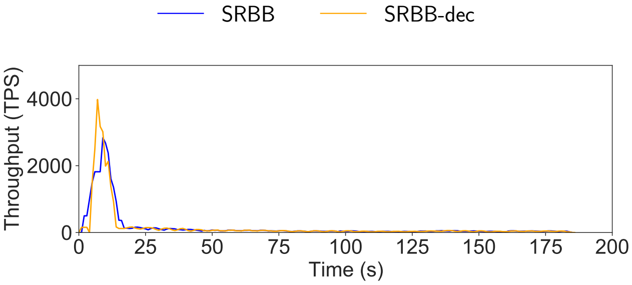


Fig. 6: The throughput over time of SRBB-dec (The decoupled version of SRBB) and SRBB for the NASDAQ workload.

Fig. 6 shows the throughput over time for SRBB-dec and SRBB. SRBB-dec reaches a throughput of 4000 TPS whereas SRBB reaches a throughput of ∼3000 TPS. Thus, SRBB-dec improves the throughput of SRBB by 33% for the same number of machines.

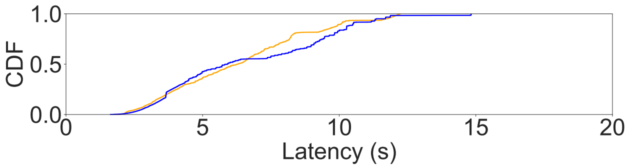


Fig. 7: The cumulative distribution function (CDF) of latencies of SRBB-dec and SRBB for the NASDAQ workload.

Figure 7 shows the Cumulative Distributed Function (CDF) latencies of SRBB-dec and SRBB. We observe that SRBB-dec commits 100% of the transactions within 12 seconds whereas SRBB takes 15 seconds to commit 100% of the transactions.

B. *Scalability of SRBB-dec*

To evaluate the scalability of SRBB-dec, we deployed SRBB-dec on SRBB VM nodes of type c5.2xlarge with 8 vCPUs, 16 GiB of memory, and consensus nodes of type c5.4xlarge with 16 vCPUs, 32 GiB of memory. The chosen 10 regions in 5 continents were Canada, London, Mumbai, Oregon, Paris, São Paulo, Singapore, Stockholm, Sydney and Tokyo. Each SRBB VM node received a burst workload of 1500 native payment transactions concurrently at a rate of 1500 TPS. As previously mentioned, each participant was considered to execute both a consensus node and a SRBB VM node.

Figure 8 depicts the average throughput without end-to-end encryption (w/o TLS) and with encryption (with TLS) for SRBB-dec. As we execute SRBB-dec with an increasing number of nodes, we start our experiment with 20 machines spread evenly in the 10 AWS regions and add machines by groups of 20 evenly spread in the 10 AWS regions until we reach 200 machines (i.e., 100 participants). We observe that the throughput increases as we increase the number of nodes from 1100 TPS at 20 machines to 2038 TPS at 200 machines, demonstrating the scalability of SRBB-dec in a geo-distributed setting. The curve flattens out at large scale between 140 and 200 nodes, indicating that the gain obtained in throughput by adding more machine becomes lower and lower. This is due to an increase in the number of machines consuming the available bandwidth. Finally, we observe, as expected, that the TLS encryption comes at a cost. However, this overhead is negligible in comparison to the overall performance as the peak average throughput with TLS (1960 TPS) is only 4% lower than the peak average throughput without TLS (2038 TPS).

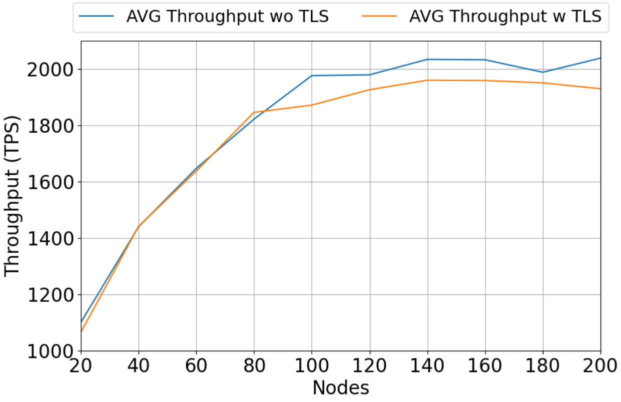


Fig. 8: The average throughput of SRBB-dec w/ and w/o TLS

Figure 9 shows the latency of transactions for SRBB-dec in the aforementioned geo-distributed environment as the number of nodes increase. We observe that the latency increases with the number of nodes. We observe similar minimum latencies across all system sizes but the 99^{th} percentile indicates that some requests can take much longer especially at large scale: the transactions take less than 10 seconds to execute on up to 40 nodes while they take less than 40 seconds to execute at 200 nodes. It is important to note that these latencies can be viewed as time for a transaction to become final: thanks to our deterministic byzantine fault tolerance consensus [13], transactions are committed (and thus final) as soon as the consensus ends and the superblock is executed and appended to the chain. This differs from classic blockchains [36], [27] whose consensus is reached after the block is appended and after more "block confirmations" occur. Interestingly, the latency increase does not prevent the throughput of SRBB-dec from scaling as seen by Fig. 8. This is due to the superblock optimization: As more machines participate, more blocks get proposed and running consensus takes more time, which increases the latency, however, the number of transactions decided per consensus instance also increases, which provides scalability.

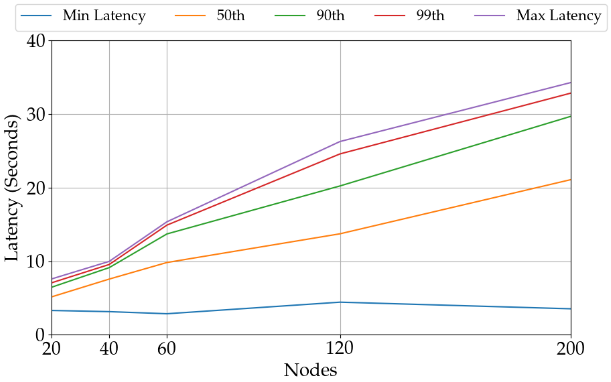


Fig. 9: The percentile latencies of SRBB-dec

VII. Discussion

A. Censorship of transactions

In classic blockchains (e.g., Ethereum) if a validator decides not to include a transaction in its new block, this transaction may eventually be included in another block by another validator, due to transactions being propagated to all validators. This prevents censorship. With SRBB, since there is no individual transaction propagation among validators, if a Byzantine validator decides not to include a transaction from a client in its new block, the transaction becomes censored. While introducing a load balancer [32] can mitigate this problem by increasing the probability of sending a transaction to a correct validator, note that it does not prevent SRBB to implement a Blockchain System (Def.1). As in other blockchains, the ideal solution to this censorship is to have clients resending their transaction to different validators if it does not get included in the blockchain.

B. Applicability of TVPR to other blockchains

In contrast to SRBB, implementing TVPR on modern blockchains can be problematic. Due to removing propagation of transactions among validators through TVPR, the first validator receiving a transaction should include it in a block for the transaction to be eventually executed. If the first validator receiving a transaction is weak, (e.g., has a low probability of creating blocks), it will rarely win the consensus protocol. Thus, a client can expect to wait a long time for their transaction to be included in a block. To prevent this drawback, clients may submit transactions to the most powerful validators in the hope of increasing the probability of their transactions being included in blocks sooner. This can centralize the blockchain towards a few validators and these few validators can be overloaded with transactions leading to a DoS.

In contrast, with SRBB, despite having TVPR, clients do not have to wait a long time for their transactions to be included in a block. This is because, in SRBB, all validators regardless of being weak or powerful can make block proposals per consensus round and combine their blocks to create a superblock [15]. More specifically, since SRBB uses the Redbelly consensus [15] (Section III-D) there is no single validator winning one consensus round. Every validator gets to include a block in the decided superblock per consensus round if every validator proposes a block. Thus, a transaction sent by a client to any validator will be included in the superblock in the same consensus iteration or the next (e.g., with 1000 validators, a client does not have to wait for 1000 iteration before its transaction is included in a block as all 1000 validators can propose blocks per consensus round and include their block in the decided superblock).

C. Removing eager validations completely

Several works like Hyperledger Fabric [7] and PRISM [34] completely remove eagerly validating transactions. However, unlike SRBB, these two blockchains do not remove the initial transaction propagation as in our TVPR solution. The complete removal of transaction eager validations increases the chances of DoS attacks on the blockchain through spam transactions consuming network bandwidth. This is because there is no validator initially validating the transactions, leaving the potential for spam transactions to be submitted to the system. In contrast, SRBB's validators eagerly validate transactions received directly from clients.

D. Fault tolerance of SRBB-dec

For SRBB-dec to achieve the same fault tolerance as SRBB, the consensus node and the SRBB VM node should be considered as belonging to the same participant (i.e., entity) such that $f < n/3$ participants are Byzantine where the total number of participants are n. A limitation with decoupling is if we consider the SRBB VM nodes and consensus nodes separately then the fault tolerance will be much lower than SRBB. This is because the consensus nodes alone as a separate entity can only tolerate a third of its nodes failing and with the addition of failures from SRBB VM nodes, SRBB-dec will have a lower fault tolerance (i.e., inability to recover from a little number of failures).

E. Additional resource usage

Decoupling can increase resource usage. With SRBB, a single user can use a single machine to join as a SRBB validator whereas with SRBB-dec a single user requires two machines to execute a consensus and a SRBB VM node separately.

F. SRBB vs SRBB-dec comparison fairness

Although in our evaluation we compare SRBB's performance to SRBB-dec, one may argue such a comparison is not fair. We used the same number of total machines when comparing the performance of SRBB and SRBB-dec to ensure that decoupling itself helps performance rather than an increase in the number of machines that increases the processing power. However, one might argue that to compare SRBB-dec to SRBB fairly, the consensus nodes in the SRBB-dec should equal the nodes of SRBB if consensus is the bottleneck. However, we see that SRBB-dec's consensus scales as the number of consensus nodes and SRBB VM nodes increase (Fig. 8). Thus, showing that consensus is not a bottleneck. It is true that comparing a blockchain to its decoupled counterpart can be unreasonable. As the saying goes, we want to compare apples with apples. However, our comparison is merely to show that SRBB decoupling can improve performance with the same resources.

G. Communication overhead in SRBB-dec

In SRBB-dec, the SRBB VM communicates with consensus nodes via gRPC. Unlike SRBB, this introduces a communication overhead between the two decoupled components. However, as observed by SRBB-dec's better performance over SRBB, the benefits of better management of resources offered through separation of concerns with the decoupled architecture outweighs the communication overhead of gRPC. This is because separation of concerns through decoupling allows (1) fine-tuning the execution implementation for CPU and consensus implementation for memory and (2) better parallelism between execution and consensus.

H. Adapting Decoupled Architecture to other Blockchains

The decoupled architecture presented in SRBB-dec splits the consensus and execution layers. Thus, the overarching idea of sending block proposals from the execution layer to consensus, and committed blocks from consensus to execution through RPC can be adapted to work with any blockchain, including PoW blockchains. The challenge, however, is the threat model. Once the consensus and execution are decoupled into separate nodes, new assumptions on the number of Byzantine nodes must be made to ensure state consistency.

VIII. Related Work

Classic blockchains. By design, Ethereum's EVM redundantly eagerly validates and propagates transactions. Every transaction is thus eagerly validated at every validator redundantly. These transactions are later propagated in blocks and via the network. Thus, transactions are redundantly propagated as well. Quorum [11], Binance Smart Chain (BSC) [2] and Cardano [3] blockchains all port the EVM as the state replication machine and thus have the same redundant eager validation and propagation of transactions problem.

Algorand [21] also suffers from the redundant eager validation and transaction propagation problem as it gossips transactions in the network and each transaction is eagerly validated at every validator. These transactions are redundantly propagated again in blocks. Similarly, Polkadot [19], Solana [37], and Tezos [1] suffer from the same problem despite introducing other optimizations in their state machine replica.

Avalanche [28], due to its snowman consensus protocol, does not propagate blocks but only transactions. Thus, it suffers from redundant eager validation of transactions but not from the redundant propagation of transactions.

Decoupled blockchains. Decoupling has been used in various forms in blockchains for varying purposes [34], [7]. For example, PRISM [34] decouples blocks into separate chains to improve performance of the longest chain rule. Hyperledger Fabric [7] decouples transaction ordering and execution using separate order nodes and peer nodes to order transactions facilitating different ordering services (i.e., consensus protocols) to be plugged into the system. Similar to Fabric, we decouple our consensus and execution to produce SRBB-dec. However, unlike Fabric the main motivation behind our decoupling is to enhance blockchain performance rather than to offer flexibility in using a customized consensus protocol. Fabric and SRBB-dec are inherently different in their architecture. More specifically, (1) Fabric follows a UTXO model whereas SRBB-dec follows a balance model (2) Fabric's client nodes submit transactions for execution and for consensus in two separate rounds whereas SRBB-dec simply submits transactions to the SRBB VM.

IX. Conclusion

In this paper, we deconstructed Smart Redbelly Blockchain (SRBB), the extension of the Redbelly Blockchain [15] to support arbitrary programs. This extension required the development of a complete new system that executes large blocks of smart contracts. We first removed the redundant transaction propagation, hence reducing the number of cryptographic verifications. We then optimised our solution by decoupling two roles played by blockchain validators, solving consensus and executing smart contracts, to obtain SRBB-dec. Our results show that SRBB outperforms Algorand, Avalanche, Diem, Ethereum, Quorum and Solana when deployed over 5 continents and under the realistic workloads of Nasdaq, Uber and Fifa using the Diablo [23] benchmark suite. Moreover, the additional decoupling optimization to produce SRBB-dec improves the peak throughput of SRBB for the Nasdaq workload by 33% and reduces its latency by 20%.

Acknowledgments

This work is supported by the ARC Future Fellowship funding #180100496 entitled "The Red Belly Blockchain".

References

[1] The lifecycle of an operation in tezos, 2019. Accessed on 2023-09-23, https://tinyurl.com/bde7wxv4.

[2] Binance smart chain, 2020. Accessed on 2023-09-23, https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md.

[3] Making the world work better for all, 2022. Accessed on 2023-09-23, https://cardano.org/.

[4] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman. Solida: A blockchain protocol based on reconfigurable byzantine consensus. Technical Report 1612.02916, arXiv, 2016.

[5] M. J. Amiri, D. Agrawal, and A. El Abbadi. Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems. In *ICDCS*, pages 1337–1347, 2019.

[6] Z. 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.

[7] E. Androulaki et al. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In *EuroSys*, pages 30:1–30:15, 2018.

[8] N. Bertrand, V. Gramoli, I. Konnov, M. Lazic, P. Tholoniat, and J. Widder. Holistic verification of blockchain consensus. In *DISC*, 2022.

[9] G. Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[10] B. Y. Chan and E. Shi. Streamlet: Textbook streamlined blockchains. In *AFT*, pages 1–11, 2020.

[11] J. Chase. Quorum whitepaper, 2019. Accessed on 2023-09-23.

[12] H. Chen and Y. Wang. SSChain: A full sharding protocol for public blockchain without data migration overhead. *PMC*, 59:101055, 2019.

[13] T. Crain, V. Gramoli, M. Larrea, and M. Raynal. DBFT: efficient leaderless Byzantine consensus and its application to blockchains. In *IEEE NCA*, pages 1–8, 2018.

[14] T. Crain, C. Natoli, and V. Gramoli. Evaluating the Red Belly Blockchain. Technical Report 1812.11747, arXiv, 2018.

[15] T. Crain, C. Natoli, and V. Gramoli. Red Belly: a secure, fair and scalable open blockchain. In *IEEE S&P*, pages 1501–1518, May 2021.

[16] Cryptokitties craze slows down transactions on ethereum, Dec. 2017. Accessed: 2020-11-14.

[17] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):pp.288–323, 1988.

[18] I've heard that on eBay there is an average of $680 worth of transactions every second. Is this true?, Feb. 2015. Accessed: 2020-11-14.

[19] J. B. et al. Overview of polkadot and its design considerations. Technical Report 2005.13456, arXiv, 2020.

[20] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, pages 281–310, 2015.

[21] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68, 2017.

[22] V. Gramoli. *Blockchain Scalability and its Foundations in Distributed Systems*. Springer, 2022.

[23] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron. Diablo: A benchmark suite for blockchains. In *EuroSys*, 2023.

[24] V. Gramoli and Q. Tang. The future of blockchain consensus. *Communications of the ACM*, 66(7), July 2023.

[25] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *SP*, pages 583–598, 2018.

[26] F. Leal, A. E. Chis, and H. González-Vélez. Performance evaluation of private ethereum networks. *SN Computer Science*, 1(285), 2020.

[27] S. Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008.

[28] T. Rocket. Snowflake to Avalanche: A novel metastable consensus protocol family for cryptocurrencies. Technical report, 2018.

[29] SRBB VM. https://github.com/deepalt92/ether-rbbc-standalone.

[30] D. Tennakoon and V. Gramoli. Blockchain proportional governance reconfiguration: Mitigating a governance oligarchy. In *CCGrid*, 2023.

[31] D. Tennakoon, Y. Hua, and V. Gramoli. Collachain: A BFT collaborative middleware for decentralized applications. Technical Report 2203.12323, arXiv, 2022.

[32] M. Toulouse, H. K. Dai, and Q. L. Nguyen. A consensus-based load-balancing algorithm for sharded blockchains. In *FDSE*, pages 239–259. Springer, 2021.

[33] G. Wang, S. Wang, V. Bagaria, D. Tse, and P. Viswanath. Prism removes consensus bottleneck for smart contracts. In *CVCBT*, pages 68–77, 2020.

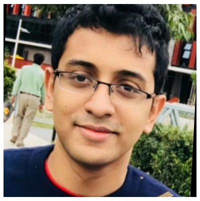
[34] G. Wang, S. Wang, V. Bagaria, D. Tse, and P. Viswanath. Prism removes consensus bottleneck for smart contracts. In *CVCBT*, pages 68–77, 2020.

[35] J. Wang and H. Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *NSDI*, pages 95–112, 2019.

[36] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Yellow paper*, 151(2014):1–32, 2014.

[37] A. Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*, 2018.

[38] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: Scaling blockchain via full sharding. In *CCS*, page 931–948, 2018.



Deepal Tennakoon. Deepal Tennakoon is a PhD graduate at the University of Sydney. He has published blockchain articles in multiple conferences including IPDPS, CCGrid and FAB. Deepal is an author of the article Smart Redbelly Blockchain published in IPDPS. He currently works as a Software Engineer (Blockchain) at iGreenData, A Synechron Company.

Vincent Gramoli. Vincent Gramoli is the Founder and CTO of Redbelly Network. He received a Future Fellowship from the Australian Research Council and leads the Concurrent Systems Research Group at the University of Sydney. In the past, Gramoli has been affiliated with INRIA, Cornell, CSIRO and EPFL. He teaches Blockchain Scalability on Coursera and his textbook is published by Springer.