Smart Redbelly Blockchain: Reducing Congestion for Web3

Deepal Tennakoon
University of Sydney
Sydney, Australia
dten6395@uni.sydney.edu.au

Yiding Hua
ETH Zurich
Zurich, Switzerland
yidhua@student.ethz.ch

Vincent Gramoli
University of Sydney
Sydney, Australia
vincent.gramoli@sydney.edu.au

***Abstract*— Decentralization promises to remedy the drawbacks of the web by executing decentralized applications (DApps) on blockchains. Unfortunately, modern blockchains cannot support realistic web application workloads mainly due to congestion.**

We introduce the Smart Redbelly Blockchain (SRBB), a provably correct permissionless blockchain that reduces congestion by (1) avoiding redundant propagation and validations of transactions with Transaction Validation and Propagation Reduction (TVPR) and (2) mitigating the propagation of invalid transactions within blocks by Byzantine nodes with a dedicated Reward-Penalty Mechanism (RPM). Our comparison of SRBB against Algorand, Avalanche, Diem, Ethereum, Quorum, and Solana, using the DIABLO benchmark suite, indicates that SRBB outperforms all these blockchains under real application workloads. Moreover, SRBB is the only blockchain to successfully execute real workloads of NASDAQ and Uber on a DApp without losing transactions. To demonstrate that TVPR and RPM are the causes of the improved performance, we compare SRBB with its naive baseline, which does not contain TVPR and RPM. Our results show that TVPR increases the throughput by 55$\times$ and divides the latency by 3.5, while RPM increases the throughput by 7% under flooding attacks. Finally, TVPR helps reduce transaction losses in the normal scenario while RPM goes further and mitigates transaction losses under flooding attacks.

***Index Terms*—Blockchain, Web3, performance**

I. INTRODUCTION

Decentralization promises to remedy some of the weaknesses of the web, including data exposures [1], user manipulations [2], and outages [3]. The idea, often called Web3, is to execute Decentralized Applications (DApps) on blockchains. Unfortunately, Web3 remains hypothetical as blockchains suffer from *congestion* when requests are received faster than they can be treated, leading to the saturation of their transaction queues. Congestion leads to transaction losses and performance degradation. Not only is this problem common to the oldest DApp-enabled blockchain, Ethereum [4], but also to one of the most recent and fastest blockchains, Solana [5]. Finally, a recent in-depth study demonstrated that, due to congestion, 6 modern blockchains lose transactions and degrade in performance when executing DApps under real application workloads [6]. Therefore, to decentralize the web, one has to first address this congestion problem in blockchains induced by realistic DApps workloads.

We identify two important causes of congestion in the modern blockchain design that leads to transaction losses and performance degradation. First, every transaction is propagated across the network of validators and validated at each validator leading to as many validations as there are validators. Thus, the transaction validation is redundant. After the validation of transactions, validators include transactions in blocks and propagate them again as part of a block across the network. Thus, transactions are propagated redundantly, first individually and then in blocks. As we see in Section V, this initial redundant validation and propagation of transactions cause congestion leading to transaction losses and performance degradation. Second, malicious (or *Byzantine*) validators can include invalid transactions in blocks and broadcast such blocks to the network of validators. This can spam the network with invalid transactions in what we coin as a *flooding attack* causing unnecessary consumption of node resources and network bandwidth, leading to congestion and in turn to transaction losses (Section V-B).

In this work, we present *Smart Redbelly Blockchain (SRBB)*, a provably correct permissionless blockchain that reduces the congestion, transaction losses, and performance degradation seen in modern blockchains due to the two aforementioned causes of congestion. To reduce congestion, SRBB introduces two novel contributions: (1) TVPR (Transaction Validation and Propagation Reduction) and (2) Reward-Penalty Mechanism (RPM). TVPR does not propagate transactions individually among nodes but only propagates transactions in blocks (Section IV), hence preventing the redundant validation and propagation of transactions without impacting the core blockchain properties of safety, liveness and validity (Section II-C). To prevent congestion induced by Byzantine validators performing flooding attacks with invalid transactions, SRBB introduces RPM. SRBB uses the superblock optimization [7] of the Red Belly Blockchain (RBBC) and the Democratic Byzantine Fault Tolerant (DBFT) consensus protocol [8] (hence its name) but unlike RBBC, SRBB supports smart contract execution.

To demonstrate that mitigating congestion is key, we show SRBB outperforms 6 modern blockchains under realistic DApp workloads (Section V). Moreover, SRBB is the only blockchain to not lose transactions under certain realistic DApp workloads. In summary, our contributions include:

- A permissionless blockchain, Smart Redbelly Blockchain (SRBB), that (i) prevents the redundant validation and propagation of transactions found in modern blockchains and (ii) mitigates the propagation of invalid transactions by malicious validators. SRBB uses TVPR for (i) and RPM for (ii). We prove that despite introducing TVPR and RPM, SRBB solves the blockchain problem, ensuring liveness, safety, and validity (Theorem 2).

- To better assess the advantage of TVPR and RPM, we compared SRBB with a baseline, which is a "naive" smart contract supported version of RBBC combining the DBFT consensus algorithm with the superblock optimization applied to the Ethereum Virtual Machine (EVM) but without our novel contributions of TVPR and RPM. Our results show that TVPR increases the throughput of the baseline by $55\times$ and divides the baseline latency by 3.5 (Section V-A). RPM increases the throughput of the baseline by 7% under flooding attacks (Table I). Lastly, both TVPR and RPM reduce transaction losses.
- To demonstrate the improvements of SRBB over modern blockchains in a common and fair setting, we used the DIABLO benchmark suite [6] with its recommended settings and real DApp workloads. We compared SRBB's performance against the reported performances of Algorand, Avalanche, Diem, Ethereum, Quorum and Solana. When deployed across the globe to execute a demanding FIFA web service workload on a DApp, SRBB commits twice as many transactions compared to the evaluated 6 modern blockchains in DIABLO. More interestingly, SRBB is the only blockchain out of the evaluated blockchains in DIABLO to commit all transactions for the application workloads of NASDAQ and Uber.

There may be highly demanding real workloads out of many untested workloads that SRBB may not support. Thus, we consider SRBB which encapsulates TVPR and RPM only as a stepping stone in the journey to reducing blockchain congestion for realizing Web3.

The remainder of this paper is structured as follows: Section II presents the background and Section III presents the problems in modern blockchains. Section IV presents SRBB (with TVPR and RPM) and proves it correct. Section V evaluates SRBB and Section VI discusses it. Finally, Section VII presents our related work and Section VIII concludes.

II. Background

A. Blockchain Background

Blockchain *nodes* are mainly *client* nodes or *validator* (i.e., miner) nodes. Client nodes send read and write requests to the blockchain. The term "client" is used to define implementations of Ethereum (e.g., Geth client – Ethereum's Golang implementation) by the Ethereum community but we identify a "client" solely as a sender of requests to the blockchain. Validators solve consensus to agree upon the order of client write requests, execute client requests, and services client read requests. Validators that follow the blockchain protocol (e.g., propose blocks with valid transactions, do not censor transactions) are *correct* while those that deviate are *Byzantine*. Clients have *accounts* in the blockchain that hold their digital assets. An account contains an address (the identifier of the account derived from the account holder's public key), and a balance (the amount of funds in the account). *Transactions* are write requests sent by clients to the blockchain and are of three main types: native payments that transfer funds between accounts, smart contract deployments that upload smart contracts to the blockchain, and smart contract invocations that invoke functions in uploaded smart contracts. A transaction has a sender and receiver address, and additional data (i.e., the amount of assets transferred if a native payment transaction or smart contract data if a smart contract transaction). Two transactions can conflict if they access the same data (e.g., smart contract variable) and one transaction is a write request [9]. A *block* is a batch of transactions.

In blockchains, 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. Validators finally append the executed blocks to a chain of blocks known as the *blockchain*.

B. Transaction Validation

To check that a transaction is valid, all validators in modern blockchains validate each transaction twice:

Eager validation: Eager validation occurs when a validator receives a transaction either from another validator or a client. It then verifies amongst other things whether the transaction is properly signed, the sender account has sufficient coins, and the transaction does not exceed a specified size limit. If eager validation of a transaction succeeds, the validator pushes the transaction to a pending queue in the transaction pool (this makes a transaction eligible to be included in a block by a validator) and propagates the valid transaction to downstream peers (i.e., validators connected to the local node that have not seen the transaction before), 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 nonce and the availability of gas (i.e., the cost required to execute a transaction). Thus, lazy validation is less time-consuming than eager validation. This lazy validation is necessary to guarantee that transactions in a newly received block are indeed valid prior to execution.

In Section III, we describe why eager validations are excessive in modern blockchains.

C. The Blockchain problem

We refer to the blockchain problem as the problem of ensuring the safety and liveness properties of blockchains taken from the definition by Garay et al. [10] and the classic validity property [7].

Definition 1 (The Blockchain Problem): The *blockchain problem* is to ensure that a distributed set of validators 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. When the chain is identical at two validators, then the state of these two validators generated deterministically from the blocks in the chain is identical.

III. Problems in modern blockchains

In this section, we define the redundant eager validation and propagation of transactions problem, and the invalid propagation of transactions problem in modern blockchains. These problems cause congestion and thereby transaction losses and performance degradation.

A. Redundant eager validation and transaction propagation

Many modern blockchains [11], [12], [13], [14], [15], [16] follow a protocol where validators first eagerly validate transactions received from clients or peer validators (Alg. 1, line 5), add these transactions to their transaction pool pending queue if valid (Alg. 1, line 7) and propagate each valid transaction individually to the network of validators (Alg. 1, line 9). Thus, every transaction in the blockchain initially gets eagerly validated at every validator node (Alg. 1, line 5) and propagated individually throughout the blockchain network of validators (Alg. 1, line 9). The transactions in the transaction pool pending queue of validators get included in blocks and propagated again as part of blocks (Alg. 1, lines 11- 13).

The initial propagation of individual transactions among validators is redundant and unnecessary since transactions are propagated in blocks. All valid transactions either end up in decided blocks and are executed (Alg. 1, line 21) or are pushed to the transaction pool pending queue at a validator from a received valid block, to be included in future blocks (Alg. 1, line 31). Thus, all valid transactions are propagated to every validator through blocks and eventually executed regardless of the initial transaction propagation. The eager validation of transactions at every validator except the first validator receiving the transaction is also redundant and unnecessary, as transactions are lazily validated prior to execution. Even if lazy validation succeeds for an invalid transaction due to lazy validation being weaker than eager validation, the transaction will fail at execution time by throwing an error (Alg. 1, line 36). Thus, the invalid transaction will have no impact on the blockchain state. In fact, we prove SRBB preserves liveness, safety, and validity without the initial propagation and

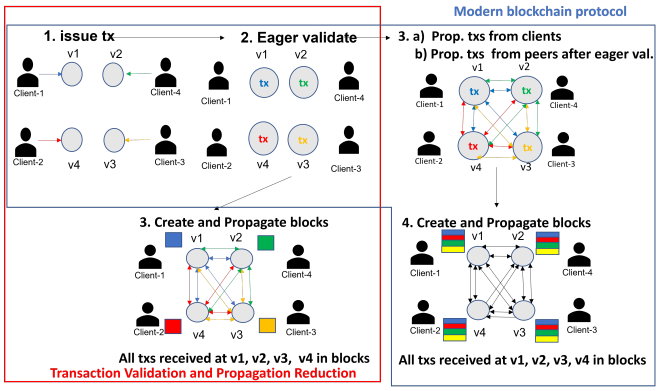
Modern blockchain protocol
1. issue tx
2. Eager validate
3. a) Prop. txs from clients
b) Prop. txs from peers after eager val.
v1
v2
v3
v4
tx
Client-1
Client-2
Client-3
Client-4
3. Create and Propagate blocks
4. Create and Propagate blocks
All txs received at v1, v2, v3, v4 in blocks
Transaction Validation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

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

eager validation of transactions (Section II-C). Due to unnecessarily eager validating and propagating transactions, we term this problem as the redundant eager validation and propagation of transactions. In our evaluation (Section V), we show the congestion impacts of this problem through transaction losses and performance (i.e., throughput and latency) degradation in modern blockchains under realistic DApp workloads.

B. Invalid transaction propagation

The invalid transaction propagation problem occurs when Byzantine validators propagate invalid transactions in blocks. This can happen when Byzantine validators falsely eagerly validate or skip the eager validation of transactions received from clients and peers but include such invalid transactions in blocks nonetheless and propagate them to the network of validators. Propagating invalid transactions in blocks does not cost anything to the validator but can cause transaction losses and performance degradation in modern blockchains due to the following reasons: (1) validators use extra CPU cycles eagerly validating invalid transactions that are dropped without contributing to the throughput and (2) the network bandwidth is consumed unnecessarily.

While mechanisms to ban validators that propagate invalid transactions can be implemented, such methods can be ineffective. For example, it is unclear whether banning validators is effective. Since banning validators include blocking a validator's address, a banned validator can easily derive a new wallet address and participate in the blockchain again [17].

Therefore, modern blockchains do not offer banning mechanisms in their implementation [11], [13], [18], [19], [14]. There are however, frameworks like flashbots[1] that can integrate validator bans in blockchains. Flashbots is known to be centralized [20]. This was also revealed by its latest Tornado Cash censorship[2].

In Section V-B, we notice transaction losses and performance degradation in modern blockchains caused by congestion when Byzantine validators propagate invalid transac-

[1]https://github.com/flashbots/block-validation-geth
[2]https://time.com/6223034/ethereum-merge-sanctions-flashbots/

tions. Thus, to mitigate Byzantine validators proposing invalid transactions, in Section IV-F, we present a novel RPM that slashes validators for propagating invalid transactions in blocks and rewards validators for proposing valid blocks. Unlike conventional validator banning methods like flashbots[3], RPM is decentralized and does not suffer from censorship issues. This is because RPM punishes validators for including invalid transactions in blocks using a decentralized smart contract-based method. While in conventional validator banning methods, validators can derive new wallet address and rejoin the blockchain, the slashing of stake integrated to RPM disincentivizes validators to rejoin the blockchain with new wallet addresses and re-propagate invalid transactions.

IV. Solution: Smart Redbelly Blockchain

In this section, we present SRBB, a permissionless blockchain that (1) prevents the redundant validation and propagation of transactions and (2) mitigates the propagation of invalid transactions. SRBB integrates TVPR to prevent the redundant validation and propagation of transactions, and a novel reward-penalty protocol to mitigate the propagation of invalid 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 TVPR which is a part of SRBB and prevents the redundant eager validation and propagation of transactions problem in modern blockchains. Second, we present the transaction life cycle of SRBB followed by the SRBB Virtual Machine (VM) implementation. Third, we discuss the membership and committee reconfiguration of SRBB followed by the novel reward mechanism of SRBB coined RPM that mitigates the invalid transactions propagation problem in modern blockchains. Finally, we prove SRBB solves the Blockchain Problem (Definition 1).

A. Assumptions

Our network consists of a set of validators V that are well-connected. As consensus cannot be solved in the asynchronous setting, we assume *partially synchronous*[4] communication [21] and out of n SRBB validators, at most f are *Byzantine* where $f < n/3$ (consensus is unsolvable in this model if $f \geq n/3$). Byzantine validators can act arbitrarily like proposing conflicting blocks and invalid transactions. Several blockchains [22], [23], [24] reconfigure their committee of n validators every epoch (i.e., a pre-specified unit of time) to mitigate 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 [23]. As n validators cannot reach consensus if $f \geq n/3$ validators are corrupt through bribery, the assumption of a slowly-adaptive adversary prevents consensus disagreements and double spending attacks. Therefore, we make the assumption that the adversary is slowly-adaptive similar to many prior works [22], [23].

B. TVPR (Transaction Validation and Propagation Reduction)

In TVPR, instead of validators eagerly validating and propagating transactions received from peer validators individually, validators only eagerly validate transactions received directly from clients. These validated transactions are then included in blocks and propagated. In other words, we get rid of step (3) in Figure 1 of the modern blockchain protocol where each validator individually propagates transactions to peers to produce TVPR outlined as Transaction Validation and Propagation Reduction. When considering Alg. 1, we remove Alg. 1, line 9 to prevent validators from propagating transactions (i.e., broadcasting transactions) individually to peers. This way, validators do not require to eagerly validate and propagate transactions received from peer validators. To be more precise, in modern blockchains during transaction propagation if the number of validators is n, a transaction t is eagerly validated n times, whereas TVPR eagerly validates a transaction t once (i.e., only the validator receiving the transaction directly from the client eagerly validates it). Thus, we remove the redundant eager validation and propagation of transactions. Note that transactions are still included in blocks and propagated. In Section IV-D we discuss in detail how reducing the eager validation of transactions does not cause the execution of invalid transactions. In Section V-A, we present the throughput improvements and latency and transaction losses reductions of integrating TVPR to SRBB compared to modern blockchains. The potential drawbacks of TVPR are deferred to the discussion section (Section VI).

C. 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 [8] combined with the superblock optimization of RBBC [7]. When describing the transaction life cycle of SRBB, we do not dwell deeply on the consensus as it is not our contribution. Instead, we give sufficient information to understand SRBB in its entirety highlighting our novelties. We refer the reader to [8], [7] for more details on the consensus protocol.

A transaction submitted by a client to SRBB goes through the stages below:

1. 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 5). If the eager validation fails, the transaction is discarded. Otherwise, the transactions are kept in a pending queue in the transaction pool (Alg. 1, line 7). Unlike in modern blockchains where transactions are propagated individually to all validators (i.e.,

[3]https://time.com/6223034/ethereum-merge-sanctions-flashbots/

[4]There exists an unknown Global Stabilization Time (GST) and a known positive duration δ such that message delays are bounded by δ after GST. In practice, we cope with partially synchronous communication by increasing timeouts [7].

Algorithm 1 Smart Redbelly Blockchain protocol

```
1:  State:
2:    $blockchain$, an array of blocks, initially
3:      $blockchain[0]$ = genesis-block

4:  receive($t$): #$t$ received from neighbors or directly from clients
5:    if eager-validate($t$) then #if eager validation succeeds
6:      if $t \notin blockchain$ and $t \notin p$ then
7:        $p \leftarrow p \cup t$ #add to txpool pending queue
8:        if TTL of $t$ not exceeded then
9:          propagate($t$)  #modern blockchains do this but SRBB does not

10: propose($p$):
11:   $b_i \leftarrow$ create-block-with($p1$) #create a $b$ from $p1 \subset p$
12:   $p \leftarrow p - p1$ #remove txs used to create $b$ from txpool
13:   propagate($b_i$)  #block propagation

14: Upon reception of $\mathbb{B}$ for index $k$ s.t. $\mathbb{B} \leftarrow \bigcup_{i=1}^{n} b_i$:
15:   for all invalid $b_i \in \mathbb{B}$  do
16:     $\mathbb{B} \leftarrow \mathbb{B} - b_i$ #discard blocks with invalid headers
17:   DBFT($\mathbb{B}$) #execute DBFT cons.
18:   if decide($\mathbb{B}^*$) then #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:           discard(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:   $\mathbb{C} \leftarrow \mathbb{B} - \mathbb{B}^*$ #undecided set of blocks
28:   for all $b_i \in \mathbb{C}$ do
29:     for $t \in b_i$ and $t \notin blockchain$ and $t \notin p$ do
30:       if eager-validate($t$) then
31:         $p \leftarrow p \cup t$ #add $t$ to txpool queue to be included in a future block

32: execute($t$):
33:   $err \leftarrow$ lazy-validate($t$) #lazy validation
34:   if $err \neq$ null then
35:     return $err$
36:   $S_r, err \leftarrow$ ApplyTransaction($t, S_i$) #Apply $t$ on state $S_i$
37:   if $err \neq$ null then
38:     return $err$
39:   else
40:     return null
```

Alg. 1, line 9), SRBB simply includes transactions in blocks and propagates blocks to the network, hence implementing our TVPR solution. In fact, SRBB is, as far as we know, the only blockchain to build upon TVPR. For index k of the blockchain every correct SRBB validator propagates a block b_i s.t. i is the ID of the SRBB validator and $i \in \mathbb{Z}^+$.

2. Consensus: An SRBB validator, after receiving a set of blocks $\mathbb{B}$ from peer validators for index k where $\mathbb{B} \leftarrow \bigcup_{i=1}^{n} b_i$, discards blocks that contain invalid headers from $\mathbb{B}$, executes the DBFT consensus [8] (Alg 1, line 17) on $\mathbb{B}$ and decides a superblock $\mathbb{B}^*$ at Alg. 1, line 18. Subsequently, the SRBB validator sends the superblock to the SRBB VM for execution (Alg. 1, lines 19-26). An SRBB validator adds transactions from the set of blocks received from validator peers that are not decided by DBFT consensus [8] (i.e., $\mathbb{C}$) to their transaction pool to be included in future blocks (Alg. 1, line 31). We refer the reader to DBFT [8] and RBBC [7] for more details on the consensus and the superblock respectively. Note that the only similarity between SRBB and RBBC is this consensus phase. RBBC does not support the execution of smart contracts or DApps and does not have TVPR and RPM.

3. Commit: An SRBB VM, upon receiving the superblock $\mathbb{B}*$, takes a block b_i at a time from the $\mathbb{B}*$, 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 lazy validates the transaction (Alg. 1, line 33). If a transaction's lazy validation succeeds, the SRBB VM attempts to apply the transaction to the current blockchain state (Alg. 1, line 36). 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 state (Alg. 1, line 38). If either the lazy validation fails or applying the transaction fails that means the transaction is invalid. The SRBB VM in this scenario discards the invalid transaction from the block b_i (Alg. 1, line 23) and moves on to the next transaction in the block. Subsequently, 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.

D. *SRBB VM Implementation*

The SRBB VM is ported from Geth and changed to prevent redundant eager validation and propagation of transactions by integrating TVPR. 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 and propagates them to the network. One might think that SRBB allows the execution of invalid transactions because a transaction is eagerly validated only once by a SRBB validator and then lazily validated at each SRBB validator prior to execution, where 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[5] which SRBB builds upon, raises an ErrInvalidSig exception if (i) is not satisfied. Overflow and VM exceptions are raised if (ii) is not satisfied.[6]

[5] L635 of ethereum/go-ethereum/blob/master/core/types/transaction.go of commit c4a6621

[6] L187-219 of ethereum/go-ethereum/blob/master/core/vm/interpreter.go, and ethereum/go-ethereum/blob/master/core/vm/errors.go.

Implementing TVPR may sound trivial, but it involved changing the convoluted Geth implementation which required a deep dive into the implementation details of Ethereum and talking to Ethereum core developers. In total, we changed 161 LOC (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.

E. Membership and committee reconfiguration

SRBB is a permissionless blockchain. An SRBB node can either be a client that sends transactions and reads the state of the blockchain or a validator that participates in consensus, executes transactions, and keeps a full state of the ledger to service clients. To be a candidate validator (i.e., a candidate validator is an applicant to the validator position), an SRBB node must deposit some tokens (i.e., each validator must deposit a pre-defined sum of ether – a unit of cryptocurrency used in Ethereum) to a committee reconfiguration smart contract. Afterward, the committee reconfiguration protocol randomly selects a set of candidate validators as validators known as the committee. These validators are rotated periodically, mitigating the committee from being bribed by a slowly-adaptive adversary. Each candidate is eventually selected as a validator because the selection is random and periodic. The requirement to deposit tokens to be a candidate validator provides a form of Sybil resistance making it costly for a single user to assume the identities of multiple candidate validators. Note that a high deposit may impact, in theory, transaction fees. This is because one needs to set transaction fees high enough to exceed the payout from a Sybil-attacking coalition that is willing to spend enough deposits to seize control of the protocol. To alleviate this stress on the transaction fees, the validator deposit is recoverable after a locked period like in the design of PoS protocols [25].

F. Reward-penalty mechanism (RPM) for SRBB validators

For a network of n SRBB validators, we assumed previously that there are more than $2n/3$ correct validators (Section IV-A). In the real world, however, validators behave rationally: they may behave selfishly and try to maximize rewards instead of blindly following the protocol. Thus, a reward-penalty mechanism (RPM) incentivizing correct behaviors while penalizing Byzantine behaviors is essential to ensure validators remain correct and continue to propose blocks.

With modern blockchains and with SRBB, the invalid transaction propagation problem (Section III) exists as Byzantine validators can propose invalid transactions in blocks that are propagated to the network. To cope with this problem we introduced RPM as a part of SRBB. In a rational validator setting, it makes sense for validators to increase their gains by bypassing eager validation and proposing invalid transactions in blocks to save validation costs. RPM incentivizes rational validators to not propose invalid transactions within blocks and propagate such blocks. As a result, RPM mitigates transaction losses and performance degradation (Table I). Like many blockchain reward mechanisms, RPM also rewards validators in a consensus round for proposing blocks. To the best of our knowledge, none of the reward and penalty mechanisms mitigate the propagation of invalid transactions within blocks [26], [27], [28], [29], [30].

Algorithm 2 The Reward-Penalty Mechanism

```
Initial State:
  For block $b$: $h_t$, $P_k$ and $S_k$ are the hash of its txs, the block sender pub and priv keys resp.
  $Cert_B \leftarrow \{P_k, (h_t)_{S_k}\}$ is the certificate of a block
  $T$ is the set of transactions in $b$ where $t \in T$
  $r_b$ is a constant block reward
  $c$ is the cost of eager validating a transaction
  $count$ is $(h \rightarrow val)$ where $count$ is a map between a hash and its count
  $N_B$ is the block number
  $i$ is the block index in the superblock and $round$ is the consensus round

propReceived($Cert_B, T, i, round$): #validators invoke when block decided
  if $invoked$[hash($i, round$)] == true then #already invok. for $b$ in $i$, $round$
    exit #exit function
  $invoked$[hash($i, round$)] = true
  $P_k, (h_t)_{S_k} \leftarrow$ retrieve($Cert_B$) #retrieve data from Cert
  $address_v \leftarrow$ derive($P_k$) #derive address of block proposer
  if $address_v \notin V$ then
    exit #invalid $Cert_B$, return function
  else
    $h_t \leftarrow P_k((h_t)_{S_k})$ #retrieve hash of transactions from Cert
    if hash(T) == $h_t$ then
      $count$[hash($P_k, T, i, r$)] $\leftarrow$ $count$[hash($P_k, T, i, r$)] + 1 #inc. count
      if $count$[hash($P_k, T, i, r$)] == $n$-$t$ then #thresh. decided the same block
        $address \leftarrow$ derive($P_k$) #derive address of block proposer
        $I \leftarrow r_b$ #incentive
        $C \leftarrow |T| \cdot c$ #cost of eager validating transactions in $b$
        $R \leftarrow I - C$ #calculate reward
        $K[address] = K[address] + R$ #add reward to proposer deposit
        $count$[hash($P_k, T$)] = 0 #reset count
report($Cert_B, N_B, t, T$): #Report validator Cert. block number and transaction
  $P_k, (h_t)_{S_k} \leftarrow$ retrieve($Cert_B$) #retrieve data from Cert
  $address \leftarrow$ derive($P_k$)
  $h_t \leftarrow P_k((h_t)_{S_k})$ #retrieve hash of transactions from Cert
  if $address_v \notin V$ or hash($T$) $\neq h_t$ or $t \notin T$ then
    exit #invalid $Cert_B$ or false report, return function
  else
    $count$[hash($P_k, N_B, t$)] = $count$[hash($P_k, N_B, t$)]+1 #inc. report count
    if $count$[hash($P_k, N_B, t$)] = $n$-$t$ then #thresh. val. reported invalid $t$
      $address \leftarrow$ derive($P_k$) #derive validator that added invalid-tx
      $K[address] = K[address] - P$ #reduce byz. validators deposit by $P$
      for all $v \in V$ and $v \neq address$ do #distribute penalty with validators
        $K[v] = K[v] + P/(|V| - 1)$
      emit $address$ #emit byz. validator event
```

Since we assumed a slowly-adaptive adversary previously (Section IV-A) the formation of a Byzantine coalition is mitigated by reconfiguring the committee as it takes time for a slowly-adaptive adversary to form a Byzantine coalition. Note that we do not discuss in depth the avoidance of Byzantine coalitions in RPM as it is outside the scope of this work. Instead, we assume that the formation of a Byzantine coalition is mitigated by committee reconfiguration which prevents rationals from deciding on conflicting blocks and double-spending.

We now define the block proposal game followed by a novel RPM for SRBB to mitigate invalid transaction propagation, followed by a reward design for RPM based on game theory. Finally, we prove our RPM incentivizes rational validators to not propagate invalid transactions within blocks.

a) The block proposal game: We use game theory to model the strategies of SRBB validators. We define a game G

per consensus round as a tuple (V, S, U) where V is the set of players who are SRBB validators, S is the set of strategies followed by players and U is the pay-off (i.e., reward or penalty) for each strategy.

A validator could follow a correct strategy or a Byzantine strategy. We consider the correct strategy as a validator proposing valid blocks (i.e., blocks with valid transactions) by properly eagerly validating transactions before including them in blocks. A Byzantine strategy is when a validator proposes invalid blocks by propagating invalid transactions in blocks (e.g., not eagerly validating transactions to save costs). Our goal is to design rewards for strategies so that the best strategy for a rational validator to follow is the correct strategy.

b) Reward-Penalty Mechanism: We present our RPM in Alg. 2. Earlier, we assumed out of n validators, at most f are Byzantine where $f < n/3$ (Section IV-A). As the committee progresses we assume that validators behave rationally.

Upon deciding on a superblock, validators invoke a propReceived function for each block in the decided superblock parsing the block proposer's certificate $Cert_B$, the set of transactions T in the block, the index of the block in the superblock i and the consensus round r (Alg. 2, line 10). The certificate of the block proposer $Cert_B$ consists of the public key of the block proposer P_k and the signed hash of the transactions in the block $(h_t)_{S_k}$ where h_t is the hash of transactions and S_k is the private key of the block proposer. Thus, $Cert_B = \{P_k, (h_t)_{S_k}\}$. Alg. 2, line 12 exits if a validator invokes propReceived more than once for the same i and r (i.e., prevents duplicate invocations). One validator cannot parse the same $Cert_B$ and T more than once to the propReceived function either as there is a checker preventing this, although not included in Alg. 2 for brevity. For each invocation of propReceived by a distinct validator, RPM retrieves data from the certificate $Cert_B$ (Alg. 2, line 14) and checks the validity of $Cert_B$ by verifying whether the $address_v$ derived from P_k is in the validator set of addresses V (Alg. 2, line 16). If not, $Cert_B$ is invalid and proposed by a non-validator so the propReceived function exits (Alg. 2, line 34). Otherwise, RPM retrieves h_t from $(h_t)_{S_k}$ (Alg. 2, line 19) and checks if the hash of T (i.e., $\mathsf{hash(T)}$) is equal to h_t (Alg. 2, line 20), verifying whether the block proposer with P_k proposed a block with transactions T. If hashes are equal, Alg. 2 increments the propReceived invocation count for a block in a superblock for a particular P_k and T (Alg. 2, line 21). If at least $n - t$ validators have decided on a superblock with a block b that has the same T and P_k, RPM derives $address$ from P_k, calculates the reward from Alg 2, lines 24- 26 (i.e., Section IV-F presents the reward design in detail) and increases the deposit of the proposer of the block that is included in the decided superblock (Alg. 2, line 27). Finally, when the epoch ends, the tokens added to a validator's deposit exceeding D_v are funded to the validator's wallet address (i.e., not included here for brevity).

Invoking propReceived is in the best interest of a rational validator due to the following reason: If a validator v_1 eventually does not invoke propReceived for a decided block $b1$ proposed by validator v_2 then v_2 may also decide not to invoke propReceived for a decided block $b2$ proposed by v_1. This could result in both blocks proposed by v_1 and v_2 not reaching the $n-t$ threshold and receiving a reward accordingly (Alg. 2).

RPM penalizes rational validators that propose blocks including invalid transactions in the following way: upon noticing an invalid transaction in a block that is part of a decided superblock, validators become reporters invoking report in Alg. 2, line 29 parsing $Cert_B$ (i.e., certificate of the block proposer that contained the invalid transaction), N_B (i.e., the block number containing the invalid transaction), t (i.e., the invalid transaction), and T (i.e., the set of transactions T in N_B). Then the validity of $Cert_B$ is verified and the function exits if an invalidity is noticed (Alg 2, lines 30- 34) avoiding false reporting. Otherwise, a count is incremented corresponding to the number of validators that observed the invalid transaction t in block N_B proposed by the block proposer with public key P_k (Alg. 2, line 36). If at least $n - t$ validators have made the same report (Alg. 2, line 37), then the Byzantine validator that proposed a block with invalid transactions receives a penalty P s.t. $P = K[address]$, which is deducted from her deposit (Alg. 2, line 39) leaving the deposit at 0. The deducted penalty is then equally distributed among the other validators in the committee (Alg. 2, line 41). In RPM validators lose deposits only if they propose a block that includes invalid transactions (Alg. 2, line 39). Correct validators eagerly validate each transaction before inclusion in a block proposal. As such, correct validators do not lose their deposit. Finally, an event is emitted containing the wallet address of the Byzantine validator (Alg. 2, line 42). All correct validators listen to this event and exclude the Byzantine validator from future communications within the committee.

Alg 2 contains a block reward value r_b, an incentive I, an eager validation cost c, and a cumulative reward R for including a block in a superblock. The goal is to design these rewards in a way that the best strategy for a validator to follow is the correct strategy. Next, we present our reward design.

c) Reward design: Let us consider R as the cumulative reward for proposing a block, I as the incentive, C as the cost of eager validating all transactions in a block, P as the amount deducted in Ether from a validator's deposit if the validator follows a Byzantine strategy (i.e., the entire current deposit is taken out), r_b as a constant ether value issued to validators for proposing a block, and $\sum_{n=0}^{N_{tx}} Txfees$ as the total transaction fees in the proposed block (i.e., the transaction fee is not included in RPM in Alg. 2 as it is charged separately at the SRBB VM when transactions are executed). Then the two reward equations are: $R = I - C - P$ and $I = r_b + \sum_{\mathsf{n=0}}^{\mathsf{N_{tx}}} Txfees$. We do not include double spending rewards rational validators may gain as double spending is mitigated through committee reconfiguration.

d) RPM proofs: In this section, we prove that rational validators will not propose blocks with invalid transactions.

Theorem 1: Rational validators proposing invalid transactions in blocks gain a negative reward.

Proof: If a rational validator proposes invalid transactions in a block to minimize C and thereby increase R (i.e., $R = I - C$), other validators are incentivized to report the invalid transaction along with the proposing validator (Alg. 2 lines 29- 41). As a result, a penalty P is deducted from the reported validator's deposit, and the validator is removed from the committee (Alg. 2, lines 39 and 42). When a Byzantine validator's block with invalid transactions is decided, we know that they gain a reward of $I - C'$ where $C' < C$ (i.e., Byzantine validators skip eager validation to save cost). If the initial deposit of the Byzantine validator is D, now it becomes $D' = D + I - C'$. However, once reported, Byzantine validators lose P s.t. $P = D' = D + I - C'$. Thus, the Byzantine validator's deposit becomes $D_{end} = D + I - C' - P$ leading to $D_{end} = 0$. Thus, the Byzantine validator loses her entire deposit D. □

From Theorem 1, proposing invalid transactions results in a validator losing her entire deposit. Rational validators try to maximize their gains effectively. Therefore, RPM discourages proposing invalid transactions in blocks.

G. Proofs of correctness: SRBB

Theorem 2: SRBB solves the blockchain problem.

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

Liveness: As a result of removing line 9 of Alg. 1 in SRBB, transactions are no longer propagated individually to the network (i.e., among validator peers) and eagerly validated at each SRBB validator. However, correct validators still create valid blocks including valid transactions, and propagate them to peers (Alg. 1, line 13). Thus, every correct SRBB validator receives a set of blocks $\mathbb{B}$ propagated by correct SRBB validators at index k (Alg. 1, line 14). An SRBB VM decides a subset of these blocks $\mathbb{B}^*$ and stores the valid transactions in these decided blocks in the blockchain (Alg. 1, lines 17-25). The SRBB validator also stores valid transactions of all received but undecided blocks at index k in the transaction pool (Alg. 1, lines 27-31). These transactions are eventually re-included in a future decided block by correct SRBB validators (Alg. 1, line 11) 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.

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 17) 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 [8] (Alg. 1, line 17). 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.

Validity: Due to the consensus protocol, all correct SRBB validators decide on the same valid superblock $\mathbb{B}^*$ at index k (Alg. 1, line 18). This means each block b_i in $\mathbb{B}^*$ has valid headers. If each block b_i in the superblock $\mathbb{B}^*$ has a valid set of transactions, it is appended to the blockchain (Alg. 1, line 25). Thus, each block appended to the blockchain of each correct SRBB validator is a set of valid non-conflicting transactions. □

V. Empirical Evaluation

In this section, we present our evaluation of SRBB and its performance compared to 6 modern blockchains. We evaluated SRBB using the DIABLO blockchain benchmark suite [6] 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[7]. NASDAQ (peak request rate - 19800 TPS, avg. request rate - 168 TPS) uses a real trace of Apple, Amazon, Facebook, Microsoft, and Google stock trades executed on a DApp, Uber (peak request rate - 900 TPS, avg. request rate - 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 (i.e., 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 instances (8 vCPUs, 16 GB RAM – equivalent to a modern-day PC) as DIABLO [6].

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 [6]. 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 [6] for realistic DApp evaluation. Similar to the DIABLO DApp evaluations [6], all workloads were evaluated with one full experimental iteration.[8]

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

[7]https://github.com/lebdron/minion/tree/aec

[8]Our discussions and comparisons with the authors of DIABLO revealed multiple runs of the same DIABLO DApp workload experiments yield minimal statistical variance in the results due to a long experimental time of ∼5 minutes (i.e., DApp workloads send transactions for 2-3 minutes and blockchains typically processed these transactions for ∼5 minutes).

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, lower latency) 6 modern blockchains for the realistic DApp workloads of NASDAQ, Uber, and FIFA [6]. Moreover, SRBB was the only blockchain out of the evaluated 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 has reduced blockchain congestion.

In this section, first, we compare SRBB with modern blockchains (Section V-A) for the real DApp workloads of NASDAQ, Uber and FIFA from the DIABLO blockchain benchmarking suite [31]. Then we evaluate the benefits in transaction losses and performance of RPM in SRBB when Byzantine validators propagate invalid transactions.

A. Comparison with other blockchains

Figures 2 and 3 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 [6] for our evaluation. Evaluating all blockchains in existence against SRBB is not realistic. We used the 6 modern blockchains evaluated in DIABLO [6] for comparison against SRBB because DIABLO reported a thorough evaluation of these blockchains under realistic DApp workloads [6]. To make the comparison fair, we simply used the same configuration parameters and DApp workloads as used in DIABLO [6] 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 [6] or they crashed due to the high load.

Figure 2 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 2 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 166.61 TPS, 835.15 TPS, and 1819 TPS for the NASDAQ, Uber and FIFA workloads respectively, which are the highest average throughputs for all evaluated blockchains. Figure 3 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 s. These performances are much lower compared to their claimed performances [6]. This indicates a major performance degradation.

Most importantly, SRBB multiplies the average throughput by 55×, divides the latency by 3.5, and reduces transaction losses considerably compared to EVM+DBFT. Since the difference between SRBB and EVM+DBFT is TVPR, TVPR is responsible for better performance and reduction in transaction losses and not prior works from RBBC [8], [7].

B. Evaluation of performance with Byzantine validators

Here we present the performance of SRBB when a Byzantine validator propagates invalid transactions. We created invalid transactions by setting the balance of the transaction sender to 0 ETH. More specifically, we compare the average throughput and transaction losses of SRBB without RPM and SRBB with RPM. Thus, this evaluation shows the performance benefits RPM yields considering throughput and transaction losses. Due to budget constraints, we had to restrict this benchmark to a single AWS region and the smallest validator size a BFT blockchain can tolerate[9] (i.e., four validators). More specifically, we performed this benchmark on the Sydney AWS region with 3 correct and 1 Byzantine validator.

	#valid txs sent	#invalid txs sent	#Byzantine Validators	throughput (TPS)	#valid txs dropped
SRBB w/o RPM	20K	10K	1	3998.2 TPS	none
SRBB w/ RPM	20K	10K	1	4285.71 TPS	none

TABLE I: The average throughput and valid transaction drops of four SRBB validators where one is Byzantine.

For Table I we used a sending rate of 15000 TPS to stress test the blockchains. From Table I, it is clear that despite Byzantine validators propagating invalid transactions, SRBB did not drop any transactions. RPM integrated with SRBB showed the best performance by increasing the average throughput to a stunning 4285 TPS which was 7% higher than SRBB without RPM. This is because Byzantine validators who are also rational (i.e., try to maximize their reward) do not propagate invalid transactions in the presence of RPM as it

[9]Using many medium sized instances instead of the four c5.2xlarge AWS instances used in Table I does not overcome our budgetary constraints as medium sized AWS instances yield minimal savings compared to c5 instances.

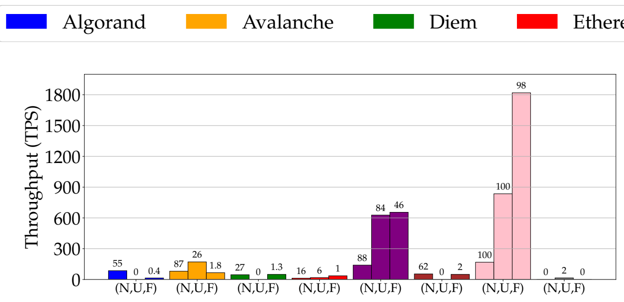

Fig. 2: Throughput (y-axis) and commit percentage (top of the bar) for NASDAQ, Uber and FIFA workloads (i.e., (N,U,F) is NASDAQ, Uber and FIFA)

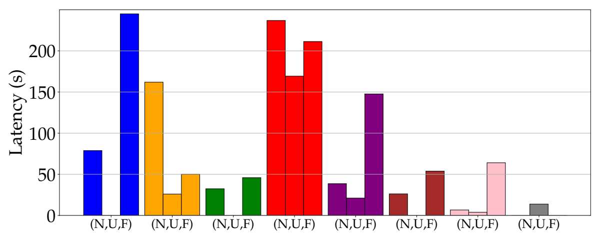

Fig. 3: Latency (y-axis) for NASDAQ, Uber and FIFA workloads (i.e., (N,U,F) is NASDAQ, Uber and FIFA)

deters propagation of invalid transactions by slashing stake. Thus, RPM helps improve performance.

VI. Discussion

In this section, we discuss the drawbacks of TVPR in SRBB and discuss how they could be addressed.

Censorship of transactions: In modern blockchains (e.g, Ethereum) if a validator decides not to include a transaction in its new block, the said transaction is likely to be included in another block by another validator eventually due to transactions being propagated to all validators initially. This prevents censorship. With SRBB since there is no individual transaction propagation among validators, if a validator decides not to include a transaction from a client in its new block, the transaction becomes censored (i.e., note we consider a validator that censors transactions as a Byzantine validator). One solution would be to let a distributed load balancer handle the censorship problem. A load balancer existing between clients and validators can randomly forward each transaction from a client to different SRBB validators to increase the probability of transaction inclusion in blocks. Even then, a transaction may be censored if the load balancer forwards the transaction to a validator that censors it. In this case, if the client does not receive a transaction receipt as proof of its execution within a period, the transaction can be resent by the client and could be forwarded to a different validator than before by the load balancer due to the randomness in forwarding. The new validator may not censor the transactions as the previous validator. If they also censor the transaction, this entire process can be repeated until a validator that does not censor the transaction receives it. This process can be automated to reduce the burden on the client. A Byzantine load balancer itself may be problematic. In such a case, a client may require to distribute transactions to multiple load balancers. We look at a few transaction load balancing and transaction distribution strategies among multiple load balancer to mitigate censorship in our future work [32].

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 low probability of creating blocks), they will rarely win the consenus 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 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 [7]. More specifically, since SRBB uses the Red Belly consensus [7] (Section IV-C.Consensus) 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).

VII. Related Work

In this section, we discuss modern blockchains with redundant eager validation and propagation of transactions that lead to their congestion under realistic DApp workloads.

Ethereum's EVM by design redundantly eagerly validates and propagates transactions. Ethereum validators propagate every received transaction individually to peer validators eventually propagating every transaction throughout the network. Thus, every transaction is eagerly validated at every validator redundantly. These transactions are later propagated as part of blocks as well throughout the network. Thus, transactions are redundantly propagated as well. Quorum, Binance Smart

Chain (BSC), Hyperledger Burrow, and Cardano [13], [19], [18], [33] blockchains all port the EVM as the state replication machine and thus have the same redundant eager validation and propagation of transactions problem.

Algorand [34] also suffers from the redundant eager validation and transaction propagation problem as it gossips transactions throughout the network where each transaction is eagerly validated at every validator. These transactions are redundantly propagated again in blocks. Similarly, Polkadot [15], Solana [14], and Tezos [30] suffer from the same problem despite introducing other optimizations in their state machine replica.

Avalanche [16] due to its snowman consensus protocol does not propagate blocks but only transactions. Thus, it only suffers from part of the problem: redundant eager validation of transactions. The propagation of transactions is not redundant as transactions are not propagated twice, once individually and then in blocks.

VIII. Conclusion

In this paper, we introduced SRBB, a provably correct permissionless blockchain to mitigate blockchain congestion. We demonstrated that reducing transaction validations and propagation in normal cases using TVPR and mitigating invalid transaction propagation under flooding attacks using RPM can lead to minimal transaction losses and significant performance improvements in SRBB. These improvements make SRBB perform significantly better than Algorand, Avalanche, Diem, Ethereum, Quorum, and Solana when executing DApps under real workloads. Our future work includes evaluating the methods of Section VI to mitigate transactions censorship.

Acknowledgments

This work is supported in part by the Australian Research Council Future Fellowship funding scheme (#180100496) and the Ethereum Foundation. We would like to thank Christopher Natoli and the anonymous reviewers for their feedback on earlier versions of this paper. Yiding Hua's work was done at the University of Sydney.

References

[1] M. Isaac and S. Frenkel, "Facebook security breach exposes accounts of 50 million users," Sept. 2018, accessed: 2023-02-24. [Online]. Available: https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html

[2] J. Prassl, *Humans as a Service: The Promise and Perils of Work in the Gig Economy*. Oxford Press, 2018.

[3] B. Provenzano, "Youtube is down for everyone right now [update: It's back]," Nov. 2020, accessed: 2020-11-14. [Online]. Available: https://www.msn.com/en-us/money/other/youtube-is-down-for-everyone-right-now-update-it-s-back/ar-BB1aVtNh

[4] "Eth network: So congested exchanges are forced to disable eth wallets," 2022, accessed: 2023-02-24 - https://www.newsbtc.com/news/ethereum-network-congested-exchanges-forced-disable-eth-wallets/.

[5] "Solana explains reasons behind the recent network slowdown," 2022, accessed: 2023-02-24 - https://tinyurl.com/5f6xjbtp.

[6] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron, "Diablo: A benchmark suite for blockchains," in *18th European Conference on Computer Systems (EuroSys)*, 2023.

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

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

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

[10] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *EUROCRYPT*, 2015, pp. 281–310.

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

[12] "Ethermint," accessed: 2020-11-14, https://github.com/cosmos/ethermint.

[13] J. Chase, "Quorum whitepaper," 2019. [Online]. Available: https://github.com/ConsenSys/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf

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

[15] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood, "Overview of polkadot and its design considerations," arXiv, Tech. Rep. 2005.13456, 2020.

[16] T. Rocket, "Snowflake to Avalanche: A novel metastable consensus protocol family for cryptocurrencies," Tech. Rep., 2018, accessed: 2021-12-01.

[17] L. Dobos, "USDC blacklist cost users an extra 3.6 million–per month," 2022.

[18] C. Kuhlman, B. Bollen, S. Davis, and D. Middleton, "Hyperledger burrow (formerly eris-db)," Mar. 2017, accessed: 2020-11-14, https://www.hyperledger.org/wp-content/uploads/2017/06/HIP_Burrowv2.pdf.

[19] "Binance smart chain," 2020, accessed on 2022-07-21, https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md.

[20] Y. Wang, P. Zuest, Y. Yao, Z. Lu, and R. Wattenhofer, "Impact and user perception of sandwich attacks in the defi ecosystem," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022.

[21] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. pp.288–323, 1988.

[22] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *IEEE S&P*, 2018, pp. 583–598.

[23] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *CCS*, 2018, p. 931–948.

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

[25] "Introduction to staking," 2022, accessed: 2022-12-22, https://wiki.polkadot.network/docs/learn-staking.

[26] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0.8.13," 2021, accessed: 2021-12-06, https://solana.com/solana-whitepaper.pdf.

[27] "Slashing," accessed: 2022-08-09, https://wiki.polkadot.network/docs/learn-staking#slashing.

[28] "Slashing," accessed: 2022-08-09, https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/.

[29] A. Ranchal-Pedrosa and V. Gramoli, "Trap: The bait of rational players to solve Byzantine consensus," in *ASIACCS*, 2022, p. 168–181.

[30] "The lifecycle of an operation in tezos," 2019, accessed on 2022-07-21, https://medium.com/everstake/how-does-slashing-work-in-tezos-and-why-is-it-important-to-delegate-only-to-reliable-bakers-like-a6c931e93c56#.

[31] "Diablo: Distributed analytical blockchain benchmark framework," 2020, accessed: 2022-04-12. [Online]. Available: https://github.com/NatoliChris/diablo-benchmark

[32] M. Toulouse, H. K. Dai, and Q. L. Nguyen, "A consensus-based load-balancing algorithm for sharded blockchains," in *Future Data and Security Engineering*. Springer, 2021, pp. 239–259.

[33] "Making the world work better for all," 2022, accessed on 2022-07-21, https://cardano.org/.

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