Chain-in-the-Box: Large-scale Blockchain Prototyping on a Single Machine

Suyan Qu, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau

University of Wisconsin-Madison
{suyan,dusseau,remzi}@cs.wisc.edu

Abstract. We present Chain-in-the-Box (CBox), an emulation-based prototyping and evaluation framework for blockchain systems. CBox modularizes blockchain systems using a staged event-driven model to facilitate prototyping and performance analysis. It enables large-scale testing on a single machine through various emulation techniques. We implemented four blockchain systems in CBox: Bitcoin, Diem, Aptos, and Avalanche. Our evaluation demonstrates that CBox can reliably emulate a reasonably large cluster on a 16-core machine. Finally, using CBox, we conduct a series of case studies that illustrate a fairness concern in Bitcoin, highlight the importance of having a rate-limiting mechanism and the preference cache to Avalanche, and quantify the benefit of separating block dissemination from consensus ordering. We also prototype Blizzard, a BFT protocol based on Avalanche, and discover that, while Blizzard offers better liveness in the presence of Byzantine validators, it fails to ensure safety during periods of network asynchrony.

1 Introduction

The rapid development cycle plays a pivotal role in achieving success. Blockchain systems, however, face long development cycles due to their inherent complexity. Blockchain systems comprise multiple independent processes that coordinate to maintain a shared state, where the actions of one process can trigger cascading effects and unpredictable outcomes across the others. They must also be robust against adversarial behavior, as participating processes may act arbitrarily or even maliciously to their own advantage. Public blockchains further complicate development by permitting open participation, resulting in networks with hundreds or even thousands of validators. Managing such large-scale deployments for experimentation is labor-intensive and can become economically infeasible, further complicating the development process.

To accelerate development cycles, we leverage two key insights in this work. First, despite the complexity and variability among different blockchain systems, transactions undergo the same sequence of stages before finalization. This uniformity allows us to employ a staged event-driven model [41] to describe most

blockchain systems. This approach not only facilitates prototype development but also enables reliable performance analysis, helping developers identify potential bottlenecks during evaluation. Second, blockchain systems rely on cryptographic primitives to ensure Byzantine safety guarantees. These primitives are computationally demanding, yet their execution time can be precisely estimated through simple mathematical models, and their outputs can be easily mocked. These characteristics make blockchain systems ideal candidates for emulation, through which we can significantly reduce the hardware resources required for each validator, thereby collocating the entire blockchain network onto one physical machine with minimal interference.

1.1 Main Contributions

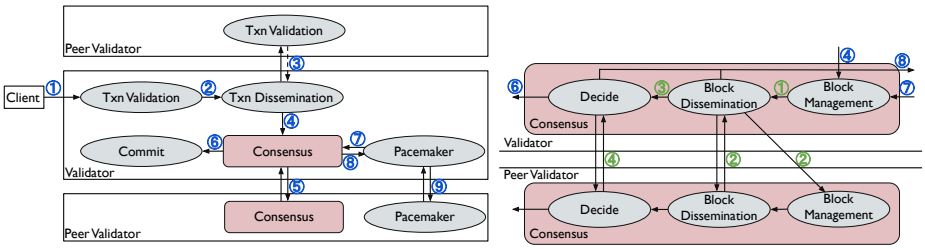
In this work, we introduce Chain-in-the-Box (CBox), an emulation-based framework designed for prototyping and evaluating blockchain systems. Within CBox, blockchain systems employ in a staged event-driven framework. This modular structure not only simplifies the prototyping process but also aids developers in understanding the system behavior. Additionally, CBox provides emulation-based implementations for commonly used cryptographic primitives, such as Merkle Trees and Signatures. These implementations reduce the resources each validator needs, permitting large-scale evaluation on a single machine.

We implemented Bitcoin [26], Avalanche [31], Diem [7], and Aptos [12] in CBox. Our evaluation demonstrates that, with a single 16-core machine, CBox can reliably emulate a cluster of validators: 200 Bitcoin validators, over 256 Diem validators, or 60 Aptos validators. In addition, using CBox, we conducted extensive case studies that quantify the effects of popular design choices or pinpoint problems that are frequently overlooked in the existing literature. Our study on Bitcoin reveals that, when the PoW task is not hard enough relative to block processing, Bitcoin shows fairness and even safety concerns if validators possess asymmetric compute power. Our experiments on Avalanche show that Avalanche requires mechanisms to keep all validators' progress at roughly the same pace to maintain its high throughput, and that having the preference cache can improve the performance of Avalanche by over $4\times$. We compare Diem against Aptos and demonstrate that disaggregating block dissemination from consensus ordering can improve Diem throughput by at least 50%. Finally, we implement a Blizzard [4] prototype within CBox. Our experiments indicate that while Blizzard enhances liveness against Byzantine validators compared to Avalanche, its consistency is compromised during periods of network partitioning, which may occur in partially synchronous networks.

2 CBox: Design and Implementation

CBox aims to provide a prototyping and evaluation environment to assist with the performance analysis of various blockchain systems. It has three goals:

– *Generic*: CBox should be generic to be adopted by various blockchain systems.



(a) Full picture of the model (b) Zoom-in view of the consensus module

Fig. 1: The staged event-driven model CBox adopts.

– *Comprehensible*: CBox should be able to provide insights on why the blockchain system behaves the way it does.
– *Lightweight*: Developers can experiment with a reasonably large blockchain network on a single physical machine.

2.1 Staged Event-Driven Model

Modularization is essential to comprehensibility. Rather than engaging with a monolithic blockchain system, breaking it down into smaller components can significantly improve understanding. CBox provides a staged event-driven model to guide the development of modularized blockchain systems. The model is grounded on our observation that almost all blockchain systems can be described with the same general stages. As we will show later, this model aligns seamlessly with an array of existing blockchain systems.

Figure 1a shows our model. After ① a transaction sent from the client reaches a correct validator, it undergoes validation and deduplication by the *Txn Validation* stage. ② Once validated, the transaction proceeds to the *Txn Dissemination* stage, where it is ③ disseminated to other validators in the network and then ④ passed to the *Consensus* module. Transactions are sent to the Txn Validation stage on peer validators, where they will similarly go through the validation, dissemination, and reach the consensus module. This process corresponds to the mempool management commonly found in blockchain systems. ⑤ Consensus modules of different validators collectively decide if the transaction should be committed. If so, ⑥ it is sent to the *Commit* stage where its effects are applied. We finally introduce a *Pacemaker* stage that ⑦ regulates block creation. It ⑧ collects statistics from the consensus module and ⑨ communicates with Pacemaker stages in other validators to make collective decisions.

Figure 1b provides a detailed view of the Consensus module. When a transaction reaches the Consensus module, ④ it first reaches the *Block Management* stage. This stage aggregates multiple transactions into blocks. Next, upon ⑦ receiving permissions from the Pacemaker stage, ① the Block Management stage emits the new block to the *Block Dissemination* stage, where ② it is propagated to other validators in the network. The Block Dissemination stages at the sending and receiving validators exchange messages to coordinate the dissemination process. ③ The block is then given to the *Decide* stage. The Block Management stages at peer validators will admit Blocks arriving at peer validators for vali-

Bitcoin [42]

Txn Validation (181 LoC): Verify transaction signatures and that transactions are valid (e.g., equal input and output UTXO).
Txn Dissemination (45 LoC): Gossip protocol.
Block Management (810 LoC): Constantly batch transactions and compute PoW to create blocks. When receiving a new block, verify that the block is well-formed and all transactions in the block are valid, then speculatively execute transactions to ensure that all input UTXOs exist and have not yet been consumed.
Block Dissemination (67 LoC): Gossip protocol.
Decide (161 LoC): Emit a block when there is a branch of length $k \geq 6$ extending from it. Send a signal to the Pacemaker stage whenever it receives a new block from upstream.
Commit (27 LoC): Persist speculative execution results.
Pacemaker: Periodically adjust PoW difficulty according to the rate at which the ledger grows recently and send the adjusted threshold to the block management stage. Because Bitcoin networks have static membership in our experiments for this work, we omit the Pacemaker stage for Bitcoin.

Diem [5]

Txn Validation (65 LoC): Verify transaction signatures and that transactions are valid (e.g., the transaction sender exists).
Txn Dissemination (45 LoC): Gossip protocol.
Block Management (665 LoC): Upon receiving a certificate from the Pacemaker stage, advance round. If the validator is the leader for the new round, batch transactions to propose a new block. Otherwise, wait for the block from the leader and speculatively execute transactions in the block to verify that sender accounts have enough balance to pay the gas fee.
Block Dissemination (66 LoC): The leader broadcasts its proposed block to all validators in the network.
Decide (402 LoC): Send a vote message to the leader of the next round if the proposed block is safe to vote. Upon receiving $N - f$ votes, aggregate them into a QC and pass it to the Pacemaker stage. Whenever it sees a valid QC, send the parent of the corresponding block to the Commit stage.
Commit (27 LoC): Persist speculative execution results.
Pacemaker (59 LoC): Keeps a timer for each round. If the timer expires before receiving a block, send a timeout message to the leader of the next round. When receiving $N - f$ timeout messages, aggregate them into a TC. If it builds a TC or receives a QC from the Decide stage, pass it to the Block Management stage. In this work, we focus on the correct path so validators don't keep timers.

Aptos [15]

Txn Validation (48 LoC): Verify transaction signatures and that transactions are valid (e.g., the transaction sender exists).
Txn Dissemination: Do nothing to simplify analysis.
Block Management (315 LoC): Batch transactions. When receiving a CoA list for the current round from the Pacemaker stage, advance round and pass a transaction batch with the CoA list to the Block Dissemination stage. When receiving a transaction batch, verify that it has at least $N - f$ CoA from the previous round before sending it downstream.
Block Dissemination (310 LoC): Broadcast batches created by self. If a batch is safe, broadcast a vote for this batch. Upon receiving $N - f$ votes from peers, aggregate votes to create a CoA. CoAs are passed to the Pacemaker stage and the Decide stage.
Decide (1.1k LoC): Store CoAs in Quorum Store and run DiemBFT to order CoAs. When DiemBFT commits a block, batches associated with CoAs in the block are added to the commit queue. If validators do not possess the corresponding batches, they fetch from peers. We focuses on the correct path in this work. Although the production Aptos chain has moved to Jolteon [34], DiemBFT deliver similar performance characteristics on correct paths [25].
Commit (112 LoC): Execute transactions.
Pacemaker (102 LoC): Collect CoAs and, upon receiving $N - f$ CoAs for a round, send these CoAs to the Block Management stage.

Avalanche [6]

Txn Validation (180 LoC): Verify transaction signatures and that transactions are valid (e.g., equal input and output UTXO).
Txn Dissemination: Do nothing (as in the Whitepaper).
Block Management (608 LoC): Batch transactions. When receiving a batch from peers, validate if each transaction is well-formed. For simplicity, validators do not add additional dependencies. We rely on the client workload generation to ensure progress.
Block Dissemination (71 LoC): For each batch created by self, multicast the batch to k randomly sampled peers in the network.
Decide (694 LoC): For each transaction in the batch, the validator checks if it and all its ancestors are preferred. The check results (preferences) are sent back to the querist. Upon receiving enough batched preferences or if a timeout happens, the validator checks for each transaction if at least αk validators prefer it and updates the conflict sets for this transaction and all its uncommitted ancestors. Transactions are emitted to the Commit stage if it is the last voted transaction in its conflict set, the counter in its conflict set exceeds β, and all its parents are committed.
Commit (27 LoC): Execute transactions.
Pacemaker: Do nothing.

Fig. 2: We implement the Bitcoin, Diem, Avalanche, and Aptos based on the CBox model. Most blockchain systems do not specify how or if transactions should be disseminated, so we use gossip protocol or disable the Txn Dissemination stage.

dation. If it is valid, it will similarly be sent to the Block Dissemination stage to continue the dissemination process and then passed to the Decide stage. ④ The Decide stages of validators in the network communicate with each other to collectively determine transactions within which blocks should be committed. If so, ⑥ these transactions are considered finalized and passed down to the Commit stage. ⑧ Finally, the consensus module communicates with the Pacemaker stage to provide statistics on the current status of the validator.

Mapping Blockchain Systems to the CBox Model Our model outlines the essential components of a blockchain system. For most blockchains, the Txn Validation stage, the Txn Dissemination stage, and the Commit stage naturally fit in, while decomposing the core consensus into three distinct stages and defining the responsibilities of the Pacemaker stage requires more consideration. Figure 2 illustrates how we implement four blockchain systems using the CBox model: Bitcoin [26], Diem [7], Aptos [12], and Avalanche [8]. These systems are chosen to cover a wide variety of blockchain designs with public vs private membership,

strong vs weak finality, partially synchronous vs asynchronous assumptions, etc. Because the model specifies the behavior of individual validators, blockchains where validators have asymmetric roles [13, 33] can also embrace this model.

Benefits The staged event-driven model provides two important benefits. First, each stage operates with its own executor and communicates with others via message queues, enabling transactions to be processed in a streaming fashion as they flow through the pipeline. As we will see in Section 3.3, this design makes it easy for developers to monitor the flow of transactions—specifically, how many enter and exit a given stage. When a stage becomes a bottleneck, it will emit fewer transactions than it receives. Second, the modular design promotes flexible component reuse and substitution across blockchain systems. Developers can easily swap out stage implementations or adapt existing ones to support new system architectures. For instance, in Figure 2, Bitcoin and Diem share the same Txn Dissemination stage.

2.2 Blockchain Network in One Machine

Studying the behavior of a blockchain network on a single machine requires consolidating all validators onto the same physical node. However, naively packing multiple validators together leads to significant resource contention, as they must share limited hardware resources. CBox addresses these hardware constraints by emulating the blockchain network. We choose emulation over simulation to provide a trade-off spectrum between the scale and the reliability of evaluation. More precisely, we leverage processing illusion (PIL) [36], an emulation technique that replaces function processing with sleep operations. As we apply PIL more aggressively to a larger set of function logics, the emulation will become less authentic; however, we will be able to launch more validators since they consume fewer resources. In this work, we apply PIL to cryptography primitives and transaction execution, while leaving the consensus logic untouched.

Processing Illusion (PIL) Applying PIL to a function logic requires careful modeling of its execution time. We apply PIL to cryptography primitives and transaction execution because these function logics are the most computationally demanding and often well-defined, allowing us to mathematically model their execution time based on the input. To derive an accurate PIL implementation, we first construct a mathematical model based on the definition of the cryptography primitive. For instance, signing a message of length n consists of hashing the message ($O(n)$) followed by encrypting the hash with the private key ($O(1)$). Accordingly, we model signing latency as a linear function of the input size. Next, we empirically measure the execution time across a range of inputs and fit these measurements to our model. During execution, the PIL implementation uses the fitted model to estimate the execution time for the given input and sleeps for the corresponding duration. CBox currently provides PIL implementations for four cryptography primitives: signatures, Merkle trees, threshold signatures, and Bitcoin's PoW. To support diverse workloads, CBox does not provide PIL for specific transactions but instead allows them to run for configurable durations.

eCores Since PIL implementations do not consume real CPU cycles, validators

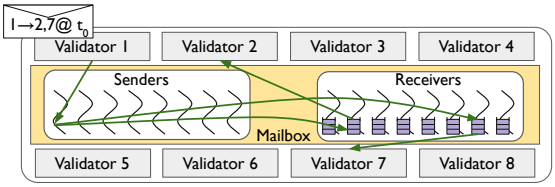


Fig. 3: The architecture of mailbox. Green arrows show how the mailbox handles the multicast request from Validator 1 to Validators 2 and 7. Each receiver thread maintains a priority queue (purple) of messages by their expected delivery times.

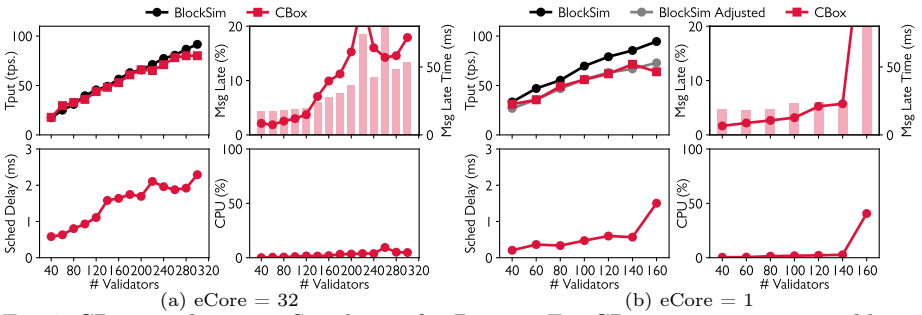
can execute many more PIL functions in parallel than they could in reality. To enforce constraints on compute power, we introduce the concept of *emulated cores* (eCores). Each validator is given a fixed number of eCores shared across all stages. To process an event, a validator must first acquire one of its eCores. If no eCores are free, it will wait until an ongoing event completes and releases its eCore. With more pending events, the validator will experience longer delays waiting for eCores to become available—similar to how validators experience resource contention under heavy workloads.

2.3 Implementation

We implement CBox in 7.6k lines of Rust. Our implementation of CBox embraces *Single-Process Cluster* (SPC) [36], where all validators run within the same process. This architecture offers several key benefits. First, SPC avoids costly kernel context switches by enabling user-level scheduling via Tokio. Second, Tokio facilitates quantifying resource contention among validators by recording scheduling delays—the time between when a task is awakened and when it actually begins execution. In our stage-event driven model, prolonged scheduling delays indicate increased contention: as more validators are added, each validator will experience an elongated scheduling delay as they compete for the limited resources. CBox logs these delays during experiments to assess the reliability of the emulation results. Third, because all validators share the same memory space in SPC, gathering cross-validator statistics for fine-grained analysis is straightforward.

The effort required to decompose blockchains to CBox's SEDA model varies across designs and implementations. To simplify this process, CBox assigns one user-level thread to each stage within each validator. Since events handled by the same stage often touch shared data, this design avoids complicated concurrency control. Although this design inevitably limits parallelism and may introduce additional queueing delay not captured by Tokio, this delay is negligible if event handlers are short-running. For long-running handlers, CBox can launch them to new user-level threads, provided that the implementation takes care of the necessary concurrency control.

Validators communicate with each other through a message exchange layer called the *mailbox*. In addition to basic message passing, the mailbox injects artificial delays to emulate network interface cards (NICs) with limited bandwidth connected by a wide-area network (WAN). For every message sent, the mailbox records the message sent time, computes its intended delivery time, and delivers it accordingly. Figure 3 shows the mailbox's architecture.



(a) eCore = 32 (b) eCore = 1

Fig. 4: CBox emulation vs Simulation for Bitcoin. For CBox experiments, in addition to the experiment results (top-left graph), we report metrics to assess the quality of emulation. The bottom-left graph presents the scheduling delay. In the top-right graph, the lines show the fraction of messages delivered late, and the bar graphs indicate how late these messages are. The bottom-right graph is the emulation CPU utilization.

3 Evaluation

In this section, we try to answer the following questions:

- Can CBox faithfully reflect the impacts of system configuration, runtime environment, and client workload on the performance of blockchain systems?
- How large a blockchain network can CBox emulate on a single physical machine? What factors limit the scale of the network CBox can emulate?

We evaluate CBox with the four blockchain systems listed in Figure 2. For Bitcoin, we compare our results with mathematical models. For Diem and Aptos, due to the lack of reliable data sources, we run multiple experiments with different experiment setups and collect additional metrics that illustrate how system designs lead to the performance we observe. For Avalanche, we validate our results against observations reported in the whitepaper [31]; due to hardware differences, we only focus on the relative performance trend.

All experiments are conducted on a c220g1 machine with two Intel E5-2630 v3 8-core CPUs @ 2.40 GHz and 128 GB memory on CloudLab. Unless otherwise specified, we emulate a WAN network where each node is equipped with a 1 Gbps NIC and each pair of nodes is connected through network channels with 30 ms delay and 100 Mbps bandwidth. For each experiment, in addition to performance metrics, we also collect the scheduling delay that the nodes in CBox observe, the fraction of messages that mailbox delivers late by more than 10 ms, how late these messages are, and CPU utilization. Our experiment shows that these metrics are sufficient to demonstrate whether CBox delivers reliable results and, if not, why it fails to emulate with precision.

3.1 Bitcoin

We show that CBox can produce results comparable to simulation when emulation metrics show no anomalies, and it can achieve higher fidelity by accounting for validator resource constraints. We conduct two experiments on Bitcoin and compare our results with simulated outcomes in BlockSim [2]. For each experiment, we scale the network size and construct a random topology by connecting

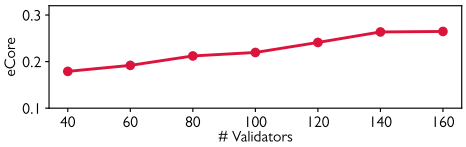


Fig. 5: eCore utilization on block processing. As the network grows, each validator must allocate more compute resources block processing, rather than proof-of-work.

each validator to five randomly selected peers.

In the first experiment, each validator is allocated 32 eCores, and the PoW threshold is set to 2^{220}. Figure 4a shows experiment results. For networks with up to 200 validators, the emulation metrics remain stable, and the throughput observed in CBox closely matches BlockSim's simulated results. We find that fewer than 16% of messages are delayed, with an average delay of less than 40 ms. Due to PIL, CPU utilization during emulation remains low throughout the experiment. This leads to high scheduling delays as Tokio executors must wait to be scheduled by the host operating system before running. Fortunately, because of PoW, the scheduling delays are amortized in Bitcoin. At 220 validators, however, we observe a sharp increase in both the frequency and magnitude of message delays, suggesting that the mailbox struggles under the growing message load induced by the gossip protocol.

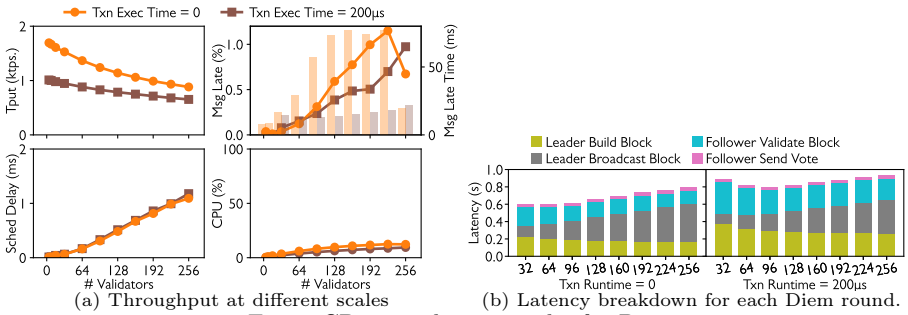
To highlight the limitation of simulation, the second experiment reduces each validator's allocation to 1 eCore, and increases the PoW threshold to 2^{226}. In practice, validators must dedicate compute resources to tasks such as executing blocks, limiting the *hash power* available for PoW. This nuance is difficult to capture in simulation, as estimating hash power through theoretical models is nontrivial. In contrast, CBox directly tracks eCore utilization and adjusts each validator's hash power accordingly. Figure 4b shows that simulated throughput consistently overestimates performance across all network sizes, with the discrepancy widening as the network grows. As illustrated in Figure 5, over 15% of eCore time is consistently consumed by non-PoW tasks, and this share increases with network size due to more frequent block generation. When we adjust the validators' hash power based on the eCore utilization reported by CBox, BlockSim reports throughput that closely aligns with our emulation results.

3.2 Diem

We demonstrate that CBox can reliably emulate a Diem network with 256 validators on a 16-core machine, and developers can collect detailed statistics in CBox to analyze network behavior. We conduct experiments with two script execution times and record the throughput in Figure 6.

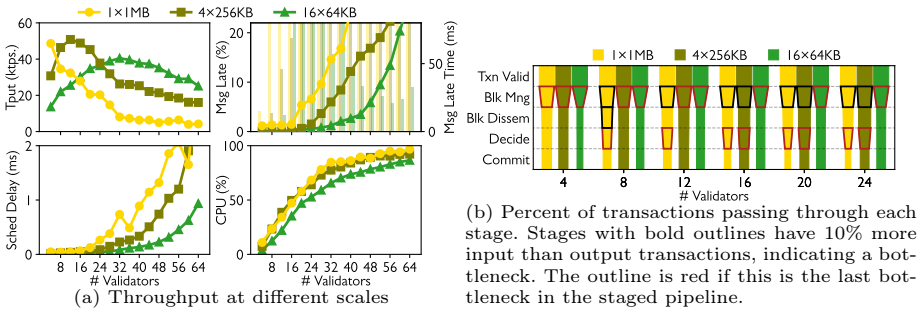
In both cases, Diem throughput is inversely proportional to the network size. To explain this behavior, we present the latency breakdown of each Diem round in Figure 6b. Importantly, we observe that the time for a proposed block to reach $\frac{2}{3}$ followers increases linearly with the network size due to congestion at the leader's egress NIC. Congestion does not occur when receiving votes because the vote messages are short and the ingress NIC consumption is negligible.

Additionally, Diem requires speculative execution of transactions by both



(a) Throughput at different scales (b) Latency breakdown for each Diem round.

Fig. 6: CBox emulation results for Diem.



(a) Throughput at different scales

(b) Percent of transactions passing through each stage. Stages with bold outlines have 10% more input than output transactions, indicating a bottleneck. The outline is red if this is the last bottleneck in the staged pipeline.

Fig. 7: CBox emulation results for Aptos.

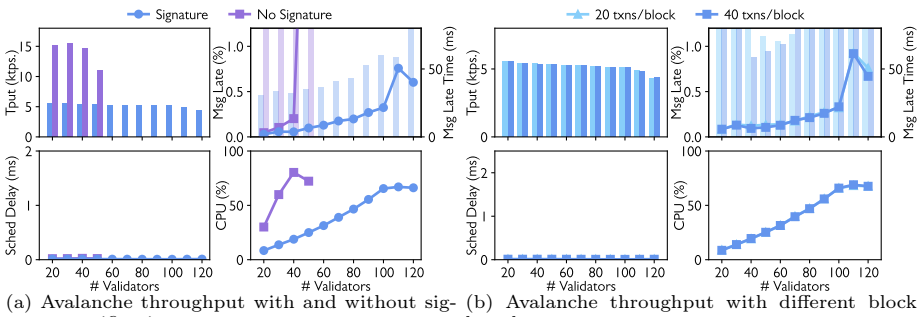
leaders (before proposing) and followers (before voting) on the critical path. Even if transactions involve little computation, changes to account balances due to gas fees will cause modifications to the state Merkle tree, elongating the block build and validation times. However, this cost remains constant across network sizes and is amortized in larger networks CBox accurately captures these behaviors.

Although the emulation result does not present any anomaly, we do not emulate Diem networks with more than 256 nodes. As a permissioned blockchain, a network of 256 validators is sufficiently large for most realistic use cases. CBox is particularly effective at emulating Diem because the consensus logic does not require complicated computation, and its 1-to-n communication pattern puts minimal strain on the mailbox.

3.3 Aptos

We show that we can emulate an Aptos network with various configurations in CBox and developers can take advantage of CBox's staged event-driven model for performance analysis. We run Aptos with different batch sizes and adjust the number of batches per Diem proposal to maintain a consistent 1 MB of information being proposed during each round. Figure 7 presents our results.

In our experiments, Aptos exhibits a unique scalability pattern: as the network grows, the throughput of the network first increases, then decreases. This special pattern stems from separating block dissemination from ordering. Figure 7b shows where the bottleneck is for each configuration. With a small cluster, the Block Dissemination stage is the major bottleneck. In Narwhal, each validator independently proposes new transaction batches, so its throughput increases



(a) Avalanche throughput with and without signature verification.

(b) Avalanche throughput with different block lengths.

Fig. 8: CBox emulation results for Avalanche.

if the network contains more validators. Since the Narwhal Block Dissemination stage is the bottleneck, the throughput of Aptos will increase with the network size. However, as the network size continues to grow, the major bottleneck gradually shifts to the Decide stage (which implements DiemBFT). When this happens, the throughput of Aptos decreases when the network size increases.

Unlike Diem, CBox struggles to emulate a large Aptos network. For all configurations, with a network of 64 validators, over 20% of messages are delivered late. In Narwhal, in every round, each validator proposes its own batch of transactions. Upon receiving a transaction batch, each validator broadcasts its vote to others. This leads to a communication complexity of $O(n^3)$ for each round, stressing the mailbox. In addition, we also observe a high CPU utilization with 64 validators. Hence, we refrain from emulating larger networks.

3.4 Avalanche

We show that CBox can emulate an Avalanche network, and developers can arrive at similar conclusions when experimenting with real machines or with CBox. In our experiment, transactions form chains: each transaction consumes one input UTXO and produces one output UTXO. Transactions do not conflict with each other. We adopt the same setup in the whitepaper: $k = 10, \alpha = 0.8, \beta_1 = 11, \beta_2 = 150$. Each validator has 2 eCores, and Each query batch contains 40 transactions. The clients maintain 160 accounts and keep 4000 transactions in flight. Because Avalanche is compute-intensive, we conduct scalability experiments on an m7i.16xLarge machine on AWS EC2. We run two experiments from the whitepaper at a smaller scale and show our emulation results in Figure 8.

In the first experiment, we run Avalanche with and without signature verification on transactions and present our results in Figure 8a. As claimed in the whitepaper, Avalanche delivers a constant throughput with and without signature verification. However, Avalanche's compute-intensive nature means that CBox cannot emulate a large Avalanche network. Without signature verification, emulating a cluster of 40 validators leads to over 80% CPU utilization, prohibiting emulation at larger scales. With signature verification, CBox can emulate over 100 validators without incurring high CPU utilization. However, because Avalanche validators need to wait for the preference from at least α of validators it queries for each batch (in contrast to $\frac{2}{3}$ of validators as in Diem and Aptos),

its performance is more sensitive to the ratio of messages being late. With 110 validators, even though less than 1%, the spike in the percentage of messages delivered late still leads to a noticeable impact on the throughput.

The second experiment evaluates Avalanche's throughput with different query batch sizes. Figure 8b shows our result. In this experiment, CBox can reliably emulate a cluster of 100 validators. Before reaching these emulation limits, Avalanche maintains nearly identical throughput for both block lengths, confirming the findings reported in the whitepaper.

While CBox fails to emulate 2000 machines as in the whitepaper, it offers a lightweight, cost-effective alternative better suited for development and iteration.

4 Insights from Case Studies

In this section, we present several case studies to demonstrate CBox's capability in studying blockchain systems. Unless otherwise specified, we use the same hardware and experiment setups as in Section 3.

4.1 Bitcoin: compute resources are not equal

Bitcoin's safety assumes that a validator's chance of mining a block is proportional to its share of the network's total compute power. However, non-PoW tasks, such as block execution, consume resources. Validators have to reserve some compute power for non-PoW tasks rather than devoting all resources to PoW. This limitation on validators' processing capacity is overlooked in many state-of-the-art analysis models [23]. Since all validators perform similar non-PoW work, they reserve a similar amount of compute power. For validators with more resources, this reserved amount accounts for a smaller portion of their total compute power, giving them disproportionately stronger hash power to mine more blocks. This imbalance raises concerns about fairness and even safety.

In this experiment, we set the PoW threshold to 2^{228} and configure the client workload so that each transaction runs for 2 ms. The network consists of 39 small validators, each equipped with 1 eCore, and a powerful validator, Validator 0. We vary the number of eCores assigned to Validator 0 and record the percentage of blocks it creates on the final chain. Figure 9 plots the portion of compute resources, rewards received, and the hash power of Validator 0 relative to the entire network. In particular, the reward validator 0 receives aligns closer with its hash power than its compute resources. With 10 eCores, although Validator 0 constitutes only ~20% of the network's total compute resources, it produces around 33% of the blocks on the longest chain. This is sufficient to dominate the entire network [10]. This experiment reveals a potential flaw in Nakamoto consensus with PoW: if non-PoW tasks consume significant compute, safety may be violated. While this particular scenario is not practical in the current Bitcoin network, the same problem may surface if we launch more compute-intensive scripts involving zero-knowledge proofs or machine learning.

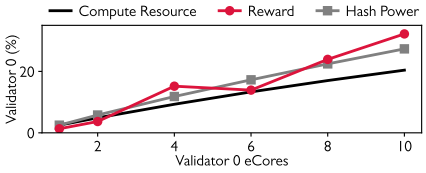


Fig. 9: Bitcoin with asymmetric compute resources. When validator 0 is assigned more cores, its hash power relative to the entire network increases disproportionately, yielding more rewards.

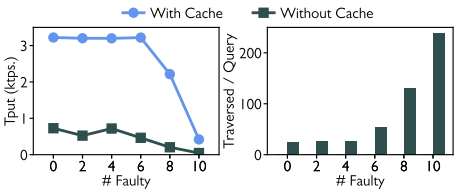


Fig. 10: Avalanche with and without preference cache. The figure on the right shows the average number of ancestors traversed for each queried transaction.

4.2 Avalanche: the preference cache

In this experiment, we analyze the benefits of the preference cache in Avalanche. In an Avalanche network, validators traverse all uncommitted ancestors of a transaction to decide if it should vote for or against this transaction. To reduce the cost of traversal, the whitepaper introduces a per-validator preference cache that keeps the preference of each transaction in the working set to avoid traversing all ancestors each time the same transaction or its descendants are queried. In this experiment, we introduce Byzantine validators that vote against all queries and do nothing else. With more faulty validators, query rounds are more likely to fail, in which transactions will be harder to commit, and new transactions will have more uncommitted ancestors. We establish a network with 40 nodes and increase the number of faulty actors in them. Figure 10 shows our result.

Since transactions are harder to commit when more validators are added to the network, Avalanche exhibits a steep performance drop with 8 or more faulty validators. We discuss the influence of faulty validators in more detail in Section 4.5. Our results show that Avalanche with the preference cache constantly outperforms Avalanche without the preference cache. With the preference cache, validators only need to traverse the queried transaction to fetch its preference from the cache in most cases. In contrast, validators without the preference cache need to traverse notably more ancestors. This cost increases exponentially as more faulty validators enter the network.

4.3 Avalanche: do you need a pacemaker?

In this experiment, we demonstrate that Avalanche cannot sustain a high throughput if there is no limit on the number of blocks validators can query concurrently. Instead of maintaining a constant number of in-flight client transactions, we fix the client request rate to 6000 transactions per second. We compare Avalanche with two configurations: the vanilla one without the Pacemaker stage, and one where the Pacemaker stage limits validators to 40 concurrent block queries.

Figure 11 shows our result. Without the Pacemaker, validators overload each other by issuing queries faster than peers can handle, effectively launching a distributed denial-of-service attack on one another, causing widespread timeouts and halted progress. Even removing timeouts doesn't help. For each query block sent, since the validators randomly sample k peers, they should also respond

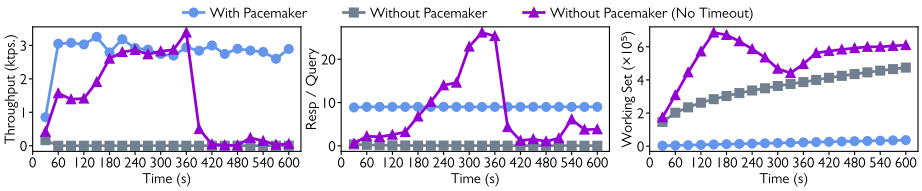


Fig. 11: Figure 12: Avalanche with and without the Pacemaker stage. The middle figure shows the number of query blocks a validator responds to for each query block it sends, and the right figure presents the number of transactions in the working set.

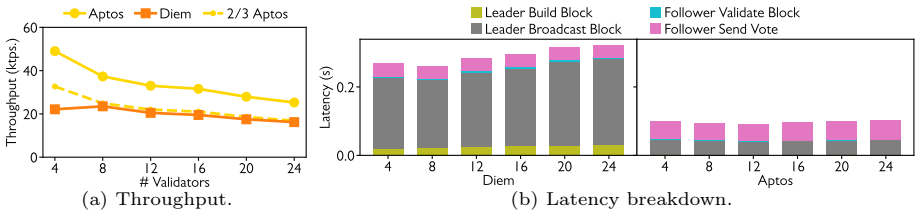
to k query blocks from peers to maintain stable throughput. However, when the Pacemaker stage is disabled, worker threads sending new query blocks and worker threads responding to received query blocks will compete for resources, destabilizing the system. In our experiment, Avalanche with the Pacemaker stage maintains a stable ratio of queries sent and responded, but this ratio varies drastically when the Pacemaker stage is disabled. To further exacerbate the problem, each validator manages a working set of transactions it has been queried but has yet to be committed. Having a larger working set means that the validator needs to maintain more information, and responding to query blocks will be more resource-demanding. In Avalanche without the Pacemaker stage, the working set each validator maintains is an order of magnitude larger than if their Pacemaker stages were enabled. This happens because validators can send query blocks unboundedly without coordination. If validators all query disjoint transaction sets, every validator will add all transactions to its working set while only transactions itself querys may be committed.

The Avalanche whitepaper [31] avoids this problem because the number of transactions in flight is limited at the client end, indirectly restricting the number of concurrent block queries. Their production implementation includes mechanisms akin to the Pacemaker to prevent such instability.

4.4 Diem vs Aptos: moving block dissemination off critical path

This experiment compares Diem and Aptos to highlight the benefits of separating block dissemination from consensus. Diem's throughput is inversely proportional to the time each round takes. As seen in Section 3.2, Diem does not scale since the time spent broadcasting block proposals dominates the round latency in large networks. Aptos improves on Diem by decoupling block dissemination: proposals include only proofs that block contents are persisted, not the contents themselves. As a side effect, since blocks do not carry transactions, validators cannot speculatively execute them when validating the proposal. They instead asynchronously fetch the block bodies for execution after ordering. This decoupling is gaining widespread adoption among blockchain networks [12, 14, 25]. To evaluate the benefit of separate block dissemination, we compare Diem against Aptos. We configure the client to send only no-op transactions. Diem proposals are 1 MB. Aptos batches are 1 MB, and each proposal contains 1 batch.

Figure 12 shows our emulation result. Aptos shows a much lower round latency compared with Diem. Because proposals only carry proofs that are orders



(a) Throughput. (b) Latency breakdown.

Fig. 12: Comparing Diem against Aptos in CBox.

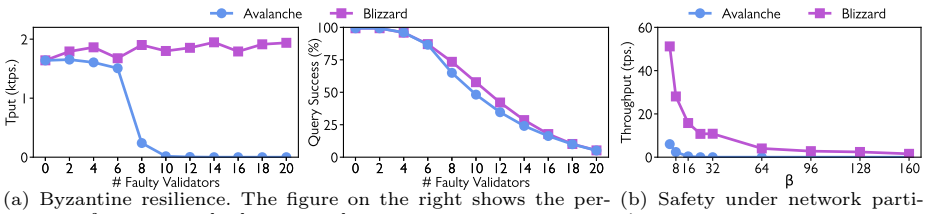
of magnitude smaller than block contents, they consume less NIC bandwidth, alleviating leader congestion. This allows Aptos to achieve a higher throughput than Diem, even when each proposal only contains one batch. However, since Aptos postpones transaction execution till after they are finalized, it will commit batches proposed by faulty validators. Assuming $\frac{1}{3}$ validators were faulty, Aptos's goodput drops to $\frac{2}{3}$ of its throughput. Although Aptos and Diem deliver similar goodput in our specific setup, Aptos can achieve much higher throughput with larger proposal sizes, significantly outperforming Diem.

4.5 Avalanche vs Blizzard

Avalanche has liveness concerns in the presence of malicious actors. Since committing a transaction requires β *consecutive* rounds of successful queries, a small group of adversaries can disrupt just one round to reset progress. Blizzard [4] addresses this issue by loosening the consecutive round requirement. In Blizzard, validators count in how many rounds each transaction is preferred. A transaction can be committed if its count exceeds all conflicting transactions combined by β. In this section, we implement a prototype of Blizzard and show that, while Blizzard is more resilient against Byzantine validators, it is not safe during network partitions. Our prototype involves 87 lines of change from Avalanche.

Byzantine Resilience To assess the Byzantine resilience of Avalanche and Blizzard, we introduce the same Byzantine validators in Section 4.3. For the correct validators, we use a Pacemaker to cap the number of concurrent queries at 40. The network consists of 40 validators. We increase the number of faulty validators in them. Figure 13a shows our results. With 8 faulty validators, an α-majority quorum only forms in less than 70% of query rounds. This makes it unlikely for Avalanche to achieve β consecutive successful rounds, causing throughput to collapse. In contrast, Blizzard tolerates low success rates since, given enough query rounds, β successful rounds will eventually accumulate, even with a small success rate, allowing Blizzard to sustain a high throughput. The performance increases slightly with more faulty validators in the network because faulty validators do not send any queries, reducing the load on correct validators.

Safety However, Blizzard's resilience to Byzantine validators comes at the cost of reduced safety against malicious clients, particularly during network partitions. In a separate experiment, we split a network of 40 validators into two groups of 20 validators each and send conflicting transactions to each group. During the network partition, each half of the network is unaware of the transactions sent to the other half and thus does not detect any conflicts. A safe protocol should avoid committing any transactions under such conditions to



(a) Byzantine resilience. The figure on the right shows the percentage of query rounds that succeed.

(b) Safety under network partition.

Fig. 13: CBox emulation results for Avalanche vs Blizzard.

prevent safety violations. Figure 13b shows the throughput of Avalanche and Blizzard with varying β. Avalanche's probabilistic consistency means that it is always possible to commit a transaction during network partitions. However, as β increases, the probability of committing such a transaction quickly converges to 0, preventing the network from making progress. In contrast, Blizzard validators will commit transactions even with a prohibitively large $\beta = 160$. This indicates that Blizzard, while offering higher throughput, is unsafe during network partitions — a risk that can occur even in a partially synchronous network.

5 Discussion and Future Directions

Multi-machine emulation CBox can scale to multiple physical nodes. A key challenge is to share the host machine's network bandwidth among colocated validators. This can be alleviated by the low per-validator bandwidth demand (typically 1 Gbps) relative to CBox's 100 Gbps data center network. Additionally, validators often send messages with the same content to each other. CBox can deduplicate messages between validators on the same physical machine pairs.

Storage emulation CBox can be extended to emulate storage. This extension requires sharing disk resources among validators. In a safe blockchain, benign validators remain synchronized—they store identical states and issue the same storage requests around the same time. CBox can use a single storage instance. For each request, CBox queries the instance once, records the result and access latency, and emulates storage accesses for duplicate requests by other validators.

Domain-specific scheduler CBox's use of Tokio scheduler brings several limitations. First, its non-preemptive nature precludes time-sharing among tasks, which can lead to starvation. Second, since the Tokio scheduler is unaware of which validator each task belongs to, it may fail to allocate the physical resources evenly among validators. A domain-specific scheduler can overcome these limitations, simplify implementation in CBox, and reduce software overheads.

6 Related Work

Evaluating Blockchain Performance Existing evaluations of blockchain systems fall into two main categories. The first involves benchmarking real implementations using suites that generate diverse client transactions [27, 28, 32, 9, 17, 34, 39, 15]. While these studies reflect realistic performance, their results depend heavily on the chosen implementation. This is problematic since these implementations often include non-essential features that skew performance. For instance,

Gromit [27] found that Avalanche underperformed due to compute-heavy credential management—a feature that can be extended to any blockchain systems.

The second category includes implementing various blockchain systems in controlled environments. CBox belongs to this category. Many works in this category use simulation, often focusing on systems using Nakamoto Consensus [2, 11, 16, 24, 29, 30]. Simulation allows experimenting with arbitrarily large networks using one physical machine, but has several drawbacks. First, in simulation, many factors, such as compute time and resource constraints, are often overlooked. Second, in blockchains, simulation time complexity typically grows superlinearly with the number of validators, limiting scalability. Bedrock [3] and BlockEmulator [22] are two exceptions. Instead of simulating the network, they implement various blockchains in a unified platform to simplify performance analysis and comparison. They, however, do not aim for large-scale evaluation.

Testing Blockchain Implementation Numerous tools [1, 5, 6, 35, 37] have been used widely in production to detect bugs in blockchain implementations. These tools run unmodified code in controlled environments to check the robustness of the tested implementation in various scenarios. In these controlled environments, concurrent events are often serialized to ensure determinism and reproducibility, allowing tests to run on a single machine but significantly perturbing performance. CBox is orthogonal to blockchain testing tools.

Network Emulation Network emulation has been extensively studied for decades [20, 21, 19, 38]. These works focus on the topology and characteristics of the network infrastructures. The mailbox currently employs a simple network model but can be extended with more advanced WAN features.

Another direction explores collocating multiple nodes on a single machine for scalability testing [36, 18, 40]. As noted in Section 2, CBox adopts emulation techniques (e.g., SPC and PIL) from STest [36]. These techniques are particularly effective for blockchains. DieCast [18] collocates nodes on one machine by time-sharing VMs, advancing time only when a VM is scheduled. While being a generic solution, DieCast is heavyweight due to the VM scheduling for the entire physical machine, making it less suited for rapid prototyping. Exalt [40] targets I/O-intensive systems. It compresses data to zero bytes on disk as the performance of these systems depends on the input length rather than the exact content. This technique is less effective for blockchains, which primarily store small values.

7 Conclusion

We present CBox, an emulation-based prototyping and evaluation framework for blockchain systems. CBox builds on our observation that almost all existing blockchain systems can be characterized using a common stage event-driven model, and emulation can significantly reduce the hardware requirements of validators. Our evaluation demonstrates that CBox can accurately emulate a large-scale network with a single machine and can be used to uncover design issues in blockchain systems that might otherwise be overlooked. We believe CBox will help developers better understand the behavior of various blockchain systems and design more reliable, high-performance blockchain solutions.

References

1. Antithesis. https://antithesis.com/
2. Alharby, M., van Moorsel, A.: Blocksim: An extensible simulation tool for blockchain systems. Frontiers Blockchain **3**, 28 (2020)
3. Amiri, M.J., Wu, C. et al.: The bedrock of byzantine fault tolerance: A unified platform for BFT protocols analysis, implementation, and experimentation. In: 21st USENIX Symposium on Networked Systems Design and Implementation, Santa Clara, CA, April 15-17, 2024
4. Amores-Sesar, I., Cachin, C., Schneider, P.: An analysis of avalanche consensus. In: Structural Information and Communication Complexity - 31st International Colloquium, Vietri sul Mare, Italy, May 27-29, 2024
5. Androulaki, E., Barger, A. et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, April 23-26, 2018
6. Aoki, Y., Otsuki, K. et al.: Simblock: A blockchain network simulator (2019), https://arxiv.org/abs/1901.09777
7. Association, D.: Diem. https://www.diem.com/en-us/
8. Avalanche: Avalanche. https://www.avax.network/
9. Dinh, T.T.A., Wang, J. et al.: BLOCKBENCH: A framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data, Chicago, IL, USA, May 14-19, 2017
10. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. Commun. ACM **61**(7), 95–102
11. Faria, C., Correia, M.: Blocksim: Blockchain simulator. In: IEEE International Conference on Blockchain, Atlanta, GA, USA, July 14-17, 2019
12. Foundation, A.: Aptos. https://aptosfoundation.org/
13. Foundation, E.: Proposer-builder separation. https://ethereum.org/en/roadmap/pbs/
14. Foundation, S.: Sui. https://sui.io/
15. Geyer, F.C., Jacobsen, H.A. et al.: An end-to-end performance comparison of seven permissioned blockchain systems. In: Proceedings of the 24th International Middleware Conference, Bologna, Italy, December 11-15, 2023
16. Gouda, D.K., Jolly, S., Kapoor, K.: Design and validation of blockeval, A blockchain simulator. In: 13th International Conference on COMmunication Systems & NETworkS, Bangalore, India, January 5-9, 2021
17. Gramoli, V., Guerraoui, R. et al.: Diablo: A benchmark suite for blockchains. In: Proceedings of the Eighteenth European Conference on Computer Systems, Rome, Italy, May 8-12, 2023
18. Gupta, D., Vishwanath, K.V., Vahdat, A.: Diecast: Testing distributed systems with an accurate scale model. In: 5th USENIX Symposium on Networked Systems Design & Implementation, April 16-18, 2008, San Francisco, CA, USA
19. Handigol, N., Heller, B. et al.: Reproducible network experiments using container-based emulation. In: Conference on emerging Networking Experiments and Technologies, December 10-13, 2012, Nice, France
20. Hemminger, S., others: Network emulation with netem. In: Linux conf au. vol. 5
21. Hibler, M., Ricci, R. et al.: Large-scale virtualization in the emulab network testbed. In: Proceedings of the 2008 USENIX Annual Technical Conference, Boston, MA, USA, June 22-27, 2008.
22. Huang, H., Ye, G. et al.: Blockemulator: An emulator enabling to test blockchain sharding protocols. IEEE Trans. Serv. Comput. **18**(2) (2025)

23. Kiffer, L., Neu, J. et al.: Nakamoto consensus under bounded processing capacity. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, Salt Lake City, UT, USA, October 14-18, 2024
24. Ma, X., Wu, H. et al.: Cblocksim: A modular high-performance blockchain simulator. In: IEEE International Conference on Blockchain and Cryptocurrency, Shanghai, China, May 2-5, 2022
25. Marsh, B., Landers, S., Jog, J.: Sei giga (2025), https://arxiv.org/abs/2505.14914
26. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf
27. Nasrulin, B., de Vos, M. et al.: Gromit: Benchmarking the performance and scalability of blockchain systems. CoRR **abs/2208.11254** (2022)
28. Pan, H., Duan, X. et al.: BBB: A lightweight approach to evaluate private blockchains in clouds. In: IEEE Global Communications Conference, Virtual Event, Taiwan, December 7-11, 2020
29. Pandey, S., Ojha, G. et al.: Blocksim: A practical simulation tool for optimal network design, stability and planning. In: IEEE International Conference on Blockchain and Cryptocurrency, Seoul, Korea, May 14-17, 2019
30. Ramachandran, P., Agrawal, N. et al.: Blocksim-net: a network-based blockchain simulator. Turkish J. Electr. Eng. Comput. Sci. **30**(2) (2022)
31. Rocket, T., Yin, M. et al.: Scalable and probabilistic leaderless BFT consensus through metastability. CoRR (2019)
32. Saingre, D., Ledoux, T., Menaud, J.: Bctmark: a framework for benchmarking blockchain technologies. In: 17th IEEE/ACS International Conference on Computer Systems and Applications, Antalya, Turkey, November 2-5, 2020
33. Satija, S., Mehra, A. et al.: Blockene: A high-throughput blockchain over mobile devices. In: 14th USENIX Symposium on Operating Systems Design and Implementation, Virtual Event, November 4-6, 2020
34. Sedlmeir, J., Ross, P. et al.: The DLPS: A new framework for benchmarking blockchains. In: 54th Hawaii International Conference on System Sciences, Kauai, Hawaii, USA, January 5, 2021
35. Stoykov, L., Zhang, K., Jacobsen, H.A.: Vibes: fast blockchain simulations for large-scale peer-to-peer networks. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos (2017)
36. Stuardo, C.A., Leesatapornwongsa, T. et al.: Scalecheck: A single-machine approach for discovering scalability bugs in large distributed systems. In: 17th USENIX Conference on File and Storage Technologies, Boston, MA, February 25-28, 2019
37. Truffle: Ganache. https://archive.trufflesuite.com/ganache/
38. Vahdat, A., Yocum, K. et al.: Scalability and accuracy in a large-scale network emulator. In: 5th Symposium on Operating System Design and Implementation, Boston, Massachusetts, USA, December 9-11, 2002
39. Wang, R., Ye, K. et al.: xbcbench: A benchmarking tool for analyzing the performance of blockchain systems. In: Blockchain and Trustworthy Systems - Third International Conference, Guangzhou, China, August 5-6, 2021
40. Wang, Y., Kapritsos, M. et al.: Exalt: Empowering researchers to evaluate large-scale storage systems. In: Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, Seattle, WA, USA, April 2-4, 2014
41. Welsh, M., Culler, D.E., Brewer, E.A.: SEDA: an architecture for well-conditioned, scalable internet services. In: Proceedings of the 18th ACM Symposium on Operating System Principles, Chateau Lake Louise, Banff, Alberta, Canada, October 21-24, 2001