RBlockSim: Parallel and Distributed Simulation for Blockchain Benchmarking

Adriano Pimpini
adriano.pimpini@gmail.com
Tor Vergata University of Rome
Rome, Italy

Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it
Tor Vergata University of Rome
Rome, Italy

ABSTRACT

Since the introduction of Bitcoin in 2008, blockchain technology's popularity has been rapidly on the rise. This has led to and benefited from the introduction of numerous new implementations addressing specific needs regarding performance, scalability, privacy, etc.

Developing new implementations requires making design choices that greatly influence the blockchain's behaviour. While deploying full-scale networks for evaluation is impractical and costly, simulation offers a safe and convenient environment to test and benchmark different implementations against the requirements.

Although single-threaded simulation of blockchain networks is available, it can incur long execution times when simulating large-scale scenarios. Additionally, it is constrained by the machine's RAM, limiting the size of networks one may study.

In this paper, we introduce and benchmark RBlockSim, a simulation approach that leverages optimistic parallel and distributed discrete-event simulation to overcome the above limitations and provides a modular, high-performance test-bed for blockchain evaluation.

CCS CONCEPTS

• **Computing methodologies** → **Discrete-event simulation**; **Massively parallel and high-performance simulations**; • **Security and privacy** → *Distributed systems security*; • **General and reference** → Evaluation.

KEYWORDS

Blockchain, Speculative Parallel Discrete Event Simulation, Security, Performance, Accuracy, Benchmarking.

ACM Reference Format:
Adriano Pimpini and Alessandro Pellegrini. 2025. RBlockSim: Parallel and Distributed Simulation for Blockchain Benchmarking. In *39th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '25), June 23–26, 2025, Santa Fe, NM, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3726301.3728421

1 INTRODUCTION

Blockchain technology has experienced massive growth in popularity since the introduction of Bitcoin [19] in 2008. Its multifaceted appeal stems from its various properties, including decentralisation, security, and transparency, which make blockchain attractive for a variety of applications beyond cryptocurrency.

Since then, many different blockchain implementations have appeared for specific use cases. When designing such blockchains, certain requirements must be considered. A blockchain can be tailored to meet these needs and achieve desired performance benchmarks by carefully selecting design choices, such as the used consensus algorithm, the block size, or the generation interval. Collectively, these choices may affect transaction rates, energy efficiency, propagation delays, or, in general, the balance between security and performance. However, the effects of design choices on the network's emergent behaviour cannot be precisely evaluated with pencil and paper: they need to be observed and measured at run time. Deploying the blockchain network at scale for realistic benchmarking is a prohibitive (if not unfeasible) endeavour, requiring unrealistic amounts of computational power, memory, storage, and time. Simulation is an alternative approach that offers a more efficient way to evaluate the performance of a blockchain implementation: indeed, simulation allows for the controlled and systematic testing of various design choices, significantly reducing the computational and resource overhead by mimicking costly operations. This approach eliminates hardware bottlenecks and prevents skewed results caused by physical machine limitations, ensuring a more accurate and efficient evaluation of the blockchain's performance, scalability, and security.

Simulating blockchains with Parallel Discrete Event Simulation (PDES) [10] can provide non-negligible benefits over existing solutions. PDES is superior to sequential simulation (see, e.g., [2, 9, 11]) due to its ability to leverage concurrency, thereby significantly enhancing simulation efficiency and scalability. Unlike sequential simulation, which processes events one at a time, PDES distributes events across multiple processors and/or multiple machines, allowing for concurrent event processing to reduce the overall simulation time. This concurrency is especially beneficial in large-scale simulations, such as those modelling blockchain networks, where the volume of events can be substantial. Compared to time-stepped distributed simulation (see, e.g., [24]), which advances all simulated entities in fixed time increments and can be inefficient when events are sparse, PDES processes events precisely when they occur, optimising resource usage and maintaining high simulation fidelity. The improved accuracy and fidelity of discrete-event simulation have already been recognised in multiple domains, including blockchain simulation (see, e.g., [4, 6, 22]).

At the same time, deploying a concurrent simulation requires special attention to deliver correct results, thus requiring some synchronisation protocol. Synchronisation is fundamental in PDES because it ensures the correct chronological order of events across

SIGSIM-PADS '25, June 23–26, 2025, Santa Fe, NM, USA

ACM ISBN 979-8-4007-1591-4/2025/06
https://doi.org/10.1145/3726301.3728421

multiple processors, preventing causality errors where an event might be processed before its preceding events. Without proper synchronisation, simulations can produce inaccurate results because events could be handled out of order. Two primary synchronisation methods are used: conservative [5] and optimistic [14]. Conservative synchronisation prevents any event from being processed until it is certain that no preceding events exist, possibly leading to idle waiting times but ensuring correctness. On the other hand, optimistic synchronisation allows events to be processed speculatively, rolling back and correcting any errors if out-of-order events are detected later. This method can significantly improve performance, but requires efficient error detection and rollback mechanisms to maintain accuracy. Recently, conservative synchronisation has been shown to be an optional accelerator for an underlying optimistic synchronisation algorithm [15]. Moreover, speculative synchronisation can also favour performance when the model is tightly coupled, thanks to improved locality management [13, 18]. Therefore, we explicitly exploit speculative synchronisation, based on the Time Warp synchronisation algorithm [14], as the synchronisation strategy to deliver efficient parallel/distributed blockchain simulations.

In this paper, we present RBlockSim, a high-performance, parallel, and distributed blockchain simulation framework that leverages speculative PDES to achieve scalability and efficiency in large-scale blockchain benchmarking. RBlockSim supports customisable blockchain configurations, allowing users to explore consensus protocols, network topologies, and adversarial behaviours. Its modular design facilitates rapid experimentation, while its optimised blockchain management system ensures efficient fork management and transaction handling. We release RBlockSim as open source[1] as a tool enabling researchers and practitioners to evaluate blockchain protocols, analyse emergent network behaviours, and experiment with various consensus mechanisms and adversarial scenarios in a modular and extensible environment.

The rest of this paper is structured as follows. In Section 2, we present the paper's background context and scope. In Section 3, we explore related work conducted in the area. Section 4 explores RBlockSim's architecture and various modules in detail. In Section 5, the experimental setup and the performance and security results are presented.

2 BACKGROUND AND SCOPE

A blockchain is a distributed ledger replicated across a peer-to-peer network, responsible for recording and managing a continuously growing list of transactions in a secure and tamper-proof manner. Transactions are initially created by participants, digitally signed using asymmetric cryptography to ensure authenticity and integrity, and then broadcasted to the network. Once disseminated, transactions are aggregated by network participants, typically miners or validators, into batches called *blocks*.

Each block contains a cryptographic reference to its predecessor (the previous block's hash), the current batch of validated transactions, a timestamp, and a cryptographic hash that uniquely identifies the block itself. This structure inherently creates a linear and immutable sequence—thus forming the blockchain. The cryptographic linkage between blocks ensures that modifying even a single transaction in any block invalidates the subsequent blocks, as their hashes would no longer match, guaranteeing tamper resistance across the entire chain.

The decentralized and typically trustless nature of blockchain networks necessitates mechanisms to ensure consistent integration and validation of new blocks across the distributed nodes. This consistency is maintained by consensus algorithms—distributed protocols that govern how new blocks are created, validated, and integrated into the blockchain, and how conflicts or forks (simultaneous valid but divergent blockchain states) are resolved.

Consensus algorithms such as Proof of Work (PoW) and Proof of Stake (PoS) define distinct rules for block creation and validation. In PoW-based systems, employed prominently by Bitcoin [19], participants known as miners compete to solve computationally intensive cryptographic puzzles. The first miner to solve the puzzle is entitled to propose the next block and broadcast it to the network. Nodes verify the solution and the block's content before accepting it into their local version of the blockchain. This computational cost serves as a deterrent against malicious actors attempting to alter the ledger, as the required computational power and energy consumption would be prohibitively high.

In contrast, PoS-based systems, notably adopted by Ethereum [27], rely on participants (validators) who lock or "stake" cryptocurrency as collateral for their participation in the consensus process. Validators are selected deterministically or randomly, proportional to their stake, to propose and validate blocks. Malicious behavior is economically discouraged by risking loss of staked assets, which provides security without the intensive energy consumption associated with PoW.

Maintaining consistency across nodes in the presence of faulty or malicious participants—often termed Byzantine behaviour—is critical. Byzantine Fault Tolerance (BFT) mechanisms or incentives embedded within consensus algorithms ensure resilience against participants who might act unpredictably or maliciously, thereby maintaining network reliability and block consistency.

Despite a plethora of blockchain systems having been proposed and made available in recent years (we refer the reader to [12] for a comprehensive discussion), there can be many reasons [28] for wanting to implement a new one or build on top of an existing one by modifying some of its characteristics and creating a fork. The reasons can encompass performance requirements, such as scalability and transaction throughput, or efficiency needs, such as the use of a consensus protocol requiring less computational power (e.g. Ethereum 2.0 moving from PoW to PoS), or privacy and security considerations, or technological innovations, to name a few.

Many of the characteristics mentioned above are actually *emergent properties* of the network that can only be really observed at runtime. This is because they arise from the complex interactions between numerous (logical) components, such as network nodes, consensus algorithms, and transaction flows, which cannot

[1]The RBlockSim repository can be found at https://github.com/AdrianoPi/RBlockSim. The persistent version associated with this publication is available at https://doi.org/10.5281/zenodo.15281276.

be fully predicted or modelled using static analysis or theoretical approaches. These properties, including network latency, throughput, scalability, and resistance to attacks, depend on dynamic factors like node behaviour, network topology, and transaction patterns that only manifest during actual operation. Physical deployments would be ideal for evaluating emergent behaviour. However, they are impractical because of limitations in:

- *Scalability*: networks can comprise tens or hundreds of thousands of nodes[2]. Physically deploying such a large number of nodes requires considerable amounts of memory and storage, calling for either enterprise servers or a non-minimal number of networked machines.
- *Computational power*: a large-scale network has the potential to require significant computational power even when using lightweight proofs, such as PoS.
- *Cost*: the requirements above underline the need for a substantial investment in hardware to support the deployment, as both procuring and operational costs can be prohibitive. Cloud environments, while potentially offering scalability, could exacerbate these concerns.

Overcoming the above limitations is possible through simulation. Simulation can help in studying the emergent properties of blockchain systems by providing a controlled, flexible, and scalable environment to model and analyse the complex interactions within the network. Through simulation, researchers can replicate the dynamic behaviours of nodes, transactions, and consensus mechanisms under various scenarios and configurations without the resource constraints and risks associated with deploying a real blockchain network. This approach allows for the systematic exploration of how different design choices and external conditions impact performance, scalability, security, and other emergent properties. By observing these interactions in a simulated environment, developers can gain valuable insights into potential issues, optimise system parameters, and make informed decisions to enhance the overall robustness and efficiency of blockchain implementations.

This work aims to provide a simulation test bed that can help evaluate various aspects of old and novel blockchain implementations. We achieve this via RBlockSim: a model built on top of the ROme OpTimistic Simulation framework (ROOT-Sim) [21]. RBlockSim takes care of interfacing with the simulation runtime environment to provide a core structure that users can build on top of, by changing the configuration parameters or directly modifying the behaviour of specific functions to model the blockchain they want to test.

Currently, RBlockSim ships with a sample Bitcoin model. It is also able to simulate Byzantine behaviour, as presented later. The general-purpose nature of the implementation, however, permits the simulation of arbitrary adversarial behaviours, network protocols, and consensus protocols.

3 RELATED WORK

Owing to blockchain's popularity, various simulation frameworks have been developed and presented to the community [1, 20]. One of the initial attempts to simulate blockchain networks can be found in [11]. This work introduces a quantitative framework to assess the security and performance tradeoffs of various PoW blockchain configurations, which considers real-world constraints such as network propagation delays, block sizes, and block generation intervals to evaluate the optimal adversarial strategies for double-spending and selfish mining. We retain most of these capabilities while proposing a simulation methodology that can improve the scalability and performance significantly.

There are two different simulators called BlockSim [2, 9], both of which are based on discrete-event simulation. The first one [2] was developed to support various blockchains and their configurations. It is organised into three abstraction layers: network, consensus, and incentives, enabling detailed modelling and analysis of blockchain dynamics while supporting various consensus algorithms. The second one [9] offers a discrete-event simulation framework created to assess various blockchain implementations and ships with Bitcoin and Ethereum models. BlockSim [9] models different components of blockchain systems, such as blocks, transactions, ledgers, and networks, allowing users to expand these models for their design and implementation assessments. Neither simulator supports parallel/distributed execution of the simulations, which is a focal point of our work.

The work in [2] was extended in CBlockSim [17] with the goal of improving simulation performance. This is done by integrating a network module with a realistic topology generation algorithm and an efficient block propagation algorithm. It adopts a binary transaction pool structure and bitwise operations to reduce memory usage and accelerate simulation. Yet, also no parallelism is offered by CBlockSim.

In [24], LUNES-Blockchain is introduced as an agent-based blockchain simulator that utilises Parallel and Distributed Simulation techniques to improve scalability. The simulator focuses on modelling the Bitcoin protocol and examines the impact of security attacks like the Sybil Attack on the consensus protocol. LUNES-Blockchain replicates various blockchain functions, including peer-to-peer overlay management, message dissemination, and the mining process. LUNES-Blockchain is built on top of the ARTÌS runtime environment [3], which supports speculative synchronisation based on the Time Warp protocol, as we do. Nevertheless, in [24] this capability of the underlying runtime environment was not exploited, and a time-stepped synchronisation algorithm was employed. We explicitly exploit speculative synchronisation to improve the performance, scalability and accuracy of the simulations. While some performance data for LUNES-Blockchain are reported in [25], later in this work RBlockSim's times are shown to be more than an order of magnitude smaller with the used setup, with respect to those reported for LUNES-Blockchain.

The work in [16] introduces a discrete-event simulation model that analyses the dynamics of the Bitcoin blockchain network, with a focus on the strategic interactions between individual miners and mining pool managers. The model includes realistic features such as budget constraints, hashing power, mining costs, and various mining pool reward policies to simulate the decision-making processes of miners and pool managers. The study examines how these decisions affect system metrics such as mining difficulty and total hashing power through Monte Carlo simulations. Key findings include the effectiveness of dynamic difficulty adjustment in

[2]https://blockworks.co/news/ethereum-to-reach-500000-validators, accessed on January 14, 2025.

maintaining stable block generation rates and the proportionality of Bitcoin rewards to mining capacities and budget volumes. We retain some of the simulation capabilities presented in this work, while we do not necessarily focus on Bitcoin. Also, we explore parallel and distributed simulation based on speculative synchronisation, for performance and accuracy purposes.

Digital Twins are also explored in [7, 8] as a way to benchmark blockchain technology. In particular, these works address the blockchain trilemma tradeoff—balancing decentralisation, scalability, and security. The proposed Digital Twin framework integrates Dynamic Data-Driven Application Systems (DDDAS) to dynamically manage blockchain systems by continuously adapting to real-time data through simulation and optimisation. The architecture includes a scenario generator, a simulation module, and an optimiser, enabling dynamic consensus protocol selection to optimise system performance. We do not explicitly consider digital twins in this work, which can be considered orthogonal to the benchmarking via simulation methods.

In [6], the authors introduce the BSELA architecture as a solution to the limitations found in existing blockchain simulators. The core innovation of BSELA lies in its Event-Layered Architecture and the next-event time advance mechanism based on event rounds. This improves the efficiency, stability, and maintainability of discrete-event simulation. Events are categorised into intra-block and inter-block categories, managed by a Min-Heap and a queue, respectively, which optimises event processing and resource utilisation. BSELA outperforms previous simulators, providing superior performance, accuracy, and scalability for modelling blockchain networks while improving data transfer efficiency and security. We also exploit discrete-event simulation for accuracy purposes, but we expand beyond the capabilities of BSELA, allowing parallel and distributed blockchain simulation.

4 RBLOCKSIM ARCHITECTURE

In this section, the various modules of RBlockSim are presented and explained.

4.1 Blockchain Management

We define a block's *height* as one plus the number of blocks between the block and the genesis block, and we define the block *depth* as the difference between the height of the highest seen valid block and the block's height.

Each node maintains a local copy of the chain and its view of the state of various aspects of the network (e.g., transactions) based on the fork of the chain, which it considers the main chain. At the implementation level, the chain is organised in *levels* based on *block height*: each level holds all the sibling blocks with the same height, using a non-shrinking array. Figure 1 shows a diagram of the internal organisation of the chain.

When switching the main chain, the node has to recompute the view it has of the network state by reverting the effects of the blocks on the old chain and applying the effects of those on the new chain. This chain organisation allows for $O(1)$ access to any given chain level and fast back-walking to switch chains. Block lookup given the height *height* is $O(1)$ for the access to the chain level, plus $O(N_{height})$, where N_{height} is the number of blocks at

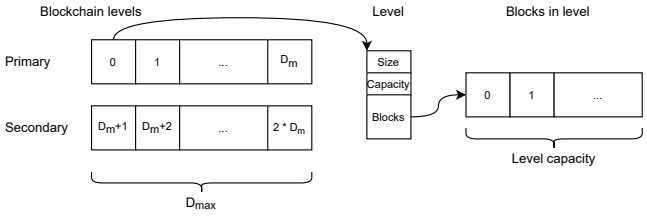


Figure 1: blockchain is managed in height levels, stored using two arrays.

that height; this kind of lookup happens when inserting a new block in the chain, which requires looking for its parent, and then possibly connecting any of the block's children that were delivered earlier than the block itself. Block access is $O(1)$ when the block's index inside the level is known. It is to be noted that N_{height} is expected to be extremely small, as the number of forks that happen at a given height is ideally very limited, making the lookup light. Besides supporting the implemented longest chain fork resolution rule, it also fits rules like GHOST [26] as it allows for fast retrieval and/or selection for inclusion of uncle blocks.

Looking deeper into the implementation, two buffers are used to store the chain data, each of fixed size D_{max}. The buffers hold the latest levels of the chain, with all the respective blocks, and alternate between one of two roles "primary" and "secondary": the primary buffer holds the highest levels. When the primary buffer becomes full, the secondary one is emptied (maintaining level capacity), and the two buffers are swapped, avoiding continuous allocations and thus capping the memory usage. With optimistic synchronisation, the advantage of decreased memory usage is doubled as it decreases the size of the memory snapshots the simulator has to keep to rollback the execution when needed.

The quantity D_{max} is configurable at compile time and can be picked based on the protocol implementation to suit the specific needs. Intuitively, D_{max} should strike the balance between being large enough to allow for smooth operation of the data structure, and being small enough to limit the cost of checkpointing required by optimistic synchronisation. This approach introduces two main constraints that are, however, not limiting:

(1) Everything deeper than D_{max} is potentially lost and, as such, to be considered committed, inaccessible and immutable. This is only partially a limitation because, while blocks are never technically committed in a blockchain, for practical uses, it is necessary to decide on a depth D_{commit} after which to consider a block as committed. Picking a value for D_{max} such that $D_{max} > D_{commit}$ is required. D_{commit} is often a reasonably small number—the number of blocks produced in a few hours—which will usually be much smaller than D_{max} in any case.
(2) No forks can be longer than D_{max}, as a node needs access to the blocks leading to the fork to compute the view of the state adequately. Since it is desirable for forks to be as short-lived as possible for any protocol, this is, again, not a real

limitation. The case of hard forks can be supported by ignoring the blocks on the uninteresting fork and/or reshaping the network topology.

4.2 Block Generation/Proposal

The implemented consensus protocol is PoW. The core principle of PoW is to demonstrate computational effort. Miners try to solve computationally expensive cryptographic problems by participating in a distributed competition. The first miner to solve a problem earns the right to propose the next block to be added to the blockchain and receives the reward. The modular implementation of RBlockSim, anyhow, allows to implement and experiment with any other consensus protocol, such as PoS, quite easily. Anyhow, PoW is a computationally heavier protocol, which better highlights the benefits of relying on simulation-based evaluation of blockchains.

With PoW, given a block containing a number of transactions, the cryptographic problem consists of finding a nonce value that, combined with the block's payload data, results in a block hash satisfying some specific criterion (e.g., having X leading 0s). Other cryptographic problems are theoretically possible, but the simplicity and effectiveness of hashing make it the most popular method for PoW by a considerable margin. In any case, the intrinsically expensive nature of PoW makes using a real implementation to benchmark a blockchain infeasible, requiring a way to simulate PoW.

Given a desired block interval τ_b, each node n has a probability of mining the block that is proportional to:

$$\frac{H_n}{\sum_{i=0}^{N} H_i}, \tag{1}$$

where H_i is the hash-rate of node i. The block generation process is memoryless, a property captured by exponential probability distributions. As such, to simulate PoW, node n does not solve a cryptographic puzzle, but rather it draws a value from an exponential distribution with mean:

$$\tau_b \frac{\sum_{i}^{N} H_i}{H_n}. \tag{2}$$

The extracted value M_{delay} will be the time to pass before n gets to mine its next block. Each time the main chain's state changes (e.g., a new block is added or another chain is identified as the main chain), a new sample has to be extracted from the exponential distribution to determine the next generation delay. A mining (`GENERATE_BLOCK`) event is consequently scheduled at time $t_{now} + M_{delay}$. Figure 2 shows the execution flow in response to the various events. In response to the mining event, the transactions to be included in the block are selected, and the block is constructed. Then, that block is applied to the local chain and propagated through gossiping.

Internally, block generation events are implemented using retractable messages, which can be rescheduled to allow a change in an event's timestamp. This prevents scheduling events, only to have to ignore them after a change in the chain, reducing memory usage and, thus, execution time.

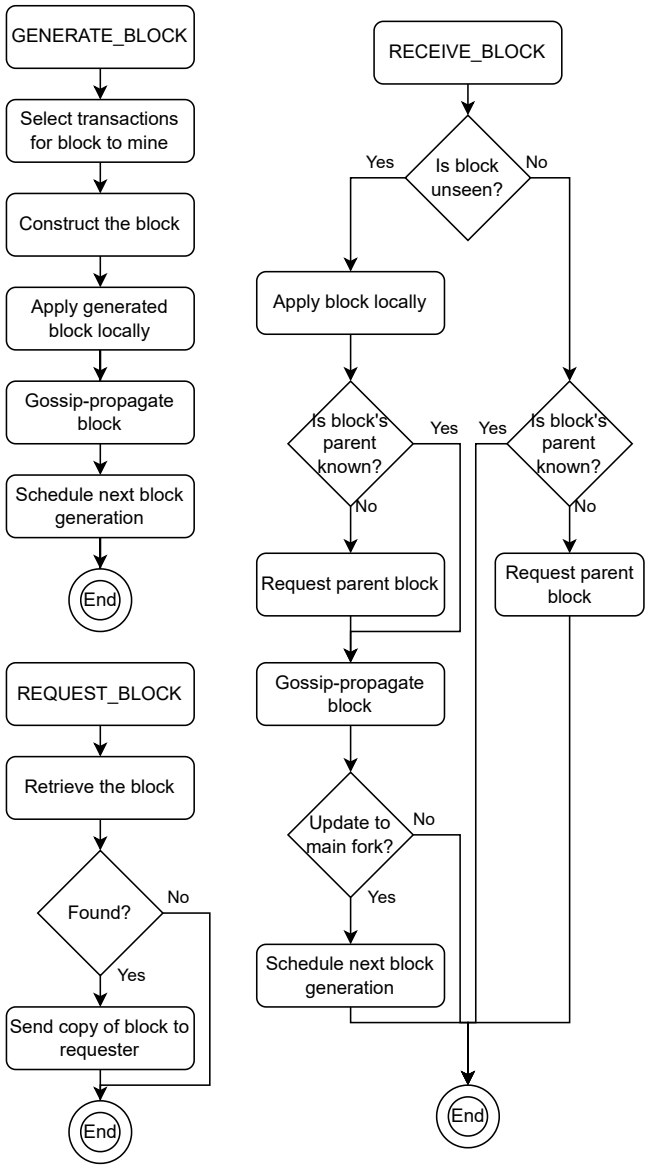


Figure 2: RBlockSim's event handling flows

Due to the speculative discrete event simulation approach, the nodes' statuses are unaligned, making a dynamic global state impossible. As such, the current implementation assumes each node's hash rate (and thus, the network's hash rate) to be constant for simplicity; more complex methods can, however, be implemented (e.g., broadcasting an update of a node's hash rate as a control message). This modelling of Proof of Work is executed in the function *scheduleNextBlockGeneration*, which the user can update to simulate different block generation approaches.

Blocks are propagated with `RECEIVE_BLOCK` events when mined or received to be forwarded. In response to them, since because of gossiping a block could be received more than once, RBlockSim first checks whether that is the case: if the received block B_r was

already seen, it only checks if the parent B_p of B_r is still missing—in which case B_p is requested from the node that sent B_r using a `REQUEST_BLOCK` event.

If B_r was instead previously unknown, it is applied to the local chain. If its parent B_p is unknown, it is requested from the node that propagated B_r. B_r is then gossiped to the node's peers. Note that blocks with unseen parents cannot become part of the main chain, in which case B_r is only stored locally without effects on the selection of the main fork. If the application of B_r updates the main fork, the block generation event for the receiving node is rescheduled.

When a node receives a `REQUEST_BLOCK` event, it seeks the requested block in the local chain. If found, the block is sent to the requester node in a `RECEIVE_BLOCK` event.

4.3 Transaction management

The used transaction management approach is taken from CBlockSim [17]: transactions are generated in bulk at the start of the simulation, and the view each node has on their availability is modelled with a bitmap that the node maintains. The bit B_t represents the state of the transaction with id t: it is set when the transaction has already been used; otherwise, it is zero. Nodes also keep track of the first transaction available to be executed and the most recent transaction seen. Similarly, blocks carry information on the id of the first transaction they include, and a bitmap representing the included transactions, all used to know which transactions to mark as executed in the node state when applying a block. When mining a block, the node selects the transactions to be included from the ones available and includes them in the block by setting the correct bit in the block's bitmap.

Nevertheless, the transaction management system is mostly a proof of concept, with static transaction generation that follows non-significant patterns, and whose main purpose is to implement functionality to have a performance comparison (see Section 5.3) using features that are as aligned as possible.

RBlockSim's modularity allows for arbitrarily more complex implementations, such as runtime transaction generation and dissemination similar to what happens with blocks, using the network module.

4.4 Topology

The blockchain network's peer-to-peer topology is randomly generated using a Python script that generates a symmetric topology given a minimum and maximum number of connections per node, then writes it to a C header and source file couple, as an adjacency list. These files are then compiled and linked with the rest of the project.

RBlockSim also supports regional distribution of nodes: in the configuration file, the user can specify:

(1) the number of regions,
(2) the percentage of simulated nodes each region holds (percentages have to add up to 1), and
(3) the average propagation latencies between different regions.

As with all other aspects of RBlockSim, this can be further expanded by providing, e.g., bandwidths to model varying transmission delay in the network module depending on block sizes and bandwidths, with limited coding effort.

4.5 Network

The network module is responsible for inter-node communication, providing primitives for block dissemination. The implemented dissemination approach is gossiping—in the same fashion as in Bitcoin Core's implementation—with a configurable fanout ϕ. Nodes have a list of peers to which they are connected (according to the generated topology—see Section 4.4). When a node mines a block, it sends it to all its neighbours. The rationale is that a block's miner wants as many nodes as possible to know about the newly mined block as fast as possible (unless it is attempting a block-withholding attack). When a node receives a block for the first time, it randomly selects ϕ of its peers to relay the block to. These actions are actually carried out rather than simulated, decoupling the Network module from the Topology module, allowing for flexible network topology to be implemented if needed. This approach also allows for a more accurate simulation of the block dissemination process.

The propagation delay for each block is computed by drawing from an exponential distribution with an average of the configured region-to-region delays.

Using gossiping also means that a node could receive a block more than once, in which case it is ignored. Another limitation to note is that a node might never receive a block through gossiping alone. For this reason, nodes can ask other nodes to provide missing blocks. Specifically, in the current implementation, a node that receives a block the parent of which it has not received yet, will ask for the parent block from the peer that relayed the "orphan" block to it.

4.6 Fork resolution

Each node keeps a local copy of the blockchain. The algorithm for chain selection uses the longest chain rule: the chain with the highest number of valid blocks is selected as the main chain. In the context of this paper, the term "main chain" is used interchangeably with "main fork", identifying the sequence of blocks selected by the node as the reference ledger for the chain status.

In RBlockSim, each block holds a "score", which quantifies—based on the chain selection rule—how fit that block is to become the last block of the main chain. Comparing block scores helps decide whether to switch the status of the main fork from one fork to the other. When using the longest chain rule, the score matches the block height. For other algorithms like GHOST, the amount of work done by the ancestors can be considered, taking into account the inclusion of uncle blocks. Chain selection is rendered lightweight as each block's score can be computed starting from its predecessor's score (and will be strictly non-decreasing).

Furthermore, switching chains is only contemplated when one of the chains is updated by comparing the scores between the main chain and the newly updated one.

When switching chains, the *switchChains* function takes care of reverting the effects of the previous main chain and applying the effects of the new main chain.

4.7 Byzantine Behaviour

The simulator can model malicious node behaviour to evaluate the impact on the blockchain network. The current implementation features simulation models of the *51%* and the *Selfish mining* attacks on a bitcoin-like network using Proof of Work with Longest-Chain consensus.

A 51% attack occurs when a node (or group of coordinated nodes) controls a majority (at least 50% + 1) of the network's hash power, which enables malicious actions such as changing the ordering of transactions, and potentially rewriting portions of the blockchain.

Selfish mining is an attack in which a miner node actively conceals the blocks it has mined from other nodes in the network by delaying their dissemination. This can allow the attacker to gain an unfair advantage, as it gets a head-start on the next node to be mined. Concealing longer sequences of blocks can allow the attacker to effectively make the computational effort of other nodes go to waste, earn the block rewards for the newly revealed blocks, but also to perform attacks such as double-spending [23].

In the presented implementation, the attacker uses two parameters to guide its behaviour: a *risk tolerance*, and the attack *depth*.

Having the longest-chain consensus, the risk tolerance used by both attacks indicates how many blocks the attacker is willing to fall behind with respect to a competing fork, in favour of a shorter fork on which, however, it owns more blocks. While even honest miners will favour forks in which they own more blocks in case of equal chain height, the risk tolerance allows attackers to stick to a "losing" chain, in the hopes of outperforming the competing miners and cashing out otherwise-lost blocks, in a high-risk, high-reward approach.

The attack depth is used in selfish mining attacks and specifies the number of blocks to mine secretly before releasing them to the network to attempt a chain takeover. If the newly released fork is (permanently) adopted as the main chain by honest nodes, the attack is considered successful for our purposes.

5 EXPERIMENTAL EVALUATION

The experiments were run on a machine with two Intel Xeon CPUs E5-2696 v4 @ 2.20GHz, with 22 physical cores each, for a total of 44 physical cores, or 88 hyperthreads. The machine has 256 GB of RAM.

5.1 Statistical Validation

To validate the simulated behaviour, we check that each node mines a number of blocks that is proportional to the percentage of network hash rate it owns, irrespective of whether the blocks are part of the main chain.

To do so, we extracted the number of blocks mined by nodes that owned a known percentage of the network's hash power, compute the percentage of blocks mined with respect to the total number of mined blocks, after which the block mining rate is averaged over the runs and plotted against the node's hash rate.

Figure 3 reports the results of our experimentation: the grey, dashed line is the expected behaviour (number of mined blocks directly proportional to percentage of network hash power owned), and the blue line is the observed behaviour.

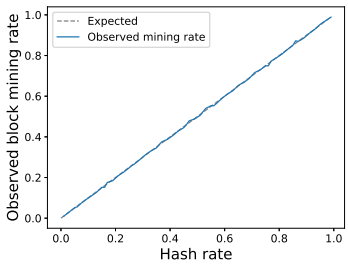


Figure 3: node's hash rate and observed block mining rate.

We can see how the observed behaviour closely follows the expected one, confirming that the simulation model adheres to the statistical one.

5.2 Byzantine Behaviour Simulation

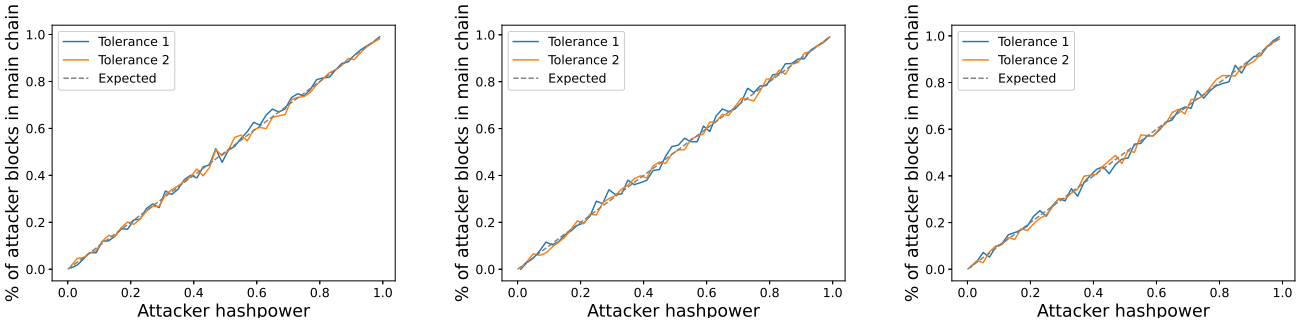
To exemplify the capabilities of the presented framework, simulation runs have been carried out exploring the network's behaviour in case of attacks, as explained in Section 4.7. Every run lasts 24 hours of simulation time and models Bitcoin's behaviour, using PoW with a block interval of 10 minutes.

When simulating the 51% attack, we varied the attacker hash rate from 1% to 99% of the network rate, while using different risk tolerance values. The simulations used networks of varying size. Figures 4a, 4b, and 4c show the results, comparing the possessed hash rate and the percentage of main chain's blocks that were mined by the attacker. We can see how possessing a higher hash rate without actively trying to sabotage the network does not seem to result in disproportionate block acceptance rates, even in the case of a stubborn node that favours non-main chain forks on which it mined more blocks.

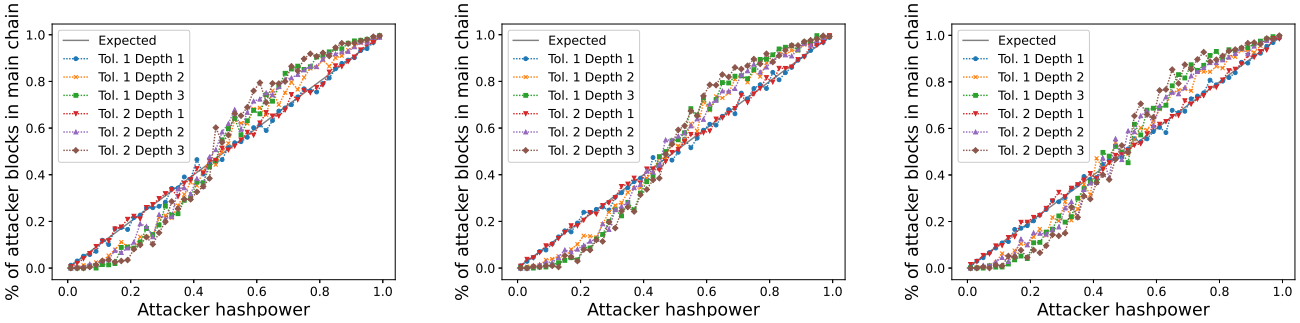
To simulate selfish mining, following the approach used for the 51% attack, we varied the attacker's hash rate, the risk tolerance, as well as the attack depth. This allows us to observe the effect of these parameters on the attack's success rate. Figures 4d, 4e and 4f compare the hash rate possessed by the node with the percentage of mined blocks included in the main chain, while Figure 5 shows the probability of a selfish mining attack succeeding, based on the hash rate that the attacker controls. It is interesting to note how, even with modest portions of a network's hash rate, an attacker can still manage to carry out a successful attack, albeit with low probability. At the same time, we see that attempting such an attack without controlling at least 50% of the global hash power is counter-productive in terms of committing blocks to the main chain. Nevertheless, the data shows that an attacker is capable of forcing a switch of honest nodes to the concealed chain after releasing it, which opens the way to, e.g., the reordering of blocks and transactions, while wasting the computational effort of competing nodes.

5.3 Performance evaluation

To evaluate RBlockSim's performance, a series of runs was performed varying both the network size and the number of worker

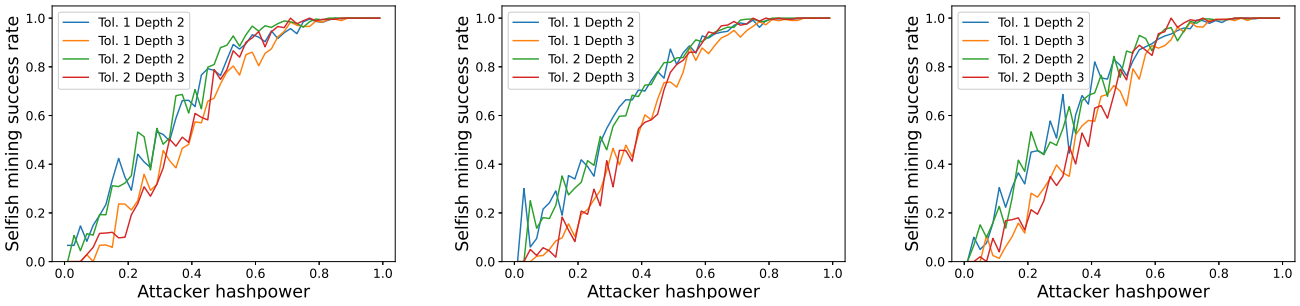


(a) 51% - 100 Nodes (b) 51% - 1,000 Nodes (c) 51% - 10,000 Nodes



(d) Selfish mining - 100 Nodes (e) Selfish mining - 1,000 Nodes (f) Selfish mining - 10,000 Nodes

Figure 4: Comparing Attacker Hash Rate and Block Success Rate



(a) 100 Nodes (b) 1,000 Nodes (c) 10,000 Nodes

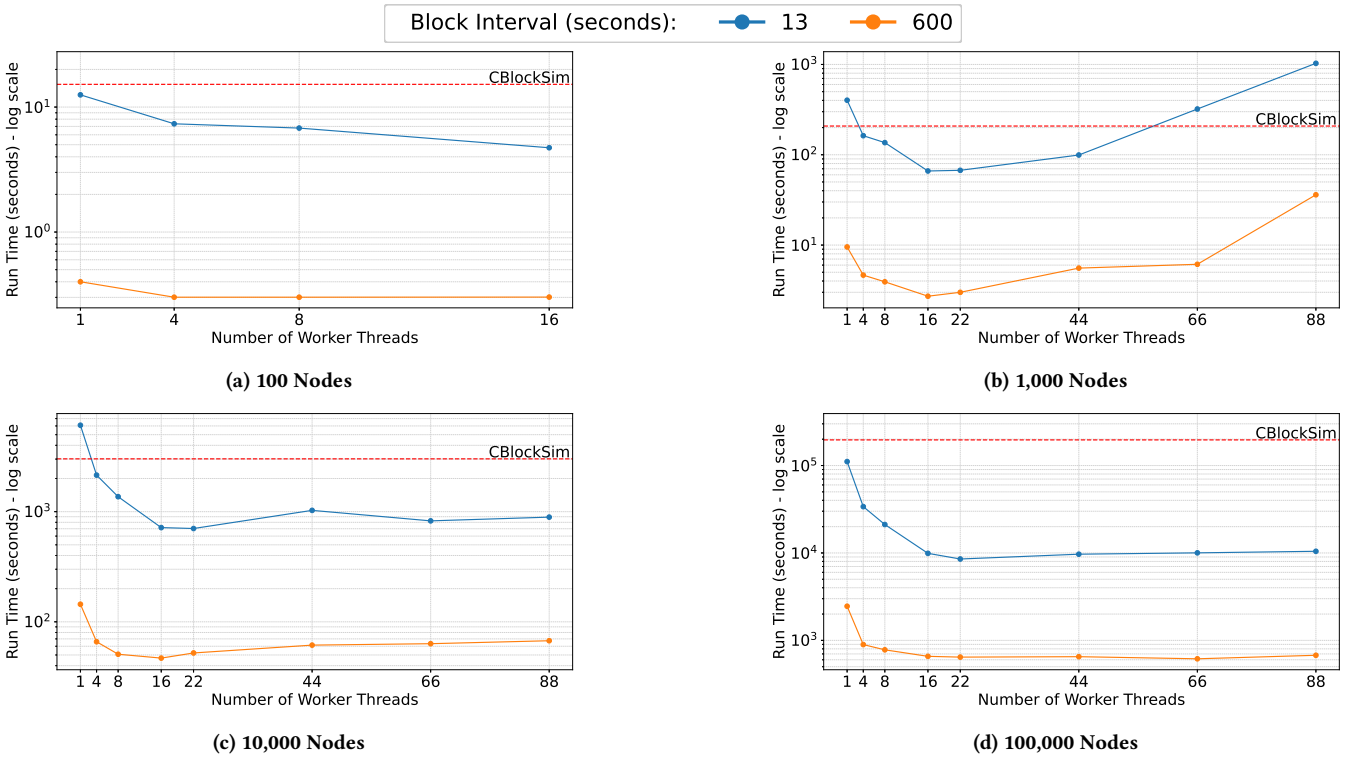
Figure 5: Success rate of selfish mining attacks when varying node hash rate

threads for the simulation. The tested networks have been simulated for 24 hours, with their sizes varying from 100 to 100,000 nodes. The results are reported in Figure 6.

Each data point represents the average of five runs, except for 100,000 nodes with a block interval of 13 seconds for thread counts 1, 4, and 8, which are single or double runs due to time constraints.

Figure 7 shows peak RAM usage across all runs, again varying worker threads, network sizes, and block generation interval. CBlockSim reports a RAM usage of 4.5GB for a network of 10,000 nodes. For a network of 100,000 nodes, we observed a peak RAM usage of 99.8 GB.

The graphs show how RBlockSim benefits from the multiple threads made available to it while staying competitive at lower thread counts. Performance peaks at 16 to 22 worker threads, after which it degrades. This degradation is more noticeable on smaller networks, with fewer nodes per worker thread. A lower number of logical processes leads to an increase in clock skew, which leads to an increased number of rollbacks and slows down the simulation. Using a real implementation of gossiping as the communication protocol between nodes further exacerbates these issues. Indeed, since blocks are disseminated by gossip-forwarding them at each node they reach, every block mined by node m results in a total of:



(a) 100 Nodes

(b) 1,000 Nodes

(c) 10,000 Nodes

(d) 100,000 Nodes

Figure 6: RBlockSim execution Wall Clock Times when varying worker threads, network sizes, and average block interval. Each figure reports a different network size. The red dashed line indicates the time taken by CBlockSim [17] to simulate a network of the same size. Figure 6a only goes up to 16 threads as performance degrades greatly when the number of worker threads approaches that of logical processes, especially in cases of highly connected simulations.

$$p_m + \sum_{i=0, i \neq m}^{N} min(\phi, p_i) \tag{3}$$

simulation events generated, where p_i is the number of peers of node i, and ϕ is the gossiping fanout. This number of events holds regardless of the fact that nodes that have already seen the block essentially ignore further instances of its propagation. In addition to increasing the number of events to manage and propagate, this also leads to an increased rollback cost. Indeed, rolling back a single block mining event can result in a non-negligible number of cascading rollbacks, depending on how fast (in Wall-Clock Time) the event propagated to the other nodes.

Instead, the peak RAM usage graphs support the idea that RBlockSim can simulate large networks using non-prohibitive amounts of memory, which is readily available on modern personal computers. More memory is needed for very large-scale networks (100,000 nodes and up), but it is important to note that the simulator can also execute in a distributed environment using MPI. This renders the simulation of very large networks viable not only on server-size machines but also using consumer hardware: two widely available machines with 32GB of RAM each can easily support the simulation of a network with 100,000 nodes, a feat impossible to achieve at this level of detail and speed without distributed simulation. Additionally, heavier simulation models which bottleneck on RAM bandwidth could benefit from being simulated on various machines, reaping the twofold advantage of having reduced contention on the memory bus and increased bandwidth at the same time.

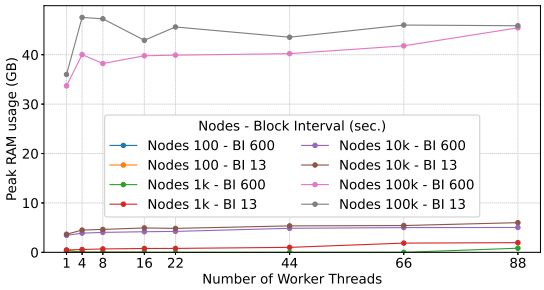


Figure 7: Peak ram usages across various configurations

In Figure 8, we show the aggregated RAM usage during the execution of attack simulations, for different network sizes. As can be seen, memory consumption is generally bounded, although in the case of very large networks, e.g. with 100,000 nodes, a steady memory consumption is reached only close to the simulation end. These results highlight that RBlockSim can deliver performance improvements while efficiently using the available resources.

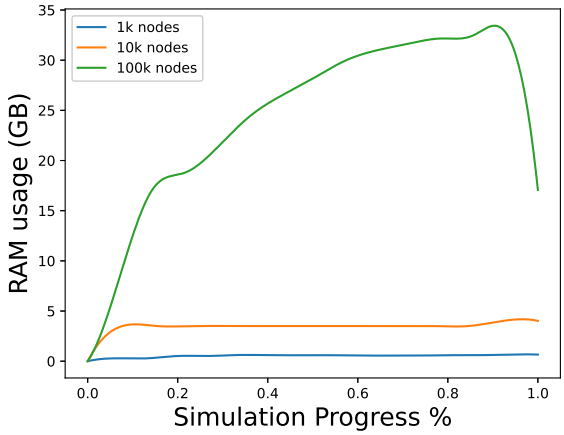


Figure 8: Aggregated RAM usages for different network sizes

6 CONCLUSIONS

We have presented and benchmarked RBlockSim, a parallel, distributed, and high-performance simulation test bed for evaluating blockchain systems and their resilience to attacks.

We introduced the various modules composing it, explained their functionality, and provided an implementation that simulates various aspects of the Bitcoin protocol.

We have shown that RBlockSim allows rapid simulations of large-scale networks, providing an efficient solution for evaluating different decisions regarding the protocol's architecture in a much shorter time frame than available alternatives. The possibility of distributing the workload allows larger networks to be simulated on widely available consumer machines, further reducing the entry barriers of provisioning and maintenance costs.

REFERENCES

[1] Adel Albshri, Ali Alzubaidi, Bakri Awaji, and Ellis Solaiman. 2022. Blockchain simulators: A systematic mapping study. In *2022 IEEE International Conference on Services Computing (SCC)*. IEEE, Piscataway, NJ, USA, 284–294. https://doi.org/10.1109/scc55611.2022.00049

[2] Maher Alharby and Aad van Moorsel. 2020. BlockSim: An Extensible Simulation Tool for Blockchain Systems. *Frontiers in Blockchain* 3 (June 2020), 459097. https://doi.org/10.3389/fbloc.2020.00028

[3] L Bononi, M Bracuto, G D'Angelo, and L Donatiello. 2006. Scalable and efficient parallel and distributed simulation of complex, dynamic and mobile systems. In *Proceedings of the 2005 Workshop on Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems (FIRB-PERF'05)*. IEEE, Piscataway, NJ, USA, 136–145. https://doi.org/10.1109/firb-perf.2005.17

[4] Vanessa Büsing-Meneses, Cristina Montañola-Sales, Josep Casanovas-Garcia, and Alessandro Pellegrini. 2015. Analysis and optimization of a demographic simulator for parallel environments. In *Proceedings of the 2015 Winter Simulation Conference (WSC)*. IEEE, Piascataway, NJ, USA, 3218–3219. https://doi.org/10.1109/wsc.2015.7408478

[5] K M Chandy and J Misra. 1981. Asynchronous distributed simulation via a sequence of parallel computations. *Commun. ACM* 24, 4 (April 1981), 198–206. https://doi.org/10.1145/358598.358613

[6] Bo Cui and Yun Hu. 2024. BSELA: A blockchain simulator with event-layered architecture. *Future generations computer systems: FGCS* 151 (Feb. 2024), 182–195. https://doi.org/10.1016/j.future.2023.09.034

[7] Georgios Diamantopoulos, Rami Bahsoon, Nikos Tziritas, and Georgios Theodoropoulos. 2023. SymBChainSim: A novel simulation tool for dynamic and adaptive blockchain management and its trilemma tradeoff. In *Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS'23)*. ACM, New York, NY, USA, 118–127. https://doi.org/10.1145/3573900.3591121

[8] Georgios Diamantopoulos, Nikos Tziritas, Rami Bahsoon, and Georgios Theodoropoulos. 2022. Digital twins for dynamic management of blockchain systems. In *2022 Winter Simulation Conference (WSC)*. IEEE, Piscataway, NJ, USA, 2876–2887. https://doi.org/10.1109/wsc57314.2022.10015447

[9] Carlos Faria and Miguel Correia. 2019. BlockSim: Blockchain Simulator. In *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, Piscataway, NJ, USA, 439–446.

[10] Richard M Fujimoto. 1990. Parallel Discrete Event Simulation. *Commun. ACM* 33, 10 (Oct. 1990), 30–53. https://doi.org/10.1145/84537.84545

[11] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. Association for Computing Machinery, New York, NY, USA, 3–16.

[12] Anjee Gorkhali, Ling Li, and Asim Shrestha. 2020. Blockchain: a literature review. *Journal of management analytics* 7, 3 (July 2020), 321–343. https://doi.org/10.1080/23270012.2020.1801529

[13] Mauro Ianni, Romolo Marotta, Davide Cingolani, Alessandro Pellegrini, and Francesco Quaglia. 2018. The Ultimate Share-Everything PDES System. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '18)*. ACM, New York, NY, USA, 73–84. https://doi.org/10.1145/3200921.3200931

[14] David R Jefferson. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7, 3 (July 1985), 404–425. https://doi.org/10.1145/3916.3988

[15] David R Jefferson and Peter D Barnes, Jr. 2022. Virtual time III, Part 1: Unified Virtual Time synchronization for parallel discrete event simulation. *ACM transactions on modeling and computer simulation: a publication of the Association for Computing Machinery* 32, 4 (Oct. 2022), 1–29. https://doi.org/10.1145/3505248

[16] Kejun Li, Yunan Liu, Hong Wan, and Yining Huang. 2021. A discrete-event simulation model for the Bitcoin blockchain network with strategic miners and mining pool managers. *Computers & operations research* 134, 105365 (Oct. 2021), 105365. https://doi.org/10.1016/j.cor.2021.105365

[17] Xuyang Ma, Han Wu, Du Xu, and Katinka Wolter. 2022. CBlockSim: A Modular High-Performance Blockchain Simulator. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, Piscataway, NJ, USA, 1–5. https://doi.org/10.1109/icbc54727.2022.9805504

[18] Federica Montesano, Romolo Marotta, and Francesco Quaglia. 2024. Spatial/temporal locality-based load-sharing in speculative discrete event simulation on multi-core machines. *ACM transactions on modeling and computer simulation: a publication of the Association for Computing Machinery* 35, 1 (Jan. 2024), 30. https://doi.org/10.1145/3639703

[19] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.

[20] Remigijus Paulavičius, Saulius Grigaitis, and Ernestas Filatovas. 2021. A Systematic Review and Empirical Analysis of Blockchain Simulators. *IEEE Access* 9 (2021), 38010–38028.

[21] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia. 2012. The ROme OpTimistic Simulator: Core Internals and Programming Model. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS)*. ICST, Brussels, Belgium, 96–98. https://doi.org/10.4108/icst.simutools.2011.245551

[22] Adriano Pimpini, Andrea Piccione, and Alessandro Pellegrini. 2022. On the accuracy and performance of spiking neural network simulations. In *Proceedings of the 2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT '22)*. IEEE, Piscataway, NJ, USA, 96–103. https://doi.org/10.1109/ds-rt55542.2022.9932062

[23] Carlos Pinzón and Camilo Rocha. 2016. Double-spend attack models with time advantange for bitcoin. *Electronic notes in theoretical computer science* 329 (Dec. 2016), 79–103. https://doi.org/10.1016/j.entcs.2016.12.006

[24] Edoardo Rosa, Gabriele D'Angelo, and Stefano Ferretti. 2019. Agent-Based Simulation of Blockchains. In *Communications in Computer and Information Science*. Springer Singapore, Singapore, 115–126. https://doi.org/10.1007/978-981-15-1078-6_10

[25] Luca Serena, Gabriele D'Angelo, and Stefano Ferretti. 2022. Security analysis of distributed ledgers and blockchains through agent-based simulation. *Simul. Model. Pract. Theory* 114, 102413 (Jan. 2022), 102413.

[26] Yonatan Sompolinsky and Aviv Zohar. 2013. Accelerating bitcoin's transaction processing. Fast money grows on trees, not chains. *IACR Cryptology ePrint Archive* 2013 (2013), 881.

[27] Gavis Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf.

[28] Karl Wust and Arthur Gervais. 2018. Do you Need a Blockchain?. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, Piscataway, NJ, USA, 45–54. https://doi.org/10.1109/cvcbt.2018.00011