SymBChainSim

Diamantopoulos, George; Bahsoon, Rami; Tziritas, Nikos; Theodoropoulos, Georgios

DOI:
10.1145/3573900.3591121

Citation for published version (Harvard):
Diamantopoulos, G, Bahsoon, R, Tziritas, N & Theodoropoulos, G 2023, SymBChainSim: A Novel Simulation Tool for Dynamic and Adaptive Blockchain Management and its Trilemma Tradeoff. in M Loper, D Jin & CD Carothers (eds), *SIGSIM-PADS '23: Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation.* PADS: Principles of Advanced and Distributed Simulation, Association for Computing Machinery (ACM), New York, pp. 118–127, SIGSIM-PADS '23, Orlando, Florida, United States, 16/06/23. https://doi.org/10.1145/3573900.3591121

SymBChainSim: A Novel Simulation Tool for Dynamic and Adaptive Blockchain Management and its Trilemma Tradeoff

Georgios Diamantopoulos
School of Computer Science
University of Birmingham
Birmingham, United Kingdom
and
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, China

Rami Bahsoon
School of Computer Science
University of Birmingham
Birmingham, United Kingdom

Nikos Tziritas
Department of Computer Science and Telecommunications
University of Thessaly
Lamia, Greece

Georgios Theodoropoulos*
Department of Computer Science and Engineering and
Research Institute for Trustworthy Autonomous Systems
Southern University of Science and Technology (SUSTech)
Shenzhen, China

ABSTRACT

Despite the recent increase in the popularity of blockchain, the technology suffers from the trilemma trade-off between security decentralisation and scalability prohibiting adoption, and limiting the efficiency and effectiveness of the induced system. Addressing the trilemma trade-off calls for dynamic management and configuration of the blockchain system. In particular, choosing an effective and efficient consensus protocol for balancing the trilemma trade-off when inducing the blockchain-based system is acknowledged to be a challenging problem given the dynamic and complex nature of the blockchain environment. DDDAS approaches are particularly suitable for this challenge, and in previous work, the authors presented a novel DDDAS-based blockchain architecture and demonstrated that it offers a promising approach for dynamically adjusting the parameters of a system and optimising for the trade-off. This paper presents a novel simulation tool that can support and satisfy the DDDAS requirements for a dynamically re-configurable blockchain system. The tool supports the simulation and the dynamic switching of consensus protocols, analysing their trilemma trade-off. The simulator design is modular and allows the implementation and analysis of a wide range of consensus protocols and their implementation scenarios, along with the ability for parallelization. The paper also presents a quantitative evaluation of the tool.

CCS CONCEPTS

• **Computing methodologies** → **Simulation tools**; • **Computer systems organization** → *Peer-to-peer architectures.*

*Corresponding Author

KEYWORDS

Blockchain, Infosymbiotic Systems, DDDAS, Digital Twin, Simulation, Optimisation

ACM Reference Format:
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 *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '23), June 21–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3573900.3591121

1 INTRODUCTION

Blockchain has seen a huge leap in popularity since its inception as an immutable, decentralized ledger used by Bitcoin [24] and the plethora of other applications that soon followed. Blockchain has been increasingly utilized in a wide range of applications including IoT, supply chain systems, e-government systems, medical databases and more recently metaverse type applications [2, 9, 15, 21, 23, 32, 33]. The potential of Blockchain technology to support sustainable development is also increasingly being acknowledged, while tokenization is viewed as the key technology to promote and power ESG, impact investment, and sustainable finance [14, 30].

Despite the widely acknowledged potential of Blockchain, current limitations are holding the technology from achieving a wider spread adoption. Blockchain's decentralised design, naturally, puts the system at a performance disadvantage as compared to traditional centralised systems. Additionally, the so-called Trilemma trade-off in Blockchain-based systems (coined by Ethereum's Vitalik Buterin), states that in blockchain one cannot improve one of the three attributes (namely decentralization, scalability, and security), without affecting one or both of the others, further complicates the optimisation process.

The consensus protocol is at the heart of every blockchain system, dictating a strict procedure used by the nodes of the system to agree upon and update a single, decentralised system state. Currently, as concluded by Bamakan in a recent review [5] studying

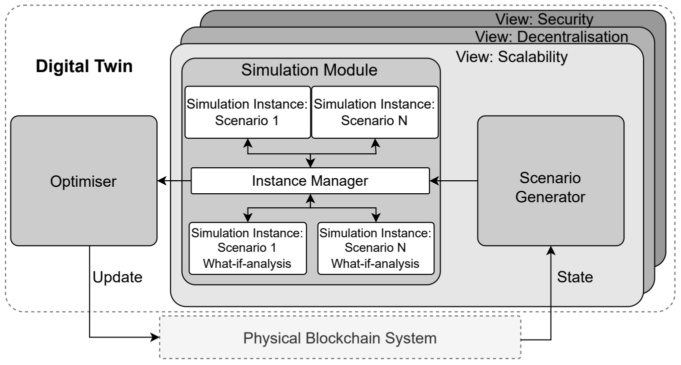

View: Security
View: Decentralisation
View: Scalability
Digital Twin
Simulation Module
Simulation Instance: Scenario 1
Simulation Instance: Scenario N
Optimiser
Instance Manager
Scenario Generator
Simulation Instance: Scenario 1 What-if-analysis
Simulation Instance: Scenario N What-if-analysis
Update
State
Physical Blockchain System

Figure 1: DDDAS-based blockchain management architecture as proposed in [11].

the state-of-the-art consensus protocols, there is no one-size-fits-all protocol. The above, combined major role of the consensus in affecting system performance, can be used to paint a picture of the trilemma trade-off. The challenge of creating general protocols has led to an influx of specialised ones aiming to offer a specific performance under specific system conditions to match the needs of various application cases[8, 18, 19, 26].

The extremely challenging, if not impossible, task of creating general consensus has created a need for alternative solutions [16]. The plethora of specific consensus protocols, in combination with the fact that blockchain systems support and are influenced by dynamic changes to the system parameters during runtime, hint towards dynamic management systems for blockchain.

In [11] the authors introduced a Digital Twin system for the dynamic management of blockchain systems (Figure 1). Specifically, the objective is to dynamically assist in managing and optimizing the Trilemma tradeoff in Blockchain-based systems through the utilisation of Digital Twins. A Digital Twin is a "combination of a computational model and a real-world system, designed to monitor, control and optimize its functionality" [6]. A Digital Twin is therefore fundamentally a Dynamic Data-Driven Application System (DDDAS), wherein a real-time info-symbiotic feedback loop between the model and the real system allows data from an observed system to be absorbed into a simulation of the system to continually adapt the model to the reality, and if necessary, making changes to the assumptions on which it is based to gradually increase the reliability of its forecasts. Additionally, the simulation's predictions can be fed back to the observed system to change or optimize its behaviour in real-time and direct the data collection and sampling [10].

At the core of the proposed Digital Twin architecture for blockchain system (Figure 1) is a simulation component that explores what-if scenarios for evaluating the effect of various systems parameters which act as the basis for optimisation. A fast and modular simulation tool, capable of simulating a wide variety of blockchain architectures, consensus protocols, and network environments, while allowing for the dynamic switching of system parameters is paramount for designing and testing dynamic blockchain management frameworks. For a DDDAS-based approach, a key feature of such a simulation tool is a mechanism that would allow for the change of system parameters during run-time. Modularity is another important feature. Dynamic blockchain optimisation can target many aspects, from consensus parameters, incentive algorithms, and network parameters such as broadcast algorithms and consensus groups (sharding), to malicious node voting strategies. Due to the impracticality of creating a tool supporting all the possible optimisation options for all aspects, modularity is key, allowing easy configuration of the desired system architecture while keeping the base simulator less bloated. Finally, in order for real-time optimisation to be effective, the timings of the decisions are critical. In order for the tool to be viable for use in systems such as the one proposed in [11] fast and efficient simulation is required. This paper presents an effort towards developing such a simulation system. Specifically, the blockchain simulation tool SymBChainSim is proposed, offering the following novel features:

- Support for dynamic updates of blockchain parameters and consensus protocol during runtime
- A modular architecture allowing for easy integration of new protocols support for simulation of various blockchain architectures
- The ability to form a real-time connection with a physical blockchain system

The rest of this paper is structured as follows: Section 2 provides an overview of the literature on blockchain simulation and published simulation tools. Section 3 describes the high-level architecture and implementation specifics of SymBChainSim. Section 4 demonstrates a use case scenario of SymBChainSim along with experimental results exploring the overhead of dynamic consensus switching. Finally, section 5 summarises this paper and briefly presents our plan to further expand SymBChainSim.

2 RELATED WORK

With cryptocurrencies being the first and, as of now, the most widespread application of blockchain, it is natural that the vast majority of blockchain simulators focus on modelling the major applications in this field. In [25] a survey was conducted studying the state-of-the-art simulation tools which concluded "...there is no universal simulator that could be applied to a wide range of scenarios." and additionally that "The capabilities of existing simulators are limited as they are designed to simulate a few critical aspects ... keeping the rest simplified".

The simulation tool at the heart of a DDDAS system must support the following (a) the ability to simulate a plethora of scenarios, (b) offer a wide range of optimisation options and the ability to change between them during run-time (c) the ability to simulate and offer control over the 5 major aspects of a blockchain system (d) the ability to simulate in faster than real-time. The rest of this section provides a brief overview of a selection of the most influential blockchain simulation tools, highlighting their scope and evaluating their design from the point of view of integration with a DDDAS system. For more information on blockchain simulators, the reader is referred to [3, 25] which dive deeper into the literature and present a more complete picture of the existing tools.

In [28] Faria proposes BlockSim, a blockchain simulation tool, focused on modelling the Proof-of-Work consensus protocol and the Bitcoin and Ethereum [7, 24] blockchain architecture. BlockSim

Name	Language	Architecture	Consensus	Dynamic	Node Behavior	Parallelizable
BlockSim (Faria) [28]	Python (SimPy)	Bitcoin Etherium	PoW	No	No	No
VIBES [29]	Scala	Generic Permissionless	PoW	No	Limited	Yes
SimBlock [4]	Java	Bitcoin Etherium	PoW	No	No	No
Talaria [31]	Python (SimPy)	Bitcoin, Etherium Permissioned	PoW, PoA, Pos, PoET, pBFT	No	Limited	No
SymBChainSim	Python	Generic Permissioned	PBFT BigFooT	Yes (Consensus + Parameters)	Benign + Byzantine faults	Yes

Table 1: Existing Blockchain Simulators Feature Comparison.

has been a very popular tool used to support research on cryptocurrency and has functioned as a base for a plethora of other simulation tools (Talaria, BlockPerf, CBlockSim, SegWit and more). The tool is written in python, utilising python's discrete-event simulation library SimPy [22]. BlockSim's high-level modelling allows for high scalability easily supporting tens of thousands of nodes. To achieve the above, BlockSim sacrifices the ability to simulate transaction generation, has a simple network model and does not support node behaviour simulation. Additionally, in BlockSim's implementation, the node architecture is determined by the consensus protocol, complicating the process of adding support for new protocols and hurting modularity. Finally, BlockSim does not support dynamic parameter changing, which is critical for use in DDDAS or parallelization.

SimBlock [4] is a blockchain simulator written in Java focused on modelling blockchain networks. SimBlock models a generic permissionless blockchain using the Proof-of-Work consensus protocol. SimBlock does not support a dynamic change of events and has low modularity due to it not being a goal of the tool.

VIBES [29] is a cloud-based blockchain Simulation Tool, which focuses on scalability and speed. It models a generic PoW blockchain and allows for some limited malicious node behaviour simulation (as a percentage of certain actions failing) but does not allow for dynamic changes in the system parameters, support for additional consensus protocols, and permissioned blockchain support.

Talaria [31] is, to the best of our knowledge, the only tool supporting the simulation of permissioned blockchain systems (supporting both a pBFT and a PoA consensus protocol). It is based on the BlockSim simulator and thus suffers from some of the same problems, i.e., requiring a different node architecture for each protocol which hurts modularity and complicates the process of adding support for dynamic parameter changes (specifically for the consensus protocols) a feature which is not naively supported.

Evidently, the existing blockchain simulation tools specialise in modelling of single aspects of the blockchain (behaviour of a specific protocol, network, incentives, security etc.) to explore and understand their characteristics and effects on the blockchain system. As a result, support for dynamic updates, and focus on the extensibility and generality of the tool are of the scope and thus not supported.

3 THE PROPOSED DYNAMIC DATA DRIVEN BLOCKCHAIN SIMULATOR (SYMBCHAINSIM)

SymBChainSim follows the approach used by most blockchain simulation tools, modelling the blockchain system in layers defined in [13]. This layer-based modelling allows for a modular design, where each layer can be easily replaced, extended or abstracted with little to no change to the others. Specifically, these layers include Application, Execution, Data, Consensus and Network with Application referring to a model of the underlying application blockchain is used in, Execution, a model of the execution of transactions and smart contracts, Data, a model of the blockchain, Consensus, a model of the consensus process and finally Network, a model of the underlying Peer-to-Peer (P2P) network and the nodes participating in it.

3.1 Layers of SymBChainSim

3.1.1 Application Layer. With the goal of SymBChainSim being to model a generic permission blockchain, the application layer was kept at a high level, modelled by a set of parameters shown in Table 2 Transactions in SymBChainSim are generated in intervals (TGI), based on the current values of the parameters (TPS and Ts) using a "transaction factory" as opposed to individual nodes generating transactions. Specifically, a "generate transactions" event invokes the transaction factory to generate transactions for the next TGI. The above parameters can either be used as static values, as the mean to an exponential distribution which is a common way of modelling random events (in the case where the application layer is specified) or used as part of a more sophisticated distribution modelling the transaction production trends of a specific application allowing for the modelling of time-variability in workloads with various resolutions and additionally the ability to form a real-time connection with a physical blockchain through a feedback loop dynamically updating the simulation according to incoming data. In the future, the creation of transaction generation models using existing datasets from IoT, Supply Chain, Smart Grid etc... will be explored for the creation of application-specific scenarios. Finally, the application layer also models the behaviour of nodes through the behaviour module which is discussed in detail in section 3.2.6.

3.1.2 Execution Layer. The execution layer of SymBChainSim models the block creation and validation times as well as the message validation times as parameters (Table 2). In the future, the execution layer will be extended to include support for smart contracts.

3.1.3 Data Layer. The data layer of SymBChainSim models the blocks and the block generation process by governing the block size and the minimum interval between two successive blocks (block time)

3.1.4 Consensus Layer. The consensus protocol is the "heart" of the blockchain system and a major factor affecting the performance of the blockchain [5]. While a common approach amongst other simulation tools is to abstract the consensus protocol model, blockchain management through DDDAS requires accurate information about the consensus protocol for decision-making. SymBChainSim takes the approach of simulating vote-based protocols at the "message level", i.e., every message exchange is modelled as an event in the system. The above approach is able to capture the complexity that arises from messages arriving out-of-order, or dropped messages, as well as the effect of various P2P algorithms e.g gossip on the performance of protocols which more abstract models fail to capture.

Another key metric in the design of the consensus model was modularity. As mentioned, a simulator focused on blockchain management should be designed to be extensible to allow for a wide range of algorithms and protocols to be considered in the optimisation process and to apply to a wide range of systems. In SymBChainSim the consensus protocols are standalone modules "agnostic" to the rest of the system. Additionally, the nodes are designed to be generic allowing for a set of nodes to freely switch between protocols and even use many protocols at once.

To keep the consensus layer standalone, each consensus protocol implemented protocol in SymBChainSim must also define a state variable, an instance of which is given to each node participating in the protocol (keeping the nodes generic), and the structure of blocks created by the protocol (keeping the data layer generic).

3.1.5 Network Layer. In SymBChainSim the network layer models the geographical location of nodes and the network graph and is used to determine (partially or fully) the execution time of events that model messages between nodes and the propagation of messages in P2P protocols such as gossip.

The parameters used to define the network can be seen in Table 2 with $B-MS$ denoting the base message (headers, signatures etc..), $P2P-P$ denoting the transmission protocol (broadcast, gossip), LC, a boolean value controlling the use of geographical location and consequently latencies, since the geographical location of nodes determines the latency between nodes and Processing Delay and Queuing delay the processing and queuing delay of messages.

Layer	Parameters
Application	Transactions Per Second (TPS)
	Transaction Size (Ts)
	Transaction Generation Interval (TGI)
	Behaviour Parameters
Execution	Block Creation Time (BCT)
	Block Validation Time (BVT)
	Message Validation Time (MVT
Data	Block Size (BS)
	Block Time (BT)
Consensus	Protocol Specific Parameters and Behaviour
Network	Base Message Size (B-MS)
	P2P protocol (P2P-P)
	Location (LC)
	Processing Delay(P)
	Queuing Delay(Q)

Table 2: SymBChainSim parameters per layer.

3.2 Architecture of SymBChainSim

SymBChainSim is composed of modules, modelling aspects of the blockchain system or implementing simulation logic. Specifically, the *Simulation*, *Node*, *Block*, *Transaction*, *Transaction Factory*, *Network*, *Consensus*, *Rounds*, and *HighLevelSync* model the blockchain following the 5 layers approach while *Event*, *EventQueue*, *Scheduler*, *Behaviour*, *Manager*, and *Parameters*, contain simulation logic. The complete architecture can be seen in Figure 2. Python was used for the implementation of the tool to allow for easy integration with a DDDAS system (itself developed in Python). Furthermore, Python's excellent library of scientific programming packages also simplifies analysis and optimisation using machine learning techniques. Finally, Python offers flexibility when connecting the simulator with a physical system, simplifying the development of interfaces. [1]

3.2.1 The Simulation, Node, Block and Transaction Modules. The simulation module includes the Nodes taking part in the system and the System Queue and is responsible for implementing the simulation loop logic. Within SymBChainSim, each Node is equipped with a range of components, including a local copy of the blockchain, a transaction poll that holds unprocessed transactions, a data structure for storing state and behaviour information that is updated in accordance with the current consensus protocol and behaviour module, a scheduler that interfaces with the network module, and a set of event queues (one for consensus and one for syncing messages), as well as a backlog queue that stores future messages to be processed at an appropriate time. The block module models a generic block structure containing a unique hash, a reference to the hash of the previous block, a list of validated transactions as well as an extra data field used by the consensus to add functionalities to the block. Finally, the Transaction module models transactions between nodes generated by the Transaction Factory based on the workload generation parameters. The nodes, block and transaction module were kept general, only containing core blockchain functionalities. The above keeps the simulator general and simplifies the process of adding additional logic for specialising the simulated system.

3.2.2 The Network Module. The Network module models the underlying P2P network using the parameters described in section

[1]An implementation of SymBChainSim can be found on GitHub (link) open-sourced under the MIT license.

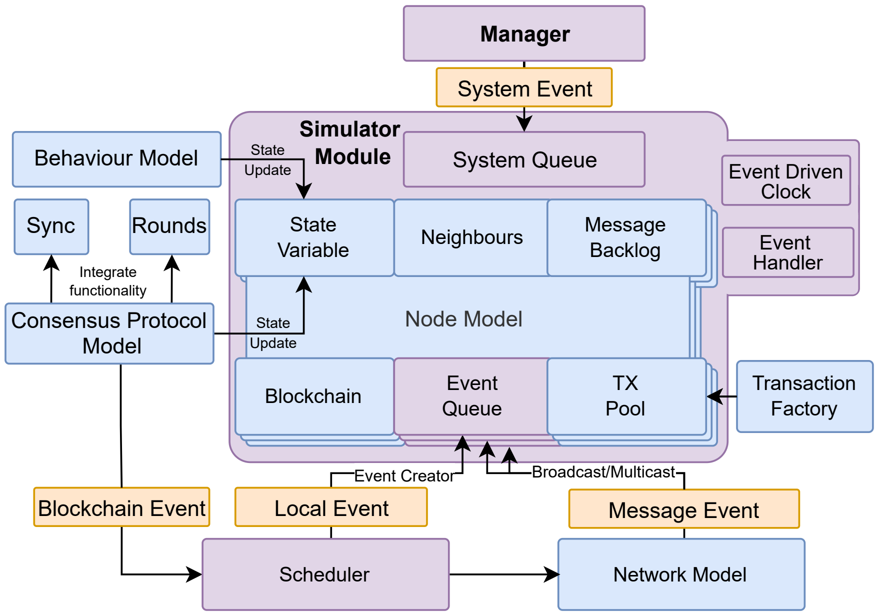


Figure 2: The architecture of SymBChainSim. Blue for blockchain models, purple for simulation components and orange for events.

3.1.5. Specifically, to calculate the network delay of a message $M^{s\rightarrow r}$ where s denotes the sender node and r the receiver node, the following formula is used:

$$Delay(M^{s\rightarrow r}) = L(s\rightarrow r) + T(M^{s\rightarrow r}) + P_r(M^{s\rightarrow r}) + Q_r(M^{s\rightarrow r})$$

were $L(s \rightarrow r)$ denotes the latency between the sender (s) and the receiver (r) obtained from [1] or calculated using distance through the method described in [17]. Transmission delay, $T(M^{s\rightarrow r})$, is defined as:

$$T(M^{s\rightarrow r}) = \frac{size(M^{s\rightarrow r})_{Mb}}{min(BW^{s}_{Mb/s}, BW^{r}_{Mb/s})} \qquad (1)$$

where $BW^{s}_{Mb/s}$ and $BW^{r}_{Mb/s}$ denote the bandwidths of s and r respectively. Finally, $P_r(M^{s\rightarrow r})$ and $Q_r(M^{s\rightarrow r})$, denote the processing and queuing delays of the message as defined by the receiving node r.

The current version of SymBChainSim assumes nodes have symmetrical upstream and downstream bandwidth but the logic to support asymmetrical bandwidth is trivial, simply changing equation 1 to:

$$AsymT(M^{s\rightarrow r}) = \frac{size(M^{s\rightarrow r})_{Mb}}{min(U\_BW^{s}_{Mb/s}, D\_BW^{r}_{Mb/s})}$$

where $U\_BW^{s}_{Mb/s}$, denotes the upstream bandwidth of the sender in Mb/s and $D\_BW^{r}_{Mb/s}$, the downstream bandwidth of the receiver in Mb/s. Through the available parameters the network module can model a range of network environments with various broadcasting protocols being able to be 'plugged-in' increasing the modularity of the simulator.

3.2.3 *The Consensus Module.*

The consensus module models the consensus process through which the nodes of the system propose and vote for new blocks. The consensus protocol performance characteristics play a major role in defining the performance of the overall system and thus much effort has been put into ensuring SymBChainSim can simulate protocols accurately at the lowest level. In SymBChainSim consensus, protocols are simulated at the message level and the module is highly decoupled from the rest. The above, make implementing new protocols or porting existing protocols to SymBChainSim very straightforward. Decoupling the consensus from the node architecture is also critical for simplifying the dynamic consensus updates. Tools such as [28], which couple node architecture with consensus limit their support for dynamic consensus updates with each new protocol requiring the definition of a new node and the update algorithm requiring the creation of new nodes whose state must be updated to match the current state of the system.

In SymBChainSim, every consensus protocol is implemented as a Python module and must contain the following (a) a "set_state" and "init" method, which initialise the state variable of a node with the required fields as defined by the consensus protocol and begin the consensus process (b) a "create_block" method which defines the structure of a block produced by the nodes following this consensus protocol (c) a "resync" method which defines any actions a node must take after receiving it's missing blocks, in order to start participating in the consensus process [2] (can be left empty if no specific actions are required) and, (d) a "handle_event" method capable to handle every type of event that is produced by this module.

[2]only if HighLevelSync is used as the resync algorithm.

3.2.4 *The Rounds and HighLevelSync Module.*

Since it is common for vote-based consensus protocols used in permissioned blockchains to work in consensus rounds SymBChainSim includes a rounds module by default to be used in modelling consensus protocols that require it. The rounds module supports both a "sticky proposer" and a "rolling proposer", and both a "round-robin" and "hash-based" proposer selection algorithm where the next proposer is selected as:

$$\text{proposer} = \begin{cases} R\%N & \text{if round-robin} \\ Hash(B_{latest})\%N & \text{if hash-based} \end{cases}$$

where R denotes the current round, N is the number of nodes, and B_{latest} is the latest block in the blockchain. In order for a consensus protocol to use the Rounds module the Rounds state must be included in the consensus state variable. Additionally, the consensus protocol should contain an *init_round_change* method which defines any protocol-specific actions needed to be taken before a node goes into the *round change* state. With the above requirements satisfied, the methods defined in the Rounds module can be called as required by the protocol.

De-synced nodes are a very common occurrence in blockchain and thus a syncing algorithm is required. To increase efficiency, SymBChainSim implements a high-level syncing algorithm which can be used by any Consensus protocol and whose logic is defined in HighLevelSync(HLS). To use HSL a CP needs to define the re-sync method. When a node realises it has fallen out of sync, i.e., when receiving an accepted 'future' block and as defined by the CP it can invoke HLS to request the missing blocks from one of its peers.

3.2.5 *The Event, Event Queue, and Event Handler Modules.*

In SymBChainSim every action is modelled as an event. The Event module models the various events used throughout the simulator, namely, SystemEvent, LocalEvent, and MessageEvent which will be discussed in depth in section 3.3. In general, every event in SymBChainSim contains an execution timestamp (in simulation time), a reference to its specific handler, a payload and a unique ID. The Event Queue module contains a sorted list of upcoming events and a hash table storing past messages. Since the creation time of events does not necessarily match their execution time, a binary insertion sort algorithm is used to keep the event list sorted, increasing efficiency. Finally, the hash map stores the history of past events in ID: Event pairs, used in P2P multi-cast algorithms such as Gossip. In the SymBChainSim, any module that generates events must also contain an associated handler that manages those events and thus the general event handler module is only responsible for performing preliminary checks, such as verifying that the node is online, before invoking the specific handler if the current event. Additionally, based on the response of the module-specific handler, specifically, in the case where the processed event changed the state of the node - the handler will try to execute messages stored in the backlog (possible responses can be seen in Table 3).

Return Value	Description
handled	Event was handled successfully
new state	Event was handled and changed state check backlog
invalid	Invalid message (old, corrupted etc..)
backlog	future message, add to the backlog
unhandled	Could not handle (error message)

Table 3: Possible return values of event handlers.

3.2.6 *The Scheduler and Behaviour Modules.*

In SymBChainSim the scheduler module acts as the interface between the nodes of the system, the network and the event queues. Each node contains a reference to the scheduler, through which modules such as consensus, rounds and sync, can access the "scheduler local event" and "schedule message event" functionalities to model logic which requires interaction with the other nodes. The behaviour module models the behaviour of nodes in the simulation. SymBChainSim architecture allows for easy parallelization, only requiring the modification of the scheduler module into a distributed scheduler module. Specifically, each process would contain an instance of the simulation module with a subset of the nodes and an instance of the distributed scheduler. The distributed scheduler will be in charge of forwarding local events to the schedulers of other processes and receiving events from other processes with the rest of the modules being 'parallelization agnostic' i.e., their functionality does not require change based on whether the simulation is running in parallel or not.

The behaviour of nodes plays a significant role in determining the system's performance. In blockchain, nodes can behave as *honest*, *faulty* or *byzantine*. Honest nodes follow the protocol to the best of their ability, processing all messages and responding accordingly and as fast as possible. Faulty nodes are honest nodes which experience random faults and thus are not able to participate in the consensus process until they recover from the fault. Finally, byzantine or "malicious" nodes are nodes which deliberately diverge from the protocol by deliberately delaying replies, not replying at all, replying with intentionally false data, or sending diverging information to different nodes. The behaviour characteristics of nodes can drastically change the performance of the system and this, node behaviour simulation is of high importance. The Behaviour module is in charge of modelling the behaviour of nodes through the behaviour state variable. In SymBChainSim, the behaviour state of a node defines whether the node is faulty, byzantine or both. Fault behaviour is modelled as random events using an exponential distribution. Specifically, each faulty node contains a "MeanTimeToFailure" and "MeanTimeForRecovery" governing the mean time between failures and the mean time for recovery if a failure occurs. The Byzantine behaviour model in the node only describes the misbehaviour in the Sync process since consensus protocols differ and thus consensus misbehaviour is better defined individually per protocol [3].

3.2.7 *The Manager and Parameters Modules.*

The manager module contains a reference to the simulation module and is responsible for managing the simulation. Through system events, the manager

[3]Only nodes who are marked as byzantine in their behaviour state can misbehave in the consensus process

modules can influence the parameters of each layer, add and remove nodes, change the consensus protocol, update the behaviour of nodes and change the workload. The manager is a key component of the simulator and allows for novel features such as dynamic updates during runtime and can function as the link between the simulator and a physical blockchain system, either directly or indirectly through a DDDAS such as a Digital Twin. When the simulator is running as "standalone", i.e., not interfacing with a physical system, the manager begins the simulation by loading the parameters from the configuration files and initiates the simulation module and network module. Before beginning the main simulation loop the manager initialises the System Queue, adding the first "generate transactions", "apply behaviour" and "update behaviour" events. In parallel execution, a single instance of the manager can forward system events to all the simulation processes lowering the parallelization complexity.

Finally, the Parameters module contains a list of all of the system parameters. It is initialised by the manager at the start of the simulation and can be updated in real-time through system events. Every module in the system contains a reference to the parameters module and thus changes in the parameters are reflected system-wide.

3.3 Events in SymBChainSim

3.3.1 Event Types. SymBChainSim takes an event-driven approach for the simulation of the blockchain system and thus every action in the simulation is modelled as an event. Specifically, two types of events are defined, "System Events" *SE* and "Blockchain Events" *BE* with the latter being split into "Local Events" *LE* and "Message Events" *ME*. A System Event is the simplest type of event in SymBChainSim only consisting of a timestamp and a payload. SEs are produced by the manager and are added to the System Queue. The primary focus of SE is to model events that manage and update the simulation with workload generation, behaviour updates (changing byzantine nodes, faulty nodes and the specifics of their behaviour) and application (random faults and recovery) updating the simulation parameters and the consensus protocol. Blockchain events, on the other hand, are designed to model blockchain-specific events. The structure of LocalEvents and MessageEvents is similar, consisting of a unique ID, a reference to a handler method capable of executing the event, a timestamp (in simulation time), a payload which contains the type of event and the necessary information for its execution, a reference to the creator node and the receiving node (only in MessageEvents) and an actor variable, which is a reference to the node that this event refers to (creator for local events/receiver for message events). The inclusion of the actor variable reduces the complexity of handling BE events, by removing the need to constantly check the type of the event in order to select the appropriate node during handling.

3.3.2 Event Execution. Multiple queues exist in SymBChainSim, each containing different types of events. Specifically, each node contains 2 event queues, a consensus queue containing consensus-related MEs and LEs and a Sync queue containing sync-related MEs and LEs. Additionally, the simulator module contains the System Queue which stores the various SEs produced by the Manager. The vast majority of events in SymBChainSim are consensus-related due to the low-level modelling of the algorithms. Additionally due to the various delays added in simulation time (network, processing etc..) generated events cannot be executed using a simple FIFO approach and need to be sorted based on execution time (in simulated time). To achieve the above the event queues were modelled as a list with new events being inserted using the binary sort insertion algorithm whose time complexity is $O(log_2 E)$ where E is the number of events currently in the list. By maintaining a sorted list, the complexity of retrieving the next event of a queue is O(1). From the above, it is, trivial to see that since insertion is the most computationally expensive operation, and its complexity is based on the number of events in the queue, keeping multiple event queues can significantly reduce the computational complexity of insertion, while only slightly increasing the computational complexity of calculating the next event thus making the use of more queues beneficial.

To implement the simulation loop, the simulator simply retrieves the next event of each online node (sync or consensus) and keeps the earliest one. The next system event to be executed is also retrieved from the system queue and is compared with the earliest blockchain event. The earliest of the two events (SE and BE) is then executed.

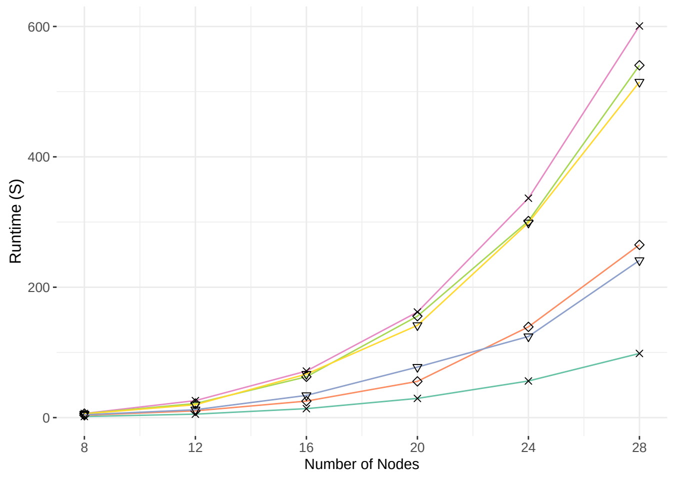
3.4 SymBChainSim for Dynamic Management of Blockchain Systems

SymBChainSims Manager module is designed to function as the interface between the simulation, and a physical blockchain system either directly or indirectly as part of a DDDAS architecture such as the one in Figure 1. Through this interference, the DDDAS system can update the state of the simulation to reflect changes in the real system. Real-time workload data, network data, and behaviour data can be fed to the manager, which, in turn, will produce the relevant events to update the simulation. The specifics of implementing a real-time DDDAS system and creating a feedback loop with a blockchain system are out of the scope of this paper. For a more detailed explanation of a DDDAS-based architecture used for the dynamic management of blockchain and the specifics of creating a feedback loop with blockchain systems, we prompt the reader to [11]. Additionally, what-if analysis can occur through the manager by cloning the current simulator state, and simulating instances with various parameters to study their effects on the system. The outputs of the simulation can then be used to make real-time optimisation decisions or to assist in the training process of intelligent optimisers such as the one proposed in [12] or in [20].

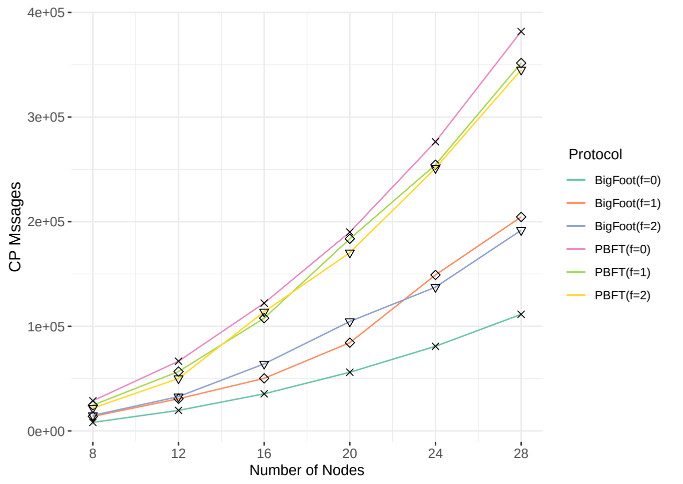
4 EXPERIMENTAL EVALUATION

4.1 Use Case: Profiling PBFT and BigFoot using SymBChainSim

4.1.1 Experimental Setup. With consensus being one of the main factors affecting the performance of a blockchain system, understanding the performance characteristics of protocols is critical in both the design and optimisation of blockchain systems. SymBChainSim is enabled with a low-level consensus model which can be employed to accurately profile the performance of protocols, under various network, workload and application environments, providing useful insights. In this example use case, SymBChainSym was used to evaluate the performance of PBFT and BigFoot under the presence of faulty nodes. Network, workload and application



(a)



(b)

Figure 3: Evolution of runtime (a) and messages sent (b) as nodes increase for PBFT and BigFoot with no faulty nodes,(f=0) one (f=1) and two (f=2) faulty nodes.

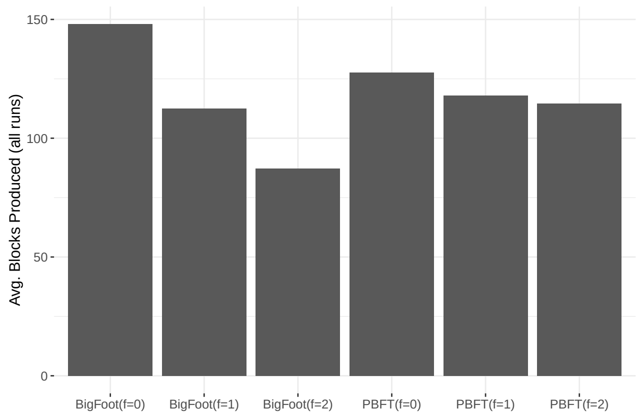


Figure 4: Average blocks produced by each CP across all runs with no faulty nodes,(f=0) one (f=1) and two (f=2) faulty nodes.

variations over time, as well as byzantine behaviour, were switched off thus making faulty nodes the only parameter affecting performance. Specifically, PBFT and BigFoot instances of SymBChainSim with 8, 12, 16, 20, and 28 nodes were used, each simulated for 10 minutes (simulation time) with 0, 1 and 2 faulty nodes (f). The PBFT and BigFoot protocol implementations are based on [8] and [27] respectively. Specifically, PBFT is designed to be robust to node faults and byzantine behaviour but suffers from low scalability, while BigFoot, is designed to be more scalable and less robust to faults. The messages and runtime of each simulation, along with the average blocks produced depending on the number of faulty nodes are used to evaluate the performance and scalability of the protocols. The above instances were executed on a 2021 Apple MacBook Air, equipped with the base Apple silicone M1 chip and 8GB of memory. The results of the above simulation runs are shown in Figures 3 and 4.

4.1.2 Scalability Results. The results obtained from the above experiments confirm that PBFT is indeed more robust than BigFoot but less scalable. Specifically, figures 3a and 3b clearly show that PBFT scales worse than BigFoot which is reflected in its runtime and messages growing at an exponential rate (consistent with the algorithm's theoretical scaling). Bigfoot, on the other hand, scales better as nodes increase, but, in contrast to PBFT, its scalability gets worse as soon as faulty nodes appear, confirming its poorer robustness. Interesting to note here is that as nodes increase, the runtime of PBFT reduces (Figure 3a) which entails fewer blocks produced (see Figure 4). The above is justified by the fact that the completion of a round from an online proposer takes more time than that of an offline proposer. This is due to the fact that a round with an offline proposer, results in a failure, and thus in fewer message exchanges and less runtime. On the other hand, in BigFoot when an offline node was part of the system, both runtime and messages increased and block throughput significantly decreases. The above is justified as a result of the fast-path stage failing (a detailed explanation of the fast-path state and the inner workings of BigFoot can be found in [27]). In the above case, the runtime of the simulation is highly correlated with the number of messages produced by the consensus process and thus the performance of the consensus protocol itself. The above is utilised to study the performance characteristics of a consensus through the runtime of the simulation. The above also affects the ability of the simulator to simulate faster-than-real-time, as shown in Figure 3a, SymBChainSim can simulate BigFoot at a fraction of real-time with 28 nodes but fails to do so with PBFT, a more robust algorithm relying on a more complex message exchange protocol. This limited scalability is a result of the low-level modelling of events and can be significantly improved through parallelization. Finally, figure 4 shows the average blocks produced by each protocol based on the number of faulty nodes in the system further supporting the above. From the results shown in figure 4, an interesting observation can be drawn in the context of blockchain optimisation through dynamic CP switching: to maximise block throughput, BigFoot can be employed when no offline nodes are present while switching to PBFT once offline nodes are detected.

4.2 Dynamic Consensus Switching Overhead

4.2.1 Experimental Setup. Besides the optimisation gains, the overhead added to the system by an optimisation approach is a critical factor. Specifically, in blockchain, due to its asynchronous nature, the process of switching protocols can result in various overheads. Thus, studying the potential overheads of switching protocols can aid our understanding of dynamically managing blockchain systems.

To better understand why blockchain's asynchronous nature can cause overheads, let's consider the following case, in which, a set of nodes are first to update their CP while the rest of the nodes are still "behind" due to network delays. Since the update mechanism does not operate in a "drop everything and switch" fashion, nodes have to finish the current proposal round in order to update. Thus, it is possible for a subset of nodes that are "ahead", to start a new round before the update and the remaining nodes to start a round after. In the above case, given that the subset of nodes with the new protocol is less than the super-majority required for consensus, the consensus process will be halted until the remaining nodes, with the old CP, fail a round, and update. The above results in an overhead equal to the time it takes for a round of the old protocol, to be declared failed.

Measuring the overhead introduced by switching protocols is not a straightforward process. Due to the complexity of the system, identifying which delays are caused by the switch is challenging. Despite that, the overhead is only reflected in a short window after switching protocols, and thus, by measuring the idle time between the 2 blocks that are proposed right before and right after the switch, and comparing them to the usual delays, we can gauge the magnitude of the overhead. Specifically, we define idle time as the interval between the addition of the current block, denoted by CP_n, and the creation time of the subsequent block denoted by CP_{n+1}. According to the above, we define the following two metrics: (a) BP_{all} which denotes the idle time between every subsequent block produced and (b) BP_{switch} which donates the idle time between subsequent blocks accepted before and after a CP switch.

The above metrics were measured for 100 random simulations to derive the results shown in Figure 5. Specifically, 100 random instances of the simulator were initialised with 4, 8 and 12 nodes and run for 20 minutes (simulation time) with the CP protocol randomly changing approximately every 30 seconds. The parameters of the simulation were kept static and the behaviour and faults simulation were turned off to reduce the factors affecting the idle time. Additionally, because block size heavily affects block delays, the block size was kept fixed at 1MB with the goal of, again, reducing factors affecting idle time.

4.2.2 Overhead Results. The results of the above experiment are shown in figure 5. By studying the figure, one can see that, as

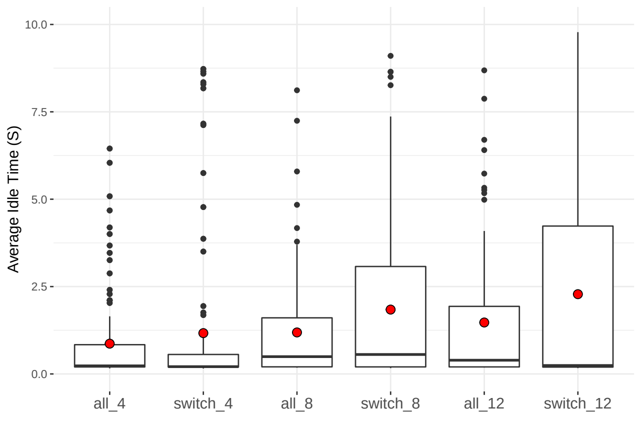


Figure 5: Average idle time between block proposal rounds for 4, 8 and 12 nodes. 'All blocks': average idle time between all blocks. "Switch blocks": average idle time between blocks before and after a CP switch. (The red dot denotes the mean value).

expected, some overhead does indeed occur [4]. Although, using a DDDAS to optimise a blockchain system can speed up transaction latency, (which is the metric most prominently affected by delayed blocks) up to several seconds, on average, per block [11] easily justifying the incurred overhead. Despite that, since switching protocols does come with a cost, one still has to be mindful of frequent switches.

5 SUMMARY AND FUTURE WORK

This paper has proposed SymBChainSim, a novel blockchain simulation tool aimed at enabling dynamic blockchain management specifically through approaches based on the DDDAS paradigm. In contrast to the current state-of-the-art blockchain simulation tools which are focused on modelling specific architectures and aspects of a blockchain system, SymBChainSim aspires to provide a generic, extensible and dynamic permissioned blockchain simulation system. SymBChainSim offers a novel dynamic parameter switching mechanism, is capable of interfacing with a blockchain system to reflect state changes in real-time, and provides what-if-analysis capabilities to be utilised by the DDDAS architecture. The experimental evaluation has confirmed that the overhead of consensus switching does not significantly diminish performance. Current limitations of the system include the low amount of consensus implementations and the lack of smart contract logic support. Additionally, the low-level modelling of events comes at the cost of scalability which although partially mitigates by parallelization, is not yet supported.

In the future, adding the above functionality is an integral part of the plan to improve the simulator and increase the range of blockchain architectures the simulator is able to model and ultimately

[4]note that for 4 nodes, the outlier values cause the mean idle time of "switch" to be higher than that of "all" despite the distribution of average idle times being seemingly better

manage. Furthure to dynamic blockchain management, SymBChain-Sim low-level models makes it a suitable platform for prototyping and testing new consensus protocols and blockchain systems. Additionally, its focus on node behaviour simulation can be utilised to create attacks used to evaluate the robustness of protocols and systems.

ACKNOWLEDGMENTS

This research was supported by: Shenzhen Science and Technology Program, China (No. GJHZ20210705141807022); SUSTech-University of Birmingham Collaborative PhD Programme; Guangdong Province Innovative and Entrepreneurial Team Programme, China (No. 2017Z T07X386); SUSTech Research Institute for Trustworthy Autonomous Systems, China.

REFERENCES

[1] 2023. AWS Latency Monitoring. https://www.cloudping.co/grid#

[2] Al-Jaroodi et al. 2019. Blockchain in Industries: A Survey. *IEEE Access* 7 (2019), 36500–36515. https://doi.org/10.1109/ACCESS.2019.2903554

[3] 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, 284–294.

[4] Aoki et al. 2019. Simblock: A blockchain network simulator. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 325–329.

[5] Seyed Mojtaba Hosseini Bamakan, Amirhossein Motavali, and Alireza Babaei Bondarti. 2020. A survey of blockchain consensus algorithms performance evaluation criteria. *Expert Systems with Applications* 154 (2020), 113385.

[6] Volker Buscher. 2019. Digital twin: towards a meaningful framework. Arup, Foresight, Research and Innovation.

[7] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014).

[8] Castro et al. 1999. Practical byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[9] Dai et al. 2019. Blockchain for Internet of Things: A Survey. *IEEE Internet of Things Journal* 6, 5 (2019), 8076–8094. https://doi.org/10.1109/JIOT.2019.2920987

[10] Darema et al. 2008. Dynamic Data Driven Applications Systems (DDDAS) – A Transformative Paradigm. In *Computational Science – ICCS 2008*, Marian Bubak, Geert Dick van Albada, Jack Dongarra, and Peter M. A. Sloot (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 5–5.

[11] Georgios Diamantopoulos, Nikos Tziritas, Rami Bahsoon, and Georgios Theodoropoulos. 2022. Digital Twins for Dynamic Management of Blockchain Systems. In *2022 Winter Simulation Conference (WSC)*. 2876–2887. https://doi.org/10.1109/WSC57314.2022.10015447

[12] Georgios Diamantopoulos, Nikos Tziritas, Rami Bahsoon, and Georgios Theodoropoulos. 2022. Dynamic Data-Driven Digital Twins for Blockchain Systems. In *Accepted for publication at 2022 Dynamic Data Driven Application Systems (DDDAS) 2022*.

[13] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1085–1100. https://doi.org/10.1145/3035918.3064033

[14] Freire et al. 2021. Harnessing Blockchain For Sustainable Development: Prospects And Challenges. United Nations Conference On Trade And Development, UNCTAD/DTL/STICT/2021/3.

[15] Gamage et al. 2020. A Survey on Blockchain Technology Concepts, Applications, and Issues. *SN Computer Science* 1, 114 (2020).

[16] Giang-Truong et al. 2018. A Survey about Consensus Algorithms Used in Blockchain. *Journal of Information Processing Systems* 14, 1 (2018).

[17] Rohitha Goonatilake and Rafic A Bachnak. 2012. Modeling latency in a network distribution. *Network and Communication Technologies* 1, 2 (2012), 1.

[18] Guerraoui et al. 2010. The next 700 BFT Protocols. In *Proceedings of the 5th European Conference on Computer Systems* (Paris, France) *(EuroSys '10)*. Association for Computing Machinery, New York, NY, USA, 363–376. https://doi.org/10.1145/1755913.1755950

[19] Kotla et al. 2007. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. 45–58.

[20] Liu et al. 2019. Performance optimization for blockchain-enabled industrial Internet of Things (IIoT) systems: A deep reinforcement learning approach. *IEEE Transactions on Industrial Informatics* 15, 6 (2019), 3559–3570.

[21] Maesa et al. 2020. Blockchain 3.0 applications survey. *J. Parallel and Distrib. Comput.* 138 (2020), 99–114. https://doi.org/10.1016/j.jpdc.2019.12.019

[22] Norm Matloff. 2008. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2, 2009 (2008), 1–33.

[23] Monrat et al. 2019. A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities. *IEEE Access* 7 (2019), 117134–117151. https://doi.org/10.1109/ACCESS.2019.2936094

[24] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[25] Remigijus Paulavičius, Saulius Grigaitis, and Ernestas Filatovas. 2021. A systematic review and empirical analysis of blockchain simulators. *IEEE access* 9 (2021), 38010–38028.

[26] Roberto Saltini. 2022. BigFooT: A robust optimal-latency BFT blockchain consensus protocol with dynamic validator membership. *Computer Networks* 204 (2022), 108632.

[27] Roberto Saltini. 2022. BigFooT: A robust optimal-latency BFT blockchain consensus protocol with dynamic validator membership. *Computer Networks* 204 (2022), 108632.

[28] Lyubomir Stoykov, Kaiwen Zhang, and Hans-Arno Jacobsen. 2017. VIBES: Fast Blockchain Simulations for Large-Scale Peer-to-Peer Networks: Demo. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos* (Las Vegas, Nevada) *(Middleware '17)*. Association for Computing Machinery, New York, NY, USA, 19–20. https://doi.org/10.1145/3155016.3155020

[29] Lyubomir Stoykov, Kaiwen Zhang, and Hans-Arno Jacobsen. 2017. VIBES: Fast Blockchain Simulations for Large-Scale Peer-to-Peer Networks: Demo. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos* (Las Vegas, Nevada) *(Middleware '17)*. Association for Computing Machinery, New York, NY, USA, 19–20. https://doi.org/10.1145/3155016.3155020

[30] Uzsoki et al. 2019. Impact Tokens A blockchain based solution for impact investing. International Institute for Sustainable Development.

[31] Xing et al. 2021. Talaria: A Framework for Simulation of Permissioned Blockchains for Logistics and Beyond. *arXiv preprint arXiv:2103.02260* (2021).

[32] Ynag et al. 2022. Fusing Blockchain and AI with Metaverse: A Survey. *arXiv preprint arXiv:2201.03201* (2022).

[33] Zheng et al. 2018. Blockchain challenges and opportunities: a survey. *International Journal of Web and Grid Services* 14, 4 (2018), 352–375. https://doi.org/10.1504/IJWGS.2018.095647 arXiv:https://www.inderscienceonline.com/doi/pdf/10.1504/IJWGS.2018.095647