FoBSim: An extensible open-source simulation tool for integrated Fog-Blockchain systems

Hamza Baniata and Attila Kertesz

Department of Software Engineering, University of Szeged, Hungary

Corresponding author:
H. Baniata

Email address: baniatah@inf.u-szeged.hu

ABSTRACT

A lot of hard work and years of research are still needed for developing successful Blockchain (BC) applications. Although it is not yet standardized, BC technology was proven as to be an enhancement factor for security, decentralization, and reliability, leading to be successfully implemented in cryptocurrency industries. Fog computing (FC) is one of the recently emerged paradigms that needs to be improved to serve Internet of Things (IoT) environments of the future. As hundreds of projects, ideas, and systems were proposed, one can find a great R&D potential for integrating BC and FC technologies. Examples of organizations contributing to the R&D of these two technologies, and their integration, include Linux, IBM, Google, Microsoft, and others. To validate an integrated Fog-Blockchain protocol or method implementation, before the deployment phase, a suitable and accurate simulation environment is needed. Such validation should save a great deal of costs and efforts on researchers and companies adopting this integration. Current available simulation environments facilitate Fog simulation, or BC simulation, but not both. In this paper, we introduce a Fog-Blockchain simulator, namely FoBSim, with the main goal is to ease the experimentation and validation of integrated Fog-Blockchain approaches. According to our proposed workflow of simulation, we implement different Consensus Algorithms (CA), different deployment options of the BC in the FC architecture, and different functionalities of the BC in the simulation. Furthermore, technical details and algorithms on the simulated integration are provided. We validate FoBSim by describing the technologies used within FoBSim, highlighting FoBSim novelty compared to the state-of-the-art, discussing the event validity in FoBSim, and providing a clear walk-through validation. Finally, we simulate two case studies, then present and analyze the obtained results.

1 INTRODUCTION

In light of the general tendency towards skepticism around Blockchain (BC) systems being reliable, huge research and industrial projects are being encouraged to address issues and vulnerabilities of those systems. This is because it is believed that a successful BC deployment would definitely advance the Internet-of-Everything (IoE) applications. Dubai, for example, has been planning for being the first smart city powered by BC [1]. China had launched, in late 2019, a BC-based smart city ID system [2], while it is planning to have its own official digital currency [3]. Before that, Liberstad, a private smart city in Norway, has officially adopted City Coin as its official currency [1].

BC is a Distributed Ledger Technology (DLT) in the form of a distributed transactional database, secured by cryptography, and governed by a consensus mechanism [4]. This technology was first introduced as the backbone of the Bitcoin ecosystem in 2009 [5]. As BC got high reputation and attention among research and industry communities, as well as governments, it has proven robustness against the disadvantages of classical centralized systems. Furthermore, different versions, uses, paradigms, and platforms were proposed, aiming to extend the deployment of BC beyond cash and payment purposes.

Concerning smart things, homes, and cities, Fog Computing (FC) paradigms become reality. FC is a horizontal, physical or virtual resource paradigm that resides between smart end-devices and traditional

[1] https://www.liberstad.com/

cloud data centers [6]. FC is conceptually an extension of the cloud at the edge of the network. Hence, most cloud services should be introduced by the fog layer as well, except the fog provides better latency measures.

Different reference architectures were proposed for the FC paradigm, e.g. by Habibi et al. [7], Dastjerdi et al. [8], the OpenFog consortium [9], and Cisco [10]. Nevertheless, they all have the same general properties of middling between end-users and the clouds, providing cloud services at the edge of the network, managing mobility issues, and introducing reliable and secure communications.

We have previously investigated the integration of BC with FC in [11]. Accordingly, we concluded that such integration may ease the optimization of several current Cloud-Edge issues, such as enhancing security, credibility, and resource efficiency. Also, decentralizing FC applications decreases the appearance of single points of failure and the control of a centralized authority. However, we found that major challenges still need more research efforts such as:

- The lack of individual standardization of both technologies, FC and BC, which leads to the lack of standardization of the integration of them.
- Many privacy issues and threats remain, such as the location awareness property of fog components, which raises some concerns.
- Ironically, as FC enhances the latency of end-user applications, BC causes the exact opposite, if the consensus mechanisms were not properly designed. Other major issues may also represent barriers if this latency issue was not addressed, such as authentication, scalability, and heterogeneity problems. This is because solving the latency problem may require waiving some advantageous protocols or mechanisms of FC.
- The aforementioned challenges may further lead to somewhat low Trust levels of the BC-FC integration, which is the main cause of the illegalization of BC technologies in general.

Consequently, the research and industry communities have been working hand-in-hand to solve these major challenges, along with other technical issues. Such efforts require reliable and flexible simulation environments that can mimic real-life scenarios with the lowest possible costs. Old, out-dated, or somewhat close simulation tools that were initially implemented for classical Peer-to-Peer networks, such as PeerSim [12], may not be able to cover all the mechanisms of a modern BC system. Although some recently proposed systems use PeerSim, such as [13], which required vast amount of changes, modifications, and additions to redesign it into a BC simulation tool.

In this paper, we propose a Fog-Blockchain simulation environment, called FoBSim, that is able to simulate different integration scenarios of FC and BC. Concerning our main contributions, we discuss and analyze the architectural elements of FC and BC based systems, and present the modules, algorithms, and strategies implemented in FoBSim. We also describe in detail the validation, the incentivization, and the confirmation mechanisms deployed in the current version of FoBSim. To exemplify its utilization, we discuss possible application scenarios of FC-BC integration, and we clarify how such applications can be simulated and optimized using FoBSim.

The main properties of the current version of FoBSim are as follows:

1. FoBSim provides different Consensus Algorithms (CA), namely PoW, PoS, and PoA that are ready to be deployed in any scenario.
2. FoBSim allows the easy deployment of the BC miners in the fog layer or the end-user layer.
3. FoBSim allows different services to be reliably provided by the BC network, namely Data Management, Identity Management, Computational Services (through Smart Contracts (SC)), and Payment/Currency transfer Services.
4. FoBSim provides both, parallel execution and non-parallel execution, of mining processing. While gossiping is conducted efficiently so that the distributed chain is consistent in different possible network topologies.
5. FoBSim is the first simulation environment that aims to mimic any integration scenario of FC and BC technologies.

The remainder of the paper is organized as follows: Section 2 presents and discusses the state-of-the-art simulation environments that are maybe suitable to simulate FC-BC systems. To properly introduce FoBSim, we discuss, in detail, how FC architectural elements are deployed in Section 3. Additionally, we discuss the categories of BC systems, each with its properties and components in Section 4. Accordingly, we propose the components, the algorithms, and the functions of the FoBSim environment in Section 5. To validate FoBSim, we simulate some use cases and present the simulation results in Section 6. Finally, we present our future work and conclude in Section 7.

2 RELATED WORK

Searching the literature for tools specifically implemented for simulating FC-BC integration scenarios, we found that no previous work has directly targeted our objective. That is, we found several simulation tools that mimic fog-enhanced cloud systems, IoT-Fog-Cloud scenarios, etc., and several tools that mimic BC scenarios, each with specific constraints on the used CAs. Nevertheless, some proposals for IoT-BC simulation tools can be somewhat related to our work. For example, the ABSOLUT tool, investigated in [14], models the deployment of BCs in IoT environments. Accordingly, some critical analysis were provided regarding network latency, effects of miners number on the overall efficiency of the IoT network, and simulation errors.

Liaskos et al. [15] proposed a general architecture that a BC simulation needs to follow in order to be considered comprehensive. Further, some properties were declared as necessary for encouraging the adoption and re-usability of the simulation. The proposed architecture includes extensible connection strategies, BC nodes, BC chains, Transactions and Transaction pools, users, events, Blocks, and most importantly Consensus mechanisms. Events can include different triggers to other events - that may be performed by any entity of the network - (such as transaction/block arrival, transaction/block validation, connection requests, etc.). Also, Events need to be handled by concise and well implemented strategies.

In light of the lack of simulation tools similar to our proposal, we found it more suitable to present this section in two separate groups: namely FC simulation tools, and BC simulation tools.

2.1 FC simulation tools

Recently, our research group has started to investigate the state-of-the-art related to cloud, IoT and fog simulation tools in [16]. Within this study, several simulation tools were classified, compared, and analyzed, such as the DockerSim tool [17], FogNetSim++ [18], and EdgeCloudSim [19]. Furthermore, technical details, advantages, vulnerabilities, and software quality issues were also discussed.

Rahman et al. [20] surveyed 15 simulation tools for cloud and data centers networks scenarios. The tools were discussed and compared according to several criteria, such as the Graphical User Interface (GUI) availability, the language with which the simulator was implemented, and the communications model. Consequently, they proposed the Nutshell tool which addresses some drawbacks that were ignored by most of the surveyed simulators. For example, most surveyed simulators had abstract network implementation and low-level details were missing. Further, non of the studied tools provided an addressing scheme, a congestion control mechanism, or a traffic pattern recognition mechanism. Out of those 15 presented simulation tools, seven were defined as extensions of the CloudSim toolkit [21].

Yousefpour et al. [22] presented a complete survey about FC, referencing 450 publications specifically concerned with FC development and applications. Within their extended survey, some FC simulation tools, such as iFogSim [23, 24], Emufog [25], Fogbed [26], and MyiFogSim [27] were discussed. As iFogSim was conceptually built using the CloudSim communications model, it inherited some of its properties, such as the ability to co-execute multiple tasks at the same time and the availability of plugable resource management policies.

Generally speaking, any cloud simulation tool can be extended to be a fog-enabled simulation tool. This is because of the fundamental property of the fog layer acting as a bridge between end-users and the cloud. In other words, adding a fog module to a cloud simulation tool, describing communications, roles, services, and parameters of fog nodes, is sufficient to claim that the tool is a fog-enhanced cloud simulation tool. Additionally, in a project that targets a Fog-BC integration applications, many researchers used a reliable, general-purpose fog simulator and implemented the BC as if it was an application case, such as in [28]. The results of such a simulation approach can be trusted valid for limited cases, such as providing a proof of concept of the proposal. However, critical issues, such as scalability and heterogeneity in huge networks, need to be simulated in a more specialized simulation environments. To mention one critical

Ref.	PL	PoW	PoS	PoA	SC	DM	PM	IDM	F
[35, 36]	Python	✓	χ	χ	✓	χ	✓	χ	χ
[37]	Python	✓	χ	χ	χ	χ	✓	χ	χ
[39]	Java	✓	χ	χ	✓	χ	χ	χ	χ
[40]	Python	✓	✓	χ	χ	✓	χ	χ	χ
[41]	Python	χ	χ	χ	χ	χ	✓	χ	χ
[42]	Java	✓	χ	χ	✓	χ	✓	χ	χ
FoBSim	Python	✓	✓	✓	✓	✓	✓	✓	✓

Table 1. Blockchain simulation tools and their properties

case, the BC protocols deployed in different CAs require more precise and accurate deployment of the BC entities and inter-operation in different layers of a Fog-enhanced IoT-Cloud paradigm. Consequently, as some simulation scenarios need an event-driven implementation, while others need a data-driven implementation, a scenario outputs may differ when simulated using different simulation environments. Such possibility of fluctuated simulation outputs should normally lead to unreliable simulation results.

2.2 BC simulation tools

As we have previously investigated how a Fog-Blockchain integration is envisioned, we started the implementation of FoBSim with a simple BC simulation tool described in [29]. Consequently, we discuss the state of the art regarding BC simulation tools available in the literature. In later sections, we describe how FoBSim serves as a reliable tool to mimic an FC-BC integration scenario.

Anilkumar et al. [30] have compared different available simulation platforms specifically mimicking the Ethereum BC, namely Remix Ethereum [31], Truffle Suite [32], Mist [33], and Geth [34]. The comparison included some guidelines and properties such as the initialization and the ease of deployment. The authors concluded that truffle suite is ideal for testing and development, Remix is ideal for compilation and error detection and correction, while Mist and Geth are relatively easy to deploy. Alharby et al. [35] and Faria et al. [36] proposed a somewhat limited simulation tool, namely BlockSim, implemented in Python, which specifically deploys the PoW algorithm to mimic the BitCoin and Ethereum systems. Similarly, Wang et al. [37] proposed a simulation model to evaluate what is named Quality of Blockchain (QoB). The proposed model targeted only the PoW-based systems aiming to evaluate the effect on changing different parameters of the simulated scenarios on the QoB. For example, average block size, number of transactions per block/day, the size of the memPool, etc. affecting the latency measurements. Further, the authors identified five main characteristics that must be available in any BC simulation tool, namely the ability to scale through time, broadcast and multi-cast messages through the network, be Event-Driven, so that miners can act on received messages while working on other BC-related tasks, process messages in parallel, and handle concurrency issues.

Gervais et al. [38] analyzed some of the probable attacks and vulnerabilities of PoW-based BCs through emulating the conditions in such systems. Sub-consequently, they categorized the parameters affecting the emulation into consensus-related, such as block distribution time, mining power, and the distribution of the miners, and network-related parameters, such as the block size distribution, the number of reachable network nodes, and the distribution of those nodes. However, they basically presented a quantitative framework to objectively compare PoW-based BCs rather than providing a general-purpose simulation tool.

Memon et al. [39] simulated the mining process in PoW-based BC using the Queuing Theory, aiming to provide statistics on those, and similar systems. Zhao et al. [40] simulated a BC system for specifically validating their proposed Proof-of-Generation (PoG) algorithm. Hence, the implementation objective was comparing the PoG with other CAs such as PoW and PoS. Another limited BC implementation was proposed by Piriou et al. in [41], where only the blocks appending and broadcasting aspects are considered. The tool was implemented using Python, and it aimed at performing Monte Carlo simulations to obtain probabilistic results on consistency and ability to discard double-spending attacks of BC protocols. In [42], the eVIBES simulation was presented, which is a configurable simulation framework for gaining empirical insights into the dynamic properties of PoW-based Ethereum BCs. However, the PoW computations were excluded in eVIBES, and the last updates on the code were committed in 2018.

To highlight the comparison between the mentioned BC simulation tools and our proposed FoBSim

tool, we gathered the differences in Table 1. PL, PoW, PoS, PoA, SC, DM, PM, IDM, and F are abbreviations for Programming Language, Proof-of-Work, Proof-of-Stake, Proof-of-Authority, Smart Contracts, Data Management, Payment Management, Identity Management, and Fog-enhanced, respectively. As shown in the table, none of the previously proposed BC simulation tools made the PoA algorithm available for simulation scenarios, provided a suitable simulation environment for identity management applications, or, most importantly, facilitated the integration of FC in a BC application.

Many other references can be found in the literature, in which a part of a BC system, or a specific mechanism is implemented. The simulated 'part' is only used to analyze a specific property in strict conditions, or to validate a proposed technique or mechanism under named and biased circumstances, such as in [43] and [44]. It is also worth mentioning here that some open-source BC projects are available and can be used to simulate BC scenarios. For example, the HyperLedger [45] projects administered by the Linux Foundation are highly sophisticated and well implemented BC systems. One can locally clone any project that suits the application needs and construct a local network. However, those projects are not targeting the simulation purposes as much as providing realized BC services for the industrial projects. Further, most of these projects, such as Indy, are hard to re-configure and, if re-configured, very sensitive to small changes in their code. Indy, for example, uses specifically a modified version of PBFT CA, namely Plenum, while Fabric uses RAFT.

3 FC ARCHITECTURAL ELEMENTS

The FC layer can be studied in three levels, namely the node level, the system level, and the service level [46]. The fog consists of several nodes connected to each other and to the cloud. The main purpose of the fog layer is to provide cloud services, when possible, closer to end-users. Further, the fog layer, conceptually, provides enhanced security and latency measures. Hence, an FC system uses its components in the fog layer to provide the services that end-users request from the cloud.

In a simple scenario, the fog receives a service request from end-users, perform the required tasks in the most efficient method available, and sends the results back to end-users. As the clouds mainly provide Infrastructure, Software, and Platform -as-a-Service models, those three models can be used for computational tasks, storage tasks, or communication tasks [47].

In a Fog-enhanced Cloud system, a general overview of the workflow is presented in Figure 1. As presented in the figure, the service is requested from end-users and the fog layer provides this service if possible, otherwise, the request is forwarded to the cloud where complex and time consuming actions are performed. However, information of the complexity of the system, and the decision making process in the fog layer, should not be within the concern of end-users. That is, end-users require their tasks to be performed within a privacy-aware context and the QoS measures implications that were agreed on.

In FoBSim, the fog layer can be configured according to the scenario that needs to be simulated. For example, the number of fog nodes, the communications within the fog layer and with other entities of the simulated system, and the services provided by the fog, can all be modified.

4 BC ARCHITECTURAL ELEMENTS

BC as a DLT consists of several elements that need to efficiently interact with each other, in order to achieve the goal of the system. A general view of BC systems suggests some fundamental components that need to be present in any BC system. A BC system implies end-users who request certain types of services from a BC network. The BC network consists of multiple nodes, who do not trust each other, that perform the requested services in a decentralized environment. Consequently, the service provided by a BC network can only be valid if the BC network deployed a trusted method, i.e. CAs, to validate the services provided by its untrusted entities.

In FoBSim, the BC network can provide two models of services; namely data storage, and computations. Meanwhile, the communications within the BC network and with the fog layer are configurable. Data storage service model implies that pieces of data are saved on the immutable distributed ledger. Such data may be of any type including data records, IDs, digital payment registration, or reputation measures of end-users or Fog components. It can also be noted that some applications require assets to be transferred between clients, such as cryptocurrency transfer applications or real estate ownership applications. Other applications do not require transferring assets rather than saving data on the chain only, such as voting applications and eHealth applications. However, the mentioned second type of applications

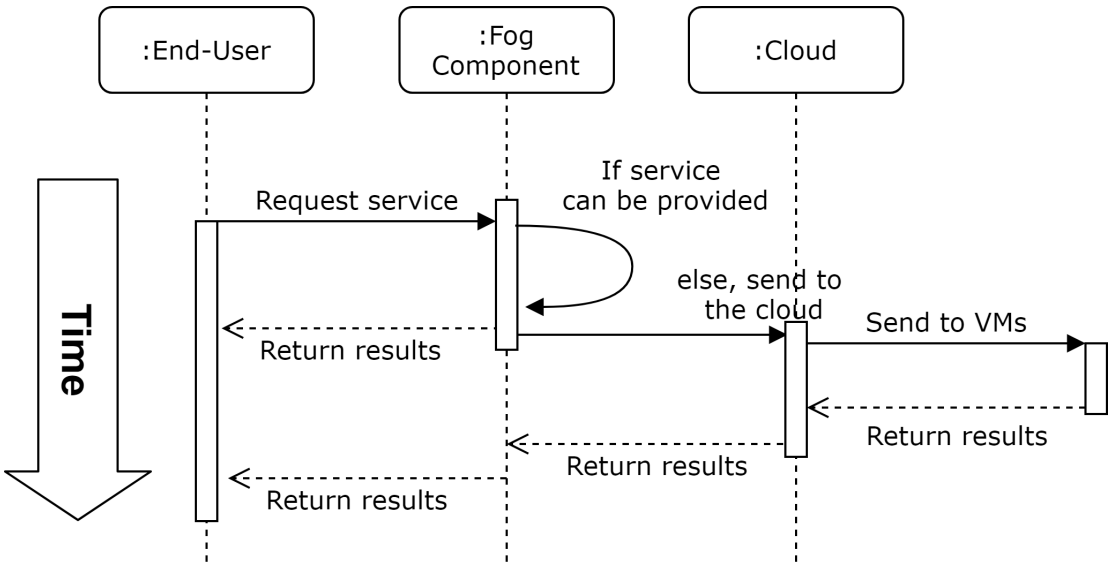

Figure 1. Workflow of an automated Fog-enhanced Cloud system

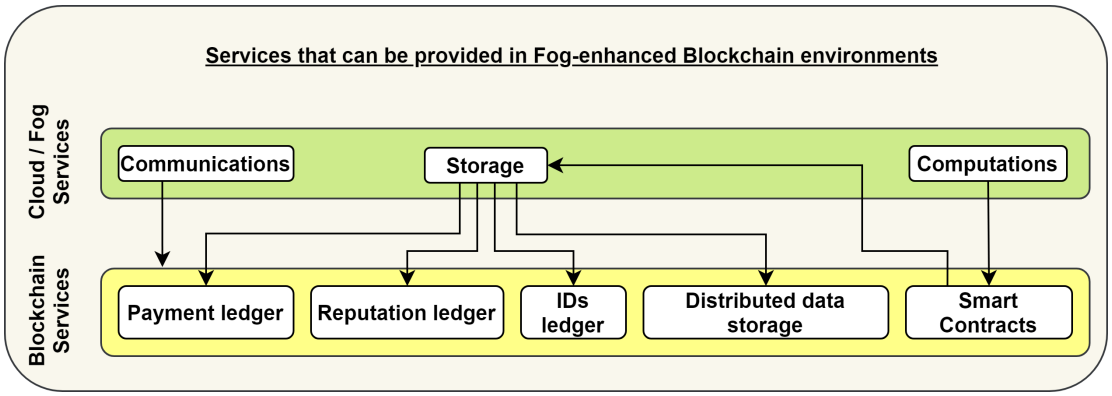

Figure 2. Service models provided by Cloud/Fog systems, and their relevant service models provided by BC systems

may also need, on some level, a digital payment method be embedded. In such cases, SCs on other payment platforms can be implemented and generated, such as Bitcoin or Ethereum.

Performing computations for end-users is the second service model that the BC in FoBSim can be configured to provide. That is, computational tasks can be sent by end-users/fog entities to the BC in the form of SC, which are small chunks of code, run by BC nodes upon fulfillment of algorithmically verifiable conditions [48]. After running the SCs, the results can be saved in a centralized or decentralized form according to the pre-run configuration. Figure 2 presents how the services, classically provided by a Cloud/Fog system, can be interpreted into the form of services that can be provided by a BC system. We can notice in the figure that SCs can be considered relevant to cloud computational services, while different types of data saved on the decentralized BC can be considered a relevant option to the centralized storage model provided by a cloud system.

4.1 Consensus Algorithms

Several approaches were proposed as a solution for the aforementioned needs, among which are the most famous Proof-of-Work (PoW) CA. PoW was deployed in 2009 in the first BC system, i.e. Bitcoin [49], and is currently used in other robust BC systems; such as Ethereum [50]. Although PoW methods have proven strong security and support to BC systems, it has some drawbacks, such as high energy consumption and high latency, that encouraged the R&D communities to search for other trusted methods.

The Proof-of-Stake (PoS) algorithm [51] was proposed couple a years later in order to solve the high energy consumption problem implied by PoW. PoS is currently being optimized to provide similar advantages as PoW. Ethereum, for example, is planning to substitute PoW with PoS in the very near future. However, some drawbacks of PoS need to be solved before its official deployment, such as The Monopoly Problem [52], The Bribe Attack [53, 54], and relatively low reliability [55].

In PoW-based BCs, a BC node proves the validity of its generated block of data by coupling a puzzle solution within the block. The puzzle solution is generally characterized by hardship to be obtained while it can easily be validated once found. Generally, the puzzle is a mathematical problem that requires high computational power to be obtained. In PoS-based BCs, the BC node that is allowed to generate the next block is chosen randomly by the system. To encourage the system to pick a specific BC node, staking more digital coins in deposit shall increase the probability of being chosen. This provides high trust measures as faulty generated blocks are not tolerated by the system, and the staked coins of the malicious/faulty BC node would be burned as a penalty.

Other approaches were proposed that provide trust in BCs. Examples include the Proof-of-Elapsed-Time (PoET) [56], and the Proof-of-Authority (PoA) [57]. PoET-based BCs generate randomly selected times for BC nodes. The one node whose randomly picked time elapses first, is the one who is granted the opportunity to generate the next block. PoA, on the other hand, implies that only blocks signed by authorized members are validated and confirmed by the BC network. Those authorized nodes must be known trusted participants that can be tracked and penalized in case of faulty behaviour. Both of these CAs share the property of being suitable for private and permissioned BCs, while PoW and PoS are known for being suitable for public and permissionless BCs.

FoBSim allows to choose the suitable CA according to the simulated scenario. While there are many versions of each CA mentioned, we currently provide the simplest version of each so that modifications can be performed with no complexities. To obtain more information about them, however, more detailed information can be found at [58, 59, 60].

4.2 Transactions

In a very simple scenario, an end-user sends a request to the BC network, which consists of BC nodes, to perform a defined transaction. As stated in the beginning of this section, transactions may be data to be stored (i.e. payment data, reputation data, identity data, etc.), or can be SCs whose results can be either saved in a centralized (in the case of Cloud) or distributed manner (in the cases of fog or BC). Once the transaction is performed, it should be agreed on by the majority of BC nodes if to be saved on the distributed ledger and, sub-consequently, be added to the chain saved in all BC nodes.

On the other hand, if the fog layer is controlling and automating the communications between the end-user layer and the BC network, as in [61], the transactions are sent from end-users to the fog. After that, some communications take place between the fog layer and the BC network in order to successfully perform the tasks requested by end-users. In such system model, we assume that the BC network lays in a different layer than the fog layer. The case where the BC network is placed in the fog layer is covered in Subsection 4.4. Nevertheless, a feedback with the appropriate result of each transaction should be easily achievable by end-users.

4.3 Distributed Ledger

In the case were data needs to be stored in a decentralized manner, no Trusted Third Party (TTP) needs to be included in the storing process. The entity considered as a TTP in regular Fog-enhanced Cloud systems is the cloud, where data is stored. However, computations can take place in the fog layer to enhance the QoS.

Within DLT-enabled systems, such as BC, groups of data are accumulated in blocks, and coupled with a proof of validity, as explained in Subsection 4.1. Once a new block of transactions is generated, and the proof is coupled with them, the new block is broadcast among all BC nodes. Nodes who receive the new blocks verify the proof and the data within each transaction, and if everything is confirmed valid, the new block is added to the local chain. With each BC node behaving this way, the new block is added to the chain in a distributed manner. That is, a copy of the same chain, with the same exact order of blocks, exists in each BC node. Further, a hash of the previous block is added to the new block, so that any alteration attack of this block in the future will be impractical, and hence almost impossible.

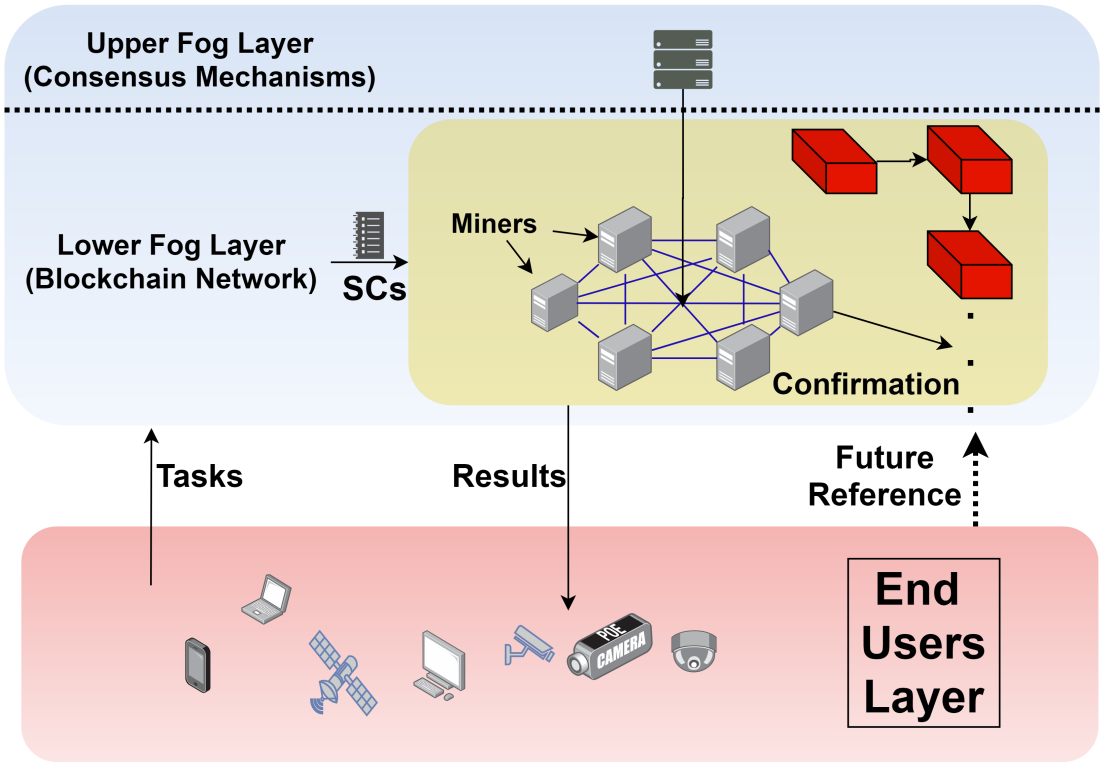

Figure 3. FC-BC integration system model, where the BC is deployed in the fog layer.

4.4 Functionality of the BC Deployment

As a BC-assisted FC system can provide computational and storage services, the BC placement within the the FC architecture may differ. That is, BC can be placed in the fog layer, the end-user layer, or the cloud layer. In FoBSim, however, we consider only the first two mentioned placement cases.

When the BC is deployed in the fog layer, storage and computational services are performed by the fog nodes them selves. In other words, fog nodes wear a second hat, which is a BC network hat. Thus, when storage to be provided by the fog while fog nodes are also BC nodes, data is stored in *all* fog nodes in the fog layer. A simple system model is demonstrated in Figure 3, where only one chain is constructed in the lower fog layer and one fog control point in the upper layer monitors the BC functionality. However, such a model is not practical and more complexities appear in a real-life scenario, including heterogeneous fog nodes, multiple BCs deployment, different CAs, and different service models. In such complex systems, FoBSim can be easily extended by adding the needed classes and modules and, hence, cover necessary proposed scenario entities. A note is worth underlining here is the importance of differentiating between the services provided by fog nodes who are BC nodes, and the services provided by fog nodes who are not BC nodes. The first type gets incentivized by end-users for providing both fog services and BC services, while the second type gets incentivized by end-users for providing only fog services. Such critical issues need to be taken care of, when simulating Fog-BC scenarios, to maximize the reliability of the obtained results.

In a system model where the BC is deployed in the end-user layer, we can distinguish two types of end-users; namely task requester and BC node. In a Fog-enhanced BC system, the fog controls the communications between the two types of end-users. Specifically, BC nodes perform the tasks that were sent to the BC network by the fog, which originally were requested by task requester end-users. Further, the fog can control the privacy preserving of data and incentivize BC nodes in the form of digital currency, as in [62]. To be specific, BC nodes can be further sub-categorized according to the scenario to be simulated. Adding other types of BC nodes is up to the developers and the system model. For example, the Bitcoin system is modeled in a simpler way, were BC is directly connected to task requester end-users, and it only provides a payment ledger service. Ethereum, on the other hand, provides computational and

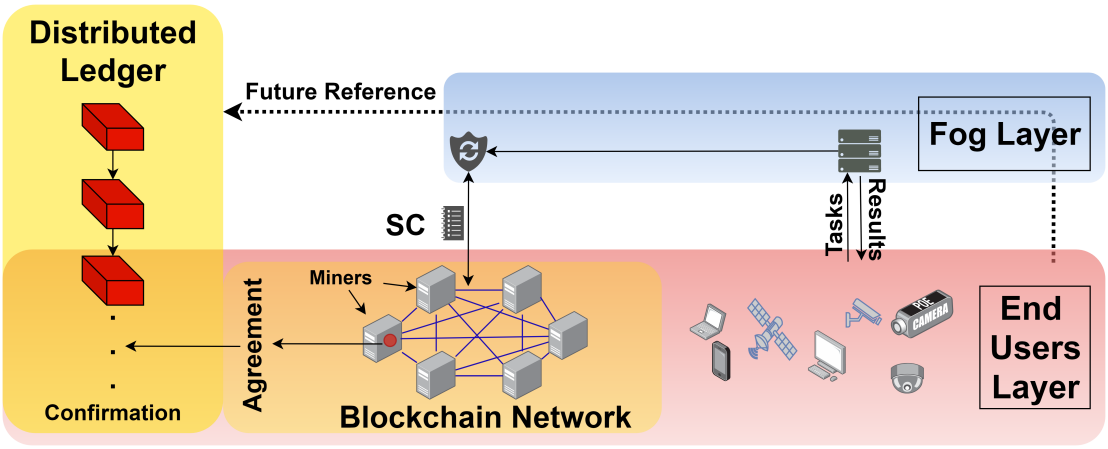

Figure 4. FC-BC integration system model, where the BC is deployed in the end-user layer

data management services. This makes Ethereum surpass Bitcoin because it can provide more services to end-users. However, FoBSim improves such system model by adding the fog layer. The system model provided by FoBSim when the BC is deployed in the end-user layer is demonstrated in Figure 4.

5 THE FOBSIM ENVIRONMENT

To cover all architectural elements described in Sections 3 and 4, we implemented FoBSim according to the conceptual workflow demonstrated in Figure 5. The current version of FoBSim covers all the architectural elements of a BC system and an FC system. This means that FoBSim successfully inlines with the general architecture of a reliable BC simulation presented in [15]. In fact, many more services and scenarios can be simulated using FoBSim, covering the fog layer inclusion besides the BC. As presented in Figure 5, different CAs can be used, different services of the BC network can be declared, and different placement scenarios of the BC network can be chosen. When the BC network is located in the fog layer, the number of BC nodes does not need to be input because, as described earlier, each fog node is also a BC node. Nevertheless, number of task requester end-users connected to each fog node needs to be input, while some fog nodes in a PoA-based scenario might be not authorized to mint new blocks. Once the network is built, running and testing the system model can take place.

The FoBSim environment is implemented using Python v3.8, with the inclusion of some common packages such as: *random*, *randrange*, *multiprocessing*, *time*, and *hashlib*. The current version of FoBSim can be cloned and directly run as all the variables, lists, dictionaries, and sets have been given initial values. However, these parameters can be modified before running the code in the *Sim_parameters.json* file. FoBSim tool is open-source and freely available at [63].

5.1 FoBSim Modules

Next, we discuss the modules of our proposed FoBSim environment and the interaction between its functions. To facilitate the understanding of FoBSim, we demonstrate the methods within each FoBSim module in Figure 6. Further, we conclude the classes and methods of FoBSim modules in Tables 2, 3, 4, 5, 6, and 7. Some notes to be taken care of need to be underlined as well:

1. There is a big opportunity for developers to implement new methods in the fog layer. For example, the fog nodes can be extensible to provide privacy-preserving mechanisms (such as described in [64]), computational services (such as described in [65]), or reputation and trust management services (such as described in [66]).

2. memPool.py: In this module, the mempool, where TXs are accumulated, is a python multiprocessing queue that allows different processes to synchronously add() and get() TXs.

3. There are other minor methods from other modules are also called by FoBSim entities that mints a new Block, or receives a new TX/Block, in order to synchronously and smoothly apply each different CA's policies, as declared in its simple version.

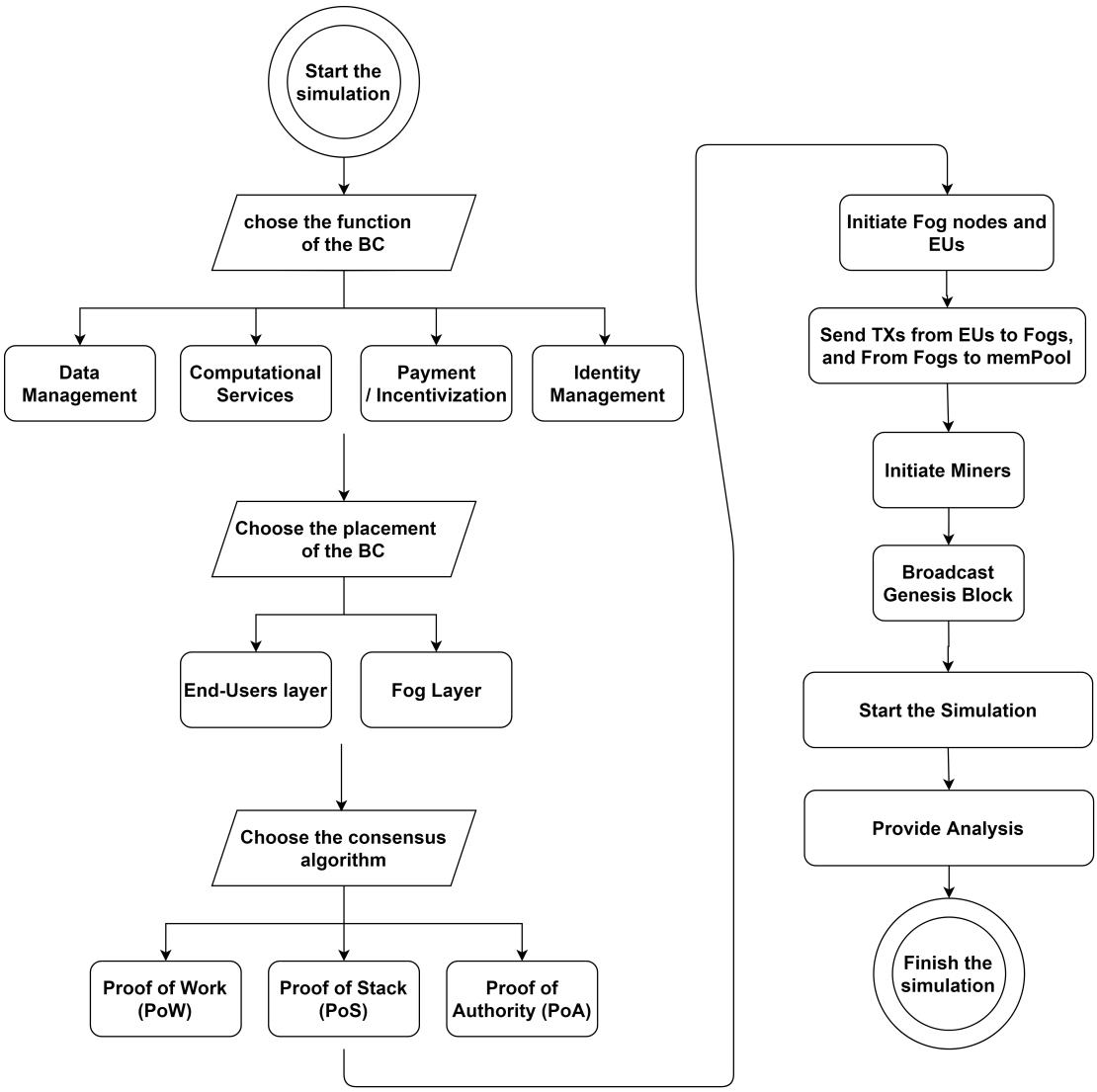

Figure 5. Workflow of a simulation run using the FoBSim environment

4. After each simulation run, some temporary files can be found in the temporary folder of FoBSim. These files are originally initiated by the main module, the BC module, or the miner module. The temporary files are used synchronously by different FoBSim entities, mimicking the real-world interaction between BC entities. The current version of FoBSim generates some or all of the following files depending on the simulated scenario:

    - Miners' local chains.
    - Miners' local records of users' wallets.
    - Log of blocks confirmed by the majority of miners.
    - Log of final amounts in miners' wallets (initial values - staked values + awards).
    - Log of coin amounts which were staked by miners.
    - The longest confirmed chain.
    - Forking log

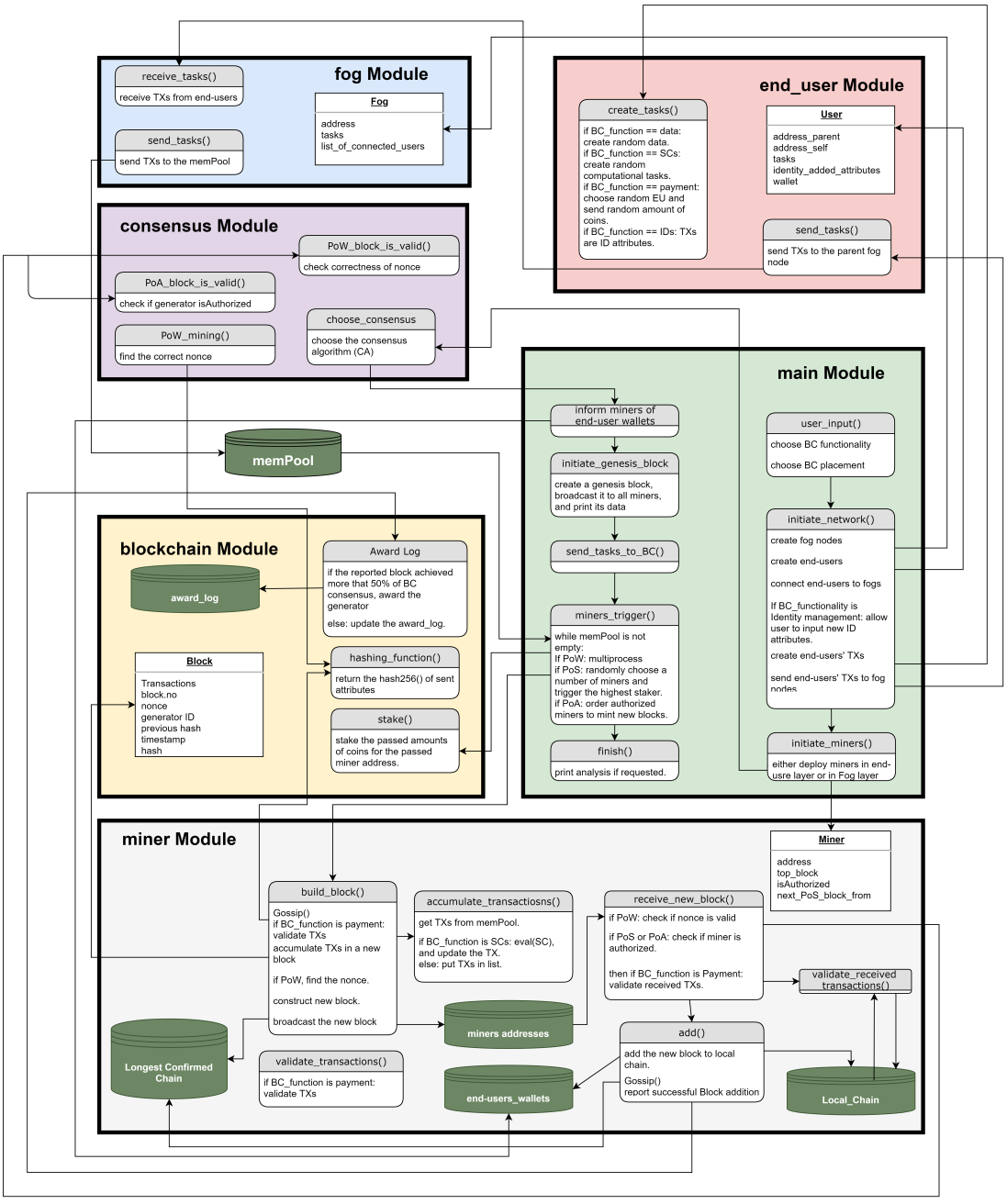

Figure 6. The interaction among modules and methods of the FoBSim environment

Function	Description
user_input()	The BC_functionality and BC_placement are input by the user. Then this function initiates temporary files. Currently, there are four functionalities available, namely Data management, Computational services, Payment, and Identity management, and two placement options, namely Fog layer and end-user layer.
initiate_network()	user inputs additional Id attributes (if applicable). Fogs/end-users are then constructed, end-users are triggered to create new TXs and send them to fogs. Fogs receive TXs and wait for trigger.
initiate_miners()	Miners are constructed and relevant temporary files to the BC construction are initiated.
connect_miners()	Miners are connected in a P2P fashion and the network is confirmed to be one giant component.
give_miners _authorization()	Allows the authorization of some miner nodes to mint new blocks in case the CA is PoA.
inform_miners _of_users_wallets()	Informs miners about the initial values of end-user wallets.
initiate_genesis _block()	A new block is built whose previous_hash value = 0, block_no = 0, and TXs are the addresses of miners (Fig. 8). Then, Fogs are triggered to send TXs in their buffers to mempool.
miners_trigger()	Triggers miners to get TXs from memPool and start minting new blocks. The approach of calling miners sequentially (FOR loop) or in parallel is discussed in Subsection 5.3.2

Table 2. Functions in the main.py module

Function	Description
Class: **User**	Initiated with the attributes: addressParent, addressSelf, tasks, identity_added_attributes, and wallet
create_tasks()	if the BC function was Data Management, a TX is a randomly generated number coupled with the end-user address. if the BC function was Computational Services, a TX is a randomly chosen Elementary arithmetic operation (i.e. +, -, *, /) coupled with two randomly generated numbers. The produced random computational tasks is coupled with the addresses of end-users. Once a miner solves a computational task, result is appended to the TX, and saved on chain. If BC function is Payment, a TX is a randomly generated amount of coins (up to the amount in the end-user's wallet), coupled with a randomly chosen end-user and the end-user's self address. Validation and confirmation is conducted by the receiver miner. If BC functionality is Identity Management, a TX is the address of the end-user, coupled with any added ID attributes by the user. Table 8 declares the four formats of TXs in FoBSim, while Figure 7 present screenshots of TXs generated by FoBSim entities.
add_attributes()	A function that allows the user to add additional ID attributes to end-user devices.
send_tasks()	each user simply sends its tasks to the fog node it is connected with. Note that in FoBSim multiple end-users can connect to one fog node, while each end-user is connected to only one fog node. However, this can be re-configured according to the simulation scenario.

Table 3. The Class and Functions in the end_user.py module

Function	Description
Class: **Fog**	initiated with the attributes: address, tasks, and list_of_connected_users.
receive_tasks()	receives the TXs from end-users and saves them in its buffer "self.tasks"
send_tasks_to_BC()	sends all TXs in its buffer to the memPool modul

Table 4. The Class and Functions in the Fog.py module

Function	Description
generate_new _block()	outputs a list of TXs, a block number, a nonce value, a generator-id, the hash of the previous Block, the timestamp of the generation, and the self hash.
hashing_function()	uses the Secure Hash Algorithm (SHA256) to generate the hash of the encoded nonce, TXs, generator-id, and previous hash.
report_a _successful_block _addition()	records the votes sent by miners to indicate a successful majority confirmation of a named block.
fork_analysis()	A method that, when called, counts the number of different chain versions in the BC network.
stake()	used when the PoS algorithm is chosen, where random amounts of coins are taken from each miner's wallet, and staked in the BC. This contributes later to the BC system choosing (randomly) the miner that will mint the next Block, biased by a tendency to choose miners with higher staked coins.
award_winning _miners()	reads the voting record of winning miners and adds the winning award to their wallets.

Table 5. Functions in the Blockchain.py module

Function	Description
Class: **Miner**	Initiated with Address, Top_block (for saving the last confirmed block), a Boolean isAuthorized attribute (for declaring whether this miner is authorized to mint new Blocks in a PoA scenario), a next_pos_block_from variable to memorize the address of the next block generator, a set of neighbors, transmission delay, and a boolean gossiping variable.
build_block()	constructs valid blocks according to the chosen BC functionality and CA.
receive_new_block()	receives new blocks from neighbours, and adds them to its local chain if it was new and valid. When the new block is successfully added, it is forwarded to neighbours, otherwise it is discarded.
Validate _transactions()	Accepts new Blocks coming from other miners, validates them according to the BC functionality and the used CA, and adds valid Blocks to the local chain.
add()	performs and reports a successful Block addition
gossip()	investigates the longest chain in the BC network and, accordingly, updates the local chain according to majority consensus

Table 6. The Class and Functions in the miner.py module

Function	Description
choose _consensus()	allows the user to choose one of the available CAs in FoBSim.
PoW_mining()	provides miners with the method to search for the puzzle solution in PoW based scenarios.
PoW_block _is_valid()	returns either True or False according to the correctness of puzzle solution. If one of the TXs were invalid, the whole Block is rejected.
PoA_block _is_valid()	checks the validity of Blocks generated when the PoA CA is chosen. Additionally to the checks performed in the **PoW_block_is_valid()**, this method checks if the miner who minted the block is authorized to do so. If False returned, all TXs within the block are sent back to memPool.

Table 7. Functions in the consensus.py module

BC functionality	TX Format
Data Management	[random number]
Computational Services	[end-user ID, random computational task, Result, Miner]
Payment	[Amount to be paid, Sender address (parent), Sender address (self), Receiver address (parent), Receiver address (self)]
Identity	[end-user_address(parent), end-user_address(self), Any user added ID attributes]

Table 8. Types and formats of TXs in FoBSim

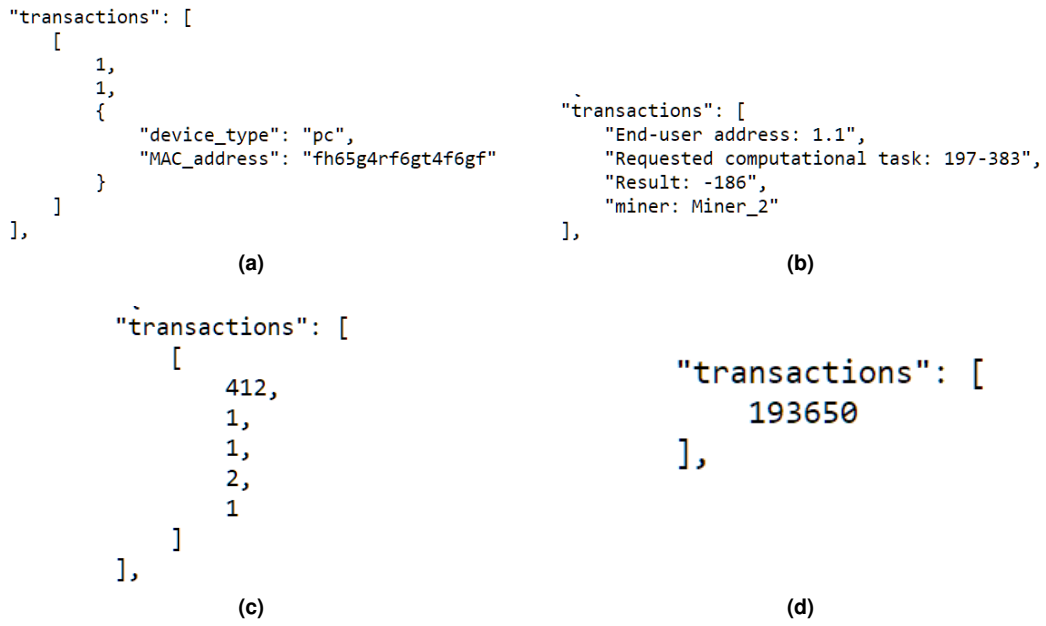

Figure 7. Samples of TXs produced by FoBSim entities (a): BC functionality is Identity Management, (b): BC functionality is Computational Service, (c): BC functionality is Payment, (d): BC functionality is Data Management

```
"0": {
    "transactions": [
        "genesis_block",
        "Miner_1",
        "Miner_2"
    ],
    "blockNo": 0,
    "nonce": 0,
    "generator_id": "The Network",
    "previous_hash": 0,
    "timestamp": "Tue Sep 29 16:28:02 2020",
    "hash": "5874f3ef3934727fa64a07c5b82df870a03c122ba5c258dbd1be6f441ad752da"
},
```


Figure 8. The Genesis block, with all its attributes, generated to miner nodes

5.2 Genesis Block Generation

The first block added to the chain in each simulation run is the most important block of the chain. Different scenarios imply different formats of this block, and different methods to broadcast it among, and be accepted by, miner nodes. In the current version of FoBSim, however, a genesis block is initiated with a list of TXs containing only the string 'genesis_block' and the labels of the miners available when this block was generated. The block number is 0, the nonce is 0, the generator_id is "The Network", previous hash is 0, and the hash is generated using the **hashing_function** in the blockchain.py module. The timestamp of genesis block indicates when the chain was launched, hence all blocks shall have bigger timestamp values than the genesis's timestamp. Figure 8 shows a standard FoBSim genesis block, generated in a BC network that consists of two miner nodes.

5.3 FoBSim Consensus Algorithms

Currently, there are three available CAs ready to be used in different simulation scenarios. Next, we describe each one individually as to facilitate any modifications by developers. However, we need to indicate that the three included CAs are in their simplest versions and may require some individual modification in case of the need of more complicated ones. Before delving into the CAs, however, we need to discuss the Gossip protocol in FoBSim, as it is deployed regardless of what CA is chosen.

5.3.1 Gossip Protocol

A Gossip Protocol [67] is usually deployed in peer-to-peer (P2P) systems for maintaining the consistency of distributed data saved in decentralized networks. Specifically in BC systems, miner nodes regularly, yet randomly, gossip to their neighbours about their current version of the chain, aiming to reach consensus finality as soon as possible. According to specific characteristics of the BC, the locally saved chains are updated so that all confirmed chains are equivalent at any given moment [68]. The equivalency that any BC system is seeking is defined by the contents similarity of the chains (i.e. TXs, hashes, etc.), and the order similarity of the confirmed blocks. That is, a chain $[b_1, b_2, b_3]$ is not equivalent to $[b_1, b_3, b_2]$ despite the fact that both have similar contents.

Gossiping protocols are usually fault tolerant as many failing nodes do not affect the protocol. Furthermore, they can adapt to the dynamics of the network, so some solutions have been proposed in the literature for nodes joining and leaving the network. However, gossiping is an iterative method that never quits as long as the network is up, and it may take time to converge. Additionally, high level of communication costs is expected for gossiping, while randomly chosen neighbors are informed about updates. Thus, one cannot provide precise analysis about the time needed for the network agreement on a piece of data.

Although the implementation of such protocol is relatively simple, it is differently implemented in different systems. Some famous examples of efficient gossiping protocols include the Push-Sum protocol [69], the Push-Flow algorithm [70], and different versions of the Push-Pull averaging protocol [71]. Furthermore, we found that its application in FoBSim was useful, when the PoW CA is used in a multiprocessing scenario, with a relatively low puzzle difficulty. Additionally, it can be easily noted that the number of simulated TXs/blocks and the initial TX per block configuration affects the speed

of the system to reach consensus finality. That is, for low number of TXs, blocks, and low ratios of TX per block, miners might not have the required time to converge locally saved chains. Accordingly, final versions of local chains in some FoBSim simulations, under such circumstances, may not coincide, which is normal and expected as described in [72]. Nevertheless, we deployed a simple Push-Pull Gossip version in FoBSim that works perfectly fine, so that modifications can be easily conducted if needed. In the current version of FoBSim, a Time To Live (TTL) parameter was not added to the Pull requests when gossiping. This, as expected, floods the network with Pull and Push requests each time a node wants to gossip. Nevertheless, we faced no problem whatsoever when the network consisted up to 1500 miners. If more miners need to be deployed in the simulation scenario,where gossiping is activated, we recommend either configuring the gossiping requests to have a TTL (i.e. a number of hops the request perform before it is terminated), and/or decreasing the number of neighbors the gossiping node is sending the gossip request to. That is, instead of gossiping with all neighbors, a miner can randomly choose a neighbor to gossip with. Consequently, each neighbor will gossip with a randomly chosen neighbor of his, etc. More details on such implementation approach can be found in [73], while detailed analysis regarding the success rate of gossiping, with a given TTL in a given P2P network, can be found in [74].

Algorithm 1 describes how the Pull-request in the default Gossip protocol of the current version of FoBSim works. If the gossiping property was set to *true*, Each miner runs this algorithm each time the Gossip() function is called for that miner (as a default, the Gossip function is called each time a miner is triggered to build a new block and when a new block is received). As demonstrated in the algorithm, a default FoBSim miner requests information about the longest chain, and adopts it if its contents were agreed on by the majority of the network, which is a condition tested using Algorithm 2. Additionally, if a miner receives a new valid block, and the resulting local chain was longer than the global chain, the miner updates the global chain instantly, which represent the Push request of the Gossip protocol in FoBSim.

In big BC networks, the mentioned issues need to be carefully designed, so that the consistency of the distributed ledger by the end of the simulation run is guaranteed, while the efficiency of the algorithm is optimized.

Algorithm 1: The default Gossip protocol in FoBSim

```
Result: Confirmed Local_chain in μ_g
initialization: Self(miner μ_g);
confirmed_chain = self.local_chain;
temporary_global_chain = longest_chain;
Condition_1 = len(temporary_global_chain) > len(confirmed_chain);
Condition_2 =blocks in temporary_global_chain are confirmed by network majority;
if Condition_1 AND Condition_2 then
    confirmed_chain = temporary_global_chain;
    self.local_chain = confirmed_chain;
    self.top_block = confirmed_chain[str(len(confirmed_chain)-1)];
    if BC_function is Payment then
        self.log_users_wallets = confirmed_chain_from.log_users_wallets
    end
end
```

5.3.2 The Proof of Work

In a simplified scenario of a PoW-based BC, miners collect TXs from the mempool (which is a shared queue in FoBSim) and accumulate them in blocks that they mint. Specifically, all available miners *compete* to produce the next block that will be added to the chain. The fastest miner producing the next block is the miner whose block is accepted by all other miners of the BC. Synchronously, all blocks that are being minted by other miners are withdrawn, and all TXs within are sent back to the mempool. To mimic this scenario in FoBSim, we needed to deploy the multiprocessing package of Python and trigger all miners to work together on the next block.

Each miner then works within an isolated core of the device on which the simulation is conducted. Using this approach is doable and explainable in simple scenarios, where each process needs to access one or few shared objects. However, we found it challenging to mimic complex scenarios, where huge

Algorithm 2: The default chain confirmation function in FoBSim

Result: bool **chain_is_confirmed**
Passed parameters: Chain ***C***, network size;
initialization: chain_is_confirmed = True;
block_confirmation_log = blockchain.confirmation_log;
Condition_1 = **not** (C[block]['hash'] in block_confirmation_log);
Condition_2 = block_confirmation_log[chain[block]['hash']]['votes'] <= (network size / 2);
for *block in* ***C*** **do**
    if *Condition_1 OR Condition_2* **then**
        chain_is_confirmed = False;
        break
    end
end
return **chain_is_confirmed**

number of processes require accessing the same shared lists. For example, when BC functionality is payment, the BC deployed in the fog layer, and the CA is PoS the wallets of end-users, fog nodes, and mining nodes need to be all global for read and update by all processes. We also experimented the Python package: multiprocessing.shared_memory, which partially solved the problem as multi processes can read and update values in a Shareable List object. However, as declared in the official Python documentation [75], the Shareable List object lacks the dynamicity required in terms of length and slicing. According to the mentioned insights, we implemented two approaches for PoW mining in FoBSim, the first starts all miners in parallel (using the multiprocessing package), while the second consequentially calls for miners to mint new blocks (using a FOR loop). Both approaches are available in the **miners_trigger()** function in the main.py module, and developers are free to use either. We do encourage the developers, however, to be cautious and carefully test their results when using the parallel processing approach, as each different scenario may require different access management scheme to different FoBSim entities. Hence, a complex scenario simulation may require some modifications to some variables and lists so that they become shareable by all processes in different modules. Detailed instructions for implementing different memory-sharing scenarios can be found in the Python official documentation [76].

When a Miner receives a new block, it checks whether the hash of the block (in which the nonce or the puzzle solution is included) is in line with the acceptance condition enforced by the blockchain.py module. Further, the receiver miner checks whether sender end-users have sufficient amount of digital coins to perform the TX (in the case of payment functionality). In the contrary to the case of PoS and PoA, all miners work at the same time for achieving the next block. Hence, any miner is authorized to produce a block and there is no miner verification required. Algorithm 3 presents how PoW is implemented in FoBSim.

5.3.3 The Proof of Stake

In a simplified version of PoS, miners stake different amounts of digital coins (which they can not claim) in the BC network. The network then randomly chooses a miner to mint the next block, with higher probability to be chosen for miners who stake more coins. Once a miner is chosen, it is the only one authorized to mint and broadcast the next block. In case of faulty TXs/blocks, the minter loses its staked coins as a penalty, while in case of correct blocks, the minter is awarded some digital coins.

To mimic this in FoBSim, each miner is initiated with specific amount of coins in its wallet. After that, randomly generated number of coins (up to the amount of coins in its wallet) is staked by each miner. In this way, every miner has different probability to be chosen by the network. Next, the network randomly chooses, say 10% of the available, miners and picks the one with the highest stake. This chosen miner's address is immediately broadcast to all miners so that any block received from any other miner is rejected. Once the new block is received, it is validated and added to the local chain. Algorithm 4 presents how PoS is implemented in FoBSim.

Here, a very wide space is available for implementing reputation management schemes in FoBSim. Different scenarios and different applications require different parameters affecting entities' reputation. Further, adding other types of miners, end-users, or even Fogs implies that different DBs can be suggested.

Algorithm 3: The default PoW mining algorithm in FoBSim miner

```
Result: New block β confirmation
initialization Self(miner μ_g);
Collect TXs from memPool;
Gossip();
if BC_function is Payment then
    validate collected TXs
else
    if BC_function is Computational Services then
        eval(TXs);
        add the evaluation results to TXs
    end
    Accumulate TXs in a new BC block β;
    Find the puzzle solution of β (nonce);
    Broadcast β to neighbors;
end
if New block β is received then
    Gossip();
    if β nonce is correct then
        if BC_function is Payment then
            validate and confirm TXs in β
        end
        add block β to the local chain;
        Broadcast β to neighbors;
        report a successful block addition [β, μ_g]
    end
end
```

It is also worth mentioning here that we found it unnecessary to use the multiprocessing package because only one miner is working on the next block. Hence, no competition is implied in the PoS scenario.

5.3.4 The Proof of Authority

In a simplified version of the PoA algorithm. only authorized network entities (by the network administrators) are illegible to mint new blocks. Regardless of the BC functionality, there is also no need to deploy the multiprocessing package for PoA-based scenarios as there is no competition as well.

To mimic the PoA in FoBSim, we allow the user to declare which entities are authorized to mint new blocks. The declaration requested from the user appears in the case of BC deployment in the fog or end-user layer. That is, each fog node is administering a group of end-users, and providing communications (and probably computations) services to them. However, it is not necessary for each fog node in the fog layer to be a BC node as well, but it should be there as only a fog node. Authorized fog nodes then are wearing both hats, fog nodes and BC miners. When the BC is deployed in the end-user layer, authorized miners are responsible for minting new blocks and maintaining the distributed ledger. Meanwhile, unauthorized miners are only responsible for validating new blocks, received from their neighbors, and maintaining the distributed ledger.

This approach allows for comfortably emulating a scenario where the BC in the fog layer and part of the fogs are included in the BC functionality. Notice that a fog node that is also a BC node performs all the required tasks in logical isolation. This means that a fog node that is administering a group of end-users has a buffer to save the end-users TXs, but it does not use these TXs to mint a new block. Rather, it sends these TXs to the mempool as required, and then, only if it was authorized, it collects TXs from the mempool. Notice also, that the mempool is a simple queue in FoBSim, yet it can be implemented for some scenarios to be a Priority Queue. Our implementation of isolating the services provided by a fog node that is also a BC miner facilitates the simulation of scenarios where TXs need to be processed according to their priority. For example, miner nodes in Ethereum usually choose the SCs with the highest Gas/award

Algorithm 4: The default PoS mining algorithm in FoBSim

```
Result: Confirmed new block β
initialization miners μ_[0,1,..n], miners.wallets, stake random no. of coins from each miner.;
The Network:;
while mempool.qsize() > 0 do
    Randomly choose a predefined no. of miners;
    Choose the miner with the highest Stake_value;
    Inform all miners of the ID of the next block generator μ_g;
end
The Miner:;
if a new ID μ_g is received from the Network then
    if MyAddress == μ_g then
        Collect TXs from memPool;
        if BC_function is Payment then
            validate collected TXs
        else
            if BC_function is Computational Services then
                eval(TXs);
                add the evaluation results to TXs
            end
        end
        Accumulate TXs in a new BC block β;
        Broadcast β;
    else
        Wait for a new block from μ_g;
        if β is received then
            if μ_g == β.generator then
                if BC_function is Payment then
                    validate and confirm TXs in β
                end
                add block β to the local chain;
                Broadcast β to neighbors;
                report a successful block addition [β, μ_g]
            end
        end
    end
end
```

provided by end-users. This is a type of prioritizing that can be simulated in FoBSim. Similarly, in Bitcoin, a priority value is computed for each TX according to Equation (1), and TXs with higher fees and higher priority values are processed faster [77]. The default PoA algorithm implemented in FoBSim is clarified in Algorithm 5.

$$Priority = \frac{\sum inputAge * inputValue}{TXsize} \tag{1}$$

5.4 Transaction/Block Validation in FoBSim

Here, we need to underline some differences between the terms Verification, Validation and Confirmation, and we need to see how FoBSim differentiates between those terms in different scenarios. As we have touched on these differences in [61], we need to accurately define each of these terms in order to correctly describe how FoBSim works.

Validation is the process when a miner (either a minter or receiver) checks the correctness of a claim. That is, in the case of a minter miner, the puzzle solution (or nonce) provided with the minted block needs

Algorithm 5: The default PoA mining algorithm in FoBSim

```
Result: Confirmed new block β
initialization Fog nodes Ψ_[0,1,..n];
if BC_placement is Fog Layer then
    User Input("address of authorized fog nodes")
else
    Input("address of authorized miners")
end
Save authorized miners μ_[0,1,..n] in Miners_List;
The Miner:;
while mempool.qsize() > 0 do
    if self.address ∈ μ_list then
        Collect TXs from memPool;
        if BC_function is Payment then
            validate collected TXs
        else
            if BC_function is Computational Services then
                eval(TXs);
                add the evaluation results to TXs
            end
        end
        Accumulate TXs in a new BC block β;
        Broadcast β to neighbors;
    end
end
if β is received then
    if μ_g ∈ μ_list then
        if BC_function is Payment then
            validate and confirm TXs in β
        end
        add block β to the local chain;
        Broadcast β to neighbors;
        report a successful block addition [β, μ_g]
    end
end
```

to be correct before the block is broadcast. If the nonce was valid, the block is broadcast, otherwise, a new solution is searched for. While in the case of a receiver miner, the nonce is checked once. If in this later case the solution was valid, the block is accepted, otherwise, the block is rejected.

In the case of payment functionality, the validity of TXs fetched from the mempool is tested. This means that the amount of coins in the wallet of the sender of each TX, in the payment functionality, is compared to the amount to be transferred. If the wallet contains less than the transferred amount, the TX is withdrawn from the block. Later when the new block is received by a miner, the same hash validation and TXs validation take place, except if one of the TXs were invalid, the whole block is rejected. In the case of a block rejection, the minter miner is usually reported in a reputation-aware context. If all the contents of a newly received block are valid (i.e. the hash, the TXs, the wallets, the block number, and the nonce) the block is added to the locally saved chain. Here, we can say that TXs are confirmed, because the block is added to the chain (i.e. the block is confirmed).

The verification, on the other hand, is the process of verifying the identity of an entity. For example, in the case of PoA, only authorized miners are allowed to mint new blocks. Similarly, in the case of PoS, the received block should be generated by a miner that all other miners expect to receive the new block from. Additionally, public information about end-users' wallets need to be accessible by miners to validate their TXs. Thus, a received a block, with some TXs generated by end-users who do not have wallets or the wallets contents are not readable by miners, can not be validated and confirmed not necessarily because

the end-users have no sufficient coins to transfer, but because the end-users can not be verified.

All of these critical principles are, by default, taken care of in FoBSim. All miners are informed about the end-users public identities and wallets contents. After that, transferred coins are updated locally in each miner. Consequently, a new TX from the same end-user will be compared to the updated amount of coins in its wallet. Invalid TXs are withdrawn from the block being minted, while invalid TXs cause the rejection of the whole received block. Once a block contents are validated, and the TXs/block generators are verified, the TXs are confirmed, the locally saved wallets amounts are updated, the block is locally confirmed and added to the chain. The most interesting thing, is that the very small probability of a double spend attack [78], which can appear in PoW-based scenarios as it is globally known about Bitcoin and Ethereum, can be easily simulated in FoBSim. All processes are actually happening during each simulation run, rather than substituting them with a small delay as in most BC simulation tools we checked. Hence, validation, verification, and confirmation processes can be modified according to the scenario to be simulated. Nevertheless, Bitcoin lowered the double spend attack probability by raising the difficulty of the puzzle through time. A property that can be as well modified in FoBSim. To facilitate the simulation of such critical scenarios, we deployed two broadcasting approaches for newly minted blocks. The first allows the broadcast process using a simple FOR loop, where miners sequentially validate and confirm new blocks. The second allows the broadcast process using the multiprocessing package, which allows all miners to receive and process new blocks at the same time. Relatively, developers need to be cautious when using the second approach, because of some critical challenges similar to those mentioned in Subsection 5.3.2.

5.5 Awarding winning miners

Generally speaking, BC miners get rewarded by two system entities for providing the BC service (i.e. BC functionality). The first is the end-user who generated the TX, who pays a small fee once the TX is confirmed (e.g. GAS in Ethereum). The second is the BC network itself (i.e. all miner nodes), who updates the winning miner's wallet once a new block (minted by the winning miner) is confirmed. We can notice here how important it was to clarify the difference between validation, verification, and confirmation. That is, a miner is verified by its public label and public wallet key/address (ID). Then, a miner being authorized to mint a new block is validated (claim). Finally, a miner is awarded for minting a conformable block (miner's wallet is updated).

In FoBSim, we implemented the second, where miners get rewarded for their services by the network. We assume this part is hard because it, also, needs to be agreed on by the majority of BC miners (i.e. at least 51%), and it requires the condition that they confirm the block. The default implementation of FoBSim does that. For the first incentivization mechanism, we thought that it is not applicable in many different scenarios, hence we left it for the developers to add it if needed. For example, to allow end-users to provide fees for getting tasks in the BC, one field can be added to generated TXs, containing the amount of fees the end-user is willing to pay for the service. Once a miner picks a TX (mostly, TXs with higher fees are faster to be picked and processed by miners) and the block containing the TX is confirmed, all miners add the TX fees to the winning miner's wallet. Figure 9-a presents a screenshot of FoBSim output, concluding that a new block was received from Miner_2 by Miner_3, and that the BC module just obtained the needed confirmations to consider the new block confirmed by the whole BC network. Thus, the minter is awarded. Later, the receiver miner presents its updated local chain according to the successful network confirmation. Figure 9-b presents a screenshot of the miner_wallets_log after a simulation run, where the PoA CA was used and all miners, except for Miner_5, were authorized to mint new blocks (initial wallet value was 1000).

5.6 Strategies in FoBSim

As had been discussed so far, there are some default strategies used by FoBSim entities throughout each simulation run. To mention some, TXs are picked by miners with no preference, e.g. the highest GAS or priority. Also, a default chain is a single linear chain and new blocks are added to the top of this chain. Some applications, however, have multiple chains or multi-dimentional chains, e.g. Directed Acyclic Graph (DAG) based chain. Additionally, if two blocks appear in the network, the block that was accepted by the majority of miners is confirmed rather than, in some BC systems, the older one is confirmed even if it was confirmed by the minority of minors. Further, a valid block is immediately announced, once found, into the FoBSim network, while in some applications, there might be a conditional delay. For instance, if

Figure 9. A sample of FoBSim output. (a) confirming a new block receipt, a new award for mining the new block (as the required percentage of confirmations was reached), and the updated state of local chain of the receiver miner (b) Final miner wallets values in a PoA scenario

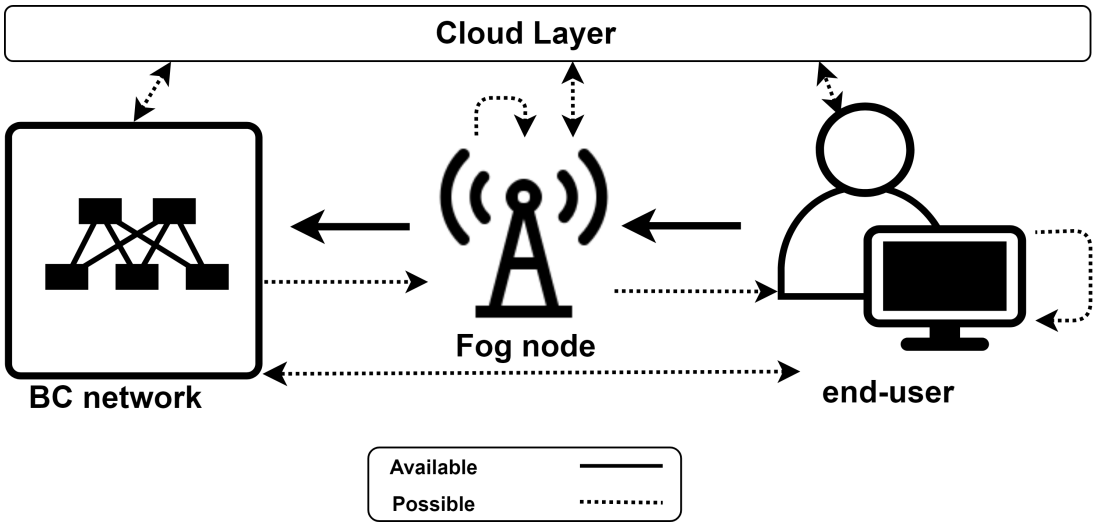

Figure 10. Possible data flow schemes in an integrated Fog-BC system

a selfish mining attack scenario to be simulated, miners would prefer to keep their newly found blocks secret, hoping they will find the next block as well [79].

The current version of FoBSim supposes that the data flows from end-users to Fogs, and from Fogs to the BC network. However, there are other possible data flow schemes that can be simulated, as depicted in Figure 10. For example, the BC in the current version provides DLT services to end-users, which are communicating with the BC through the fog layer, while services might be provided by the fog layer to the BC network or from the BC network to the fogs in some applications. Further, an application where end-users may need to request data directly from the BC might be possible, which implies different data flow scheme as well. FoBSim allows the modifications necessary for the simulated application to be easily done, and presents an extra Cloud module that can add more possibilities to the application.

Network connectivity characteristics are a major and critical concern in any BC system. To facilitate network architects job, FoBSim allows to define the number of nodes in each layer, the number of neighbors of each BC node, and the general topology of the network. Additionally, all BC nodes are connected into one giant component by default, whether they were deployed in the fog layer or end-user layer. Accordingly, the effect of manipulating the topology of simulated networks can be easily captured.

5.7 FoBSim Constraints

Some properties have not been implemented in the current version of FoBSim, such as Merkle Trees, Digital Signatures and Mining Pools. Additionally, FoBSim source code can be run on a PC with Microsoft Windows or Linux OS, but it may need some modifications if to be run on a PC with a MAC OS (some functions require access to OS operations such as deleting or modifying on files located at the

secondary memory). Finally, The default limit of recursion in python may restrict the number of miners to 1500, which may raise some error regarding the maximum allowed memory use by the interpreter. To solve this, one can modify the maximum limit using the sys.setrecursionlimit in the main function.

5.7.1 Merkle Trees

A Merkle Tree (MT), or Hash Tree, is a data structure, which is mostly a binary tree, whose leaves are chunks of data. Sub-consequently, each leaf is double hashed with its siblings to produce their new parent, which represents its two children. Hashes are recursively hashed together, in a binary manner, until obtaining one root that represents the whole tree. MTs are used in BCs such as BitCoin to decrease the probability of security attacks, along with other security measures, to reach the level where it is (a) easy for light weight nodes to validate new TXs and (b) computationally impractical to attack/alter a BC. That is, each TX in any given block is hashed with its next, and so on, so that one root hash of all TXs is saved in the block header. Using this root hash, and other components of the block, the hash of the block is generated. This means that not only a confirmed block is impossible to alter, but also a confirmed TX within a confirmed block.

However, not all BC systems deploy an MT approach due to some probable conflicts with system requirements or objectives. Thus, we decided to leave this to be implemented by developers according to the systems that need to be simulated, and we decided that the default configuration of BC nodes in the current version of FoBSim is to make all miners full node miners. That is, every miner locally stores a complete copy of the chain so that any TX can be validated according to TXs recorded locally. Additionally, there are different deployment models of MT approaches in different BC systems. That is, some BCs may deploy MTs for hashing other chunks of data/tokens instead of TXs.

To implement an MT approach in FoBSim, one can add a function that performs a loop through all TXs in a newly minted block, up to the last TX. After that, the root of the MT is added to the block before it is broadcast to the BC and the hash of the block is computed accordingly. Miners who receive a new block shall, accordingly, validate the added root. Hence, a validation step, to test the correctness of the MT root compared with TXs within the new block, needs to be added to the validation function in the miner module of FoBSim. To make use of such added property, one can define a light weight miner type which saves only the header of a newly confirmed block instead of the whole block. Accordingly, such type of miners validate new TXs according to this light chain of headers, hence consume less time, energy, and storage to maintain the work of the BC system.

5.7.2 Digital Signatures

As our main aim is to generally simulate TX generation, validation, and confirmation, in different BC-based, inter-operation, and consensus scenarios, we did not target security issues. This is because such issues are determined individually for each case to be simulated, leading to different mining economics. The discussion of security techniques and approaches in BC-based Fog and IoT systems had been discussed in many previous works, such as [80]. Specifically, digitally signed coins/tokens are primarily used in real-world applications of cryptocurrencies in order to prevent security attacks, such as the double spending attack. Different BC-based cryptocurrency systems used different mechanisms and protocols regarding signing and minting new coins, hence, different simulated scenario would require the implementation of the reference coins and digital signing techniques to be simulated. Examples might include a research work that aims at comparing different signing protocols in different CAs. Furthermore, FoBSim does not target a specific cryptocurrency system, such as BitCoin, yet it provides the generalized environment used in such systems, where problems and solutions can be implemented and emulated by researchers.

What the default version of FoBSim provides, however, is a simplified protocol of coin transfer between users. That is, each miner holds a locally saved record of user wallets, which is used in TXs validation in case of Payment BC functionality. We found that this approach can output similar results to those output by systems with signed coins, except that this approach allows a double spending attack in case of malicious end-users. If a scenario to be simulated, where there are some faulty/malicious entities among system users (which is not implemented in the default version of FoBSim), then digitally signed coins need to be implemented as well. Additionally, miner nodes in FoBSim are assumed to be trusted to send reports of confirmed blocks. Thus, reports sent by miner nodes to the network aiming to participate in voting regarding winning miners are assumed always legitimate. To sum up, FoBSim miners can track who, paid whom, how much, and they are trusted to participate in voting without a crypto-graphic proof.

While, in other implementation approaches, FoBSim miners may track who has transferred, what units, of which stocks (i.e. digitally signed coins/tokens), to whom, and their votes regarding winning miners must be verified by network entities (i.e. by also adding the new block to their local chains, and following this addition with other new blocks, each newly added block can be considered, in a sense, a confirmation). Similarly, end-users who generate new TXs do not need to sign their generated TXs as they are assumed trusted (i.e. the default implementation of FoBSim does not include malicious end-users).

5.7.3 Mining Pools

Pool mining is the collaboration between miners to form mining pools and distribute the earned rewards in accordance with pool policies to earn a steady income per miner [81]. Examples of such mining pools include BTC.com, F2Pool, and Slush Pool. Mining pools provide the advantages of making mining profits more predictable to miners and allowing small miners to participate. However, the existence of pool mining increases the probability of system centralization and discourages full nodes. The necessity of adding a mining pool extension to FoBSim is dependant on the scenario to be simulated. As the general idea of mining pools is to allow miners to perform mining under the umbrella of named group, if one of the group miners finds a block, the award is divided among all group members according to the computational power each member provides. A mining pool is managed by a pool manager, whose protocol is defined according to the business model of the pool.

In the current version of FoBSim, all miners are full nodes miners. That is, each miner tries to solve the puzzle using its own resources, it validates newly generated TXs and accumulate them into new blocks and when a block is received, it is validated and confirmed locally (all miners save the whole BC for validation, verification, and confirmation). Consequently, any profits and awards, obtained because of the full miner work, are directly added to the miner's wallet. While in the pool mining concept, a miner is awarded as much computational power it provides even if it was the one that found the next block.

6 CASE STUDIES

Following the validation and verification methods of simulation models presented in [82], we have so far discussed the technologies and the paradigms lying within our proposed FoBSim environment. Further, we highlighted our proposal novelty compared to other related works, discussed the event validity in FoBSim, and presented the algorithms and modules lying within to facilitate a structured walk-through validation.

Next, we follow an operational validity approach by presenting two case studies that we simulated using FoBSim. The setup and behaviour of FoBSim is discussed, and the results of the simulation runs are presented afterwards.

Case 1: Comparing time consumption of PoW, PoS, and PoA

When we compare PoW, PoS and PoA in terms of average time consumed for block confirmation, PoW is expected to present the highest time consumption. This is because of the mathematical puzzle that each minter needs to solve in order to prove its illegibility to mint the next block. In PoS, on the other hand, the network algorithm randomly chooses the next minter, while it slightly prefers a miner with higher amount of staked coins. Once a minter is chosen, all miners are informed about the generator of the next block and, thus, the minter needs to perform no tasks other than accumulating TXs in a new standard block. Other miners then accept the new block if it was generated by the minter they were informed about, hence the verification process takes nearly no time (assuming that the transmission delay between miners is set to 0). In simple versions of those two algorithms, all miners have the same source code, thus all miners may be minters, verifiers, and chain maintainers.

The PoA algorithm is the tricky one though. This is because all authorized miners mint new blocks, verify newly minted blocks, and maintain the chain locally. Meanwhile, other BC nodes verify new blocks and maintain the chain, but do not mint new blocks [83]. Consequently, every BC node has a list of authorized entities, including the methods to verify their newly minted blocks. This implies that the more authorized entities, the more complex the verification can be on the receiver side. Accordingly, it is advised that small number of entities be given authorization for decreasing the complexity of verification [84]. Meanwhile, the more maintainers in a PoA-based BC, the higher the overall security level of the system.

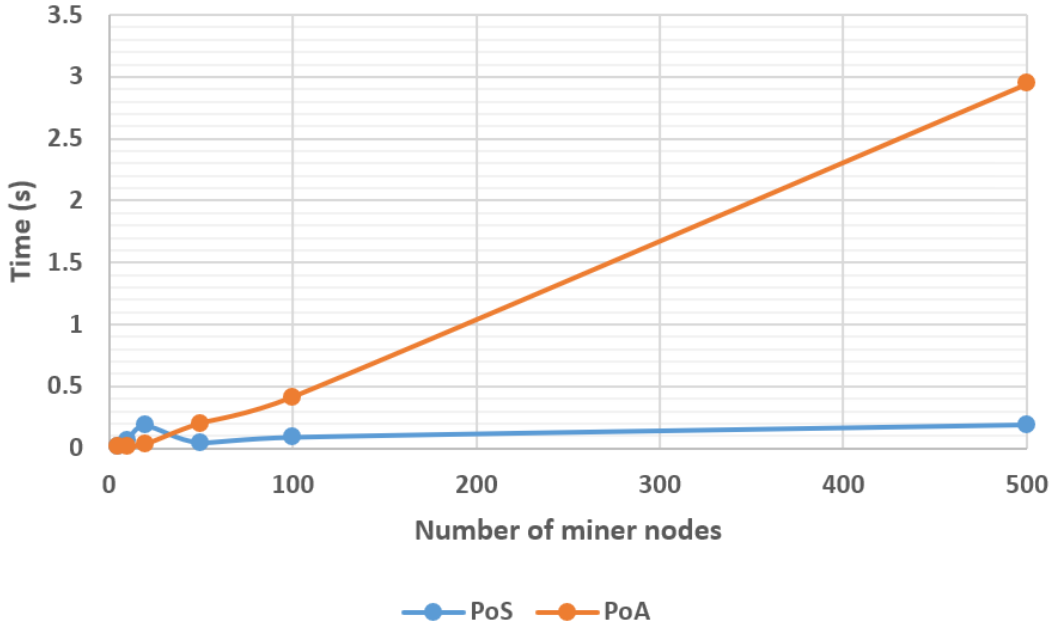

Figure 11. Average block confirmation time consumed by PoS-based BC vs. PoA-based BC, relatively to the number of miner nodes

Simulation parameter\Consensus	PoW	PoS	PoA
no. of miners	5–500	5–500	5–500
no. of neighbours per miner	4	4	4
puzzle difficulty	5–20	–	–
Authorized miners	All	Random choice	2–25
Initial wallet	–	1000	–
BC functionality	Data Management	Data Management	Data Management
BC deployment	end-user layer	end-user layer	end-user layer

Table 9. Simulation parameters configuration for Case 1

In this case study, we run FoBSim several times, with which we deploy different CAs under similar conditions. The simulation runs targeted specifically the measurement of the average time consumed by each CA, from the moment where a miner is triggered to mint a new block, until the minted block by this miner is confirmed by, at least, 51% of other BC miners. To accurately measure this average, we added some variables holding the starting time and the elapsed time, exactly before calling the build_block() function and right after a block is confirmed by reaching the required number of confirmations.

As described in Table 9, we changed the difficulty of the puzzle during the PoW-based BC simulation runs from an easy level (5), to a harder level (10), and finally to very hard levels (15) and (20). During the runs where PoA was used, we changed the number of authorized miners from 2/5 (2 authorized out of a total of 5 miners), 5/10, 10/20, and 25 authorized miners for the rest of runs.

As we wanted to abstractly measure the average confirmation time, we avoided the Computational Services and the Payment functionality, because both imply extra time consumption for performing the computational tasks, and validating the payments, respectively. We also avoided the Identity management functionality because the number of TXs per end-user is limited by the number of ID attributes required to be saved on the chain. Hence, our best choice was the data management functionality. We kept the total number of TXs delivered to the mempool unchanged, which gives equivalent input for all simulation runs. However, we changed the number of TXs generated by each user as to be equal to the number of miners in each run. More precisely, as the total number of TXs is determined using Equation 2, where$a$, b and c are the number of fog nodes, the number of end-users, and the number of TXs per end-user, respectively,

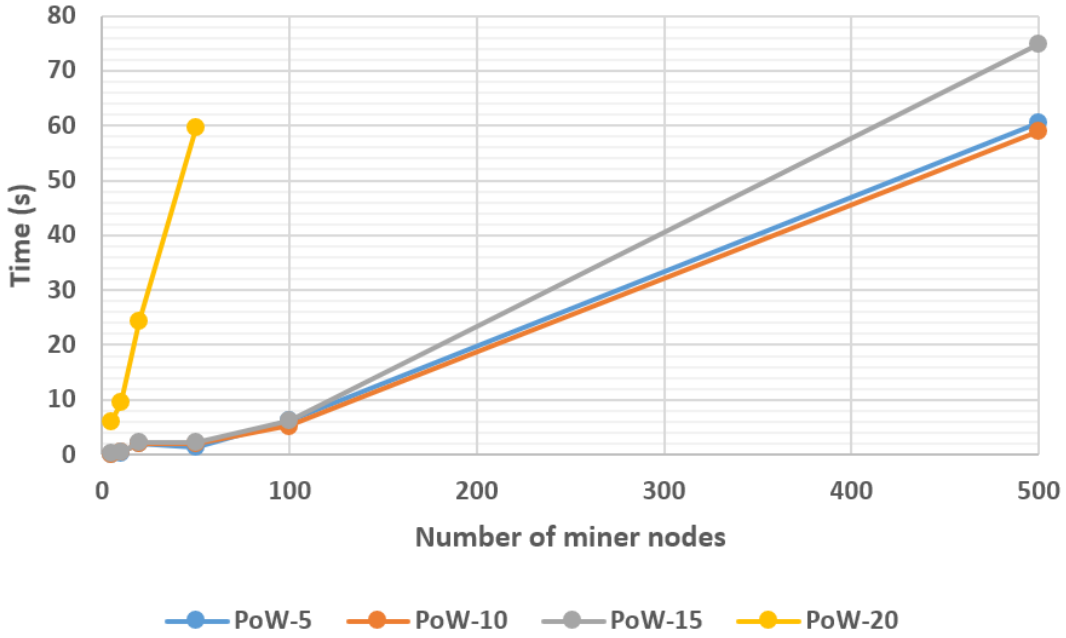

Figure 12. Average block confirmation time consumed by PoW-based BC (the cases of difficulty = 5, 10, 15, and 20), relatively to the number of miner nodes

	M=5	M=10	M=20	M=50	M=100	M=500
PoS algorithm	0.018	0.06	0.18	0.046	0.09	0.19
PoA algorithm	0.002	0.008	0.03	0.2	0.41	2.94
PoW-5 algorithm	0.08	0.36	2.1	1.31	6.15	60.6
PoW-10 algorithm	0.07	0.44	2.1	2.03	5.21	58.9
PoW-15 algorithm	0.25	0.42	2.23	2.26	6.18	74.76
PoW-20 algorithm	6.02	9.5	24.2	59.62	–	–

Table 10. Results of Case-1, where the PoW puzzle difficulty ranged from 5 to 20, and the number of Miners (M) ranged from 5 to 500.

the values of those variables fluctuated in each run. Concerning the runs where a PoS is deployed, miner nodes were initiated with a wallet that has 1000 coins, allowing miners to stake random amounts of coins. Additionally, winning miners were awarded 5 coins for each confirmed block they had minted.

$$|TXs| = a \times b \times c \tag{2}$$

We deployed the FoBSim environment on Google Cloud Platform, using a C2-standard-16 (up to 3.8 GHz, 16 vCPUs, 64 GB memory), with Debian OS. We have chosen to place the BC in the end-user layer for all runs, not for any reason other than testing the reliability and stability, of FoBSim components and results , in such complex inter-operable [85] Edge-Fog-BC scenarios. Table 10 presents the exact results we obtained, which are depicted in Figures 11 and 12.

According to the results obtained from the simulation runs, one can notice that PoW-based BCs consume much more time, to confirm a block, than PoA and PoS -based BCs, which is inline with the theoretical and experimental results of most previous research. Additionally, the average block confirmation time, in PoW-based and PoA-based BCs, seems to be directly proportional to the BC network size, which complies with the results recently presented in [86]. Comparatively, an average block confirmation time in a PoS-based BC seems unaffected by the network size, which complies with the expectations recently presented in [87].

Case 2: Capturing the effect using the Gossip protocol

In this case, we compare the number of chain forks at the end of several simulation runs, where we interchangeably activate and deactivate the gossiping property in a PoW-based BC. Accordingly, one can notice the effect of gossiping on ledger finality under different conditions, namely the puzzle difficulty and the transmission delay between miners. As it was mentioned in Subsection 5.3.1, gossiping is a continuous process during the life time of the network, which implies that miners would mostly have different chain versions at any given moment. In this case, we detect the number of chain versions at the end of simulation runs, which can be decreased to one version under strictly designed parameters, such medium network size, high puzzle difficulty, low transmission delay, low number of neighbors per miner, etc. Nevertheless, our goal in this case is to demonstrate how the activation of the gossiping property during a simulation run on FoBSim can decrease the number of chain versions and, thus, it can positively contribute to the consistency of the distributed ledger. For this case, we also deployed the FoBSim environment on the Google Cloud Platform, using a C2-standard-16 VM (up to 3.8 GHz, 16 vCPUs, 64 GB memory), with Ubuntu OS.

Table 11 presents the initial configuration in each simulation scenario, while Tables 12 and 13 present the results we obtained by running the described scenarios, which are depicted in Figures 13 and 14. As can be noted from the results, the default gossip protocol in FoBSim could decrease the number of chain versions at the end of each simulation run. Although the number of chain versions did not reach the optimum value (i.e. one chain version), it is obvious that activating the gossiping property decreases the number of chain versions at each simulation run and, thus, enhances the distributed ledger consistency.

Simulation parameter	Puzzle difficulty effect	Transmission delay effect
no. of Fog Nodes	5	5
no. of users per fog node	5	5
no. of TX per user	5	5
no. of miners	100	100
no. of neighbours per miner	2	2
no. of TX per Block	5	5
puzzle difficulty	5, 10, 15, 20	20
Max enduser payment	100	100
miners initial wallet value	100	100
mining award	5	5
delay between neighbors	0	0, 5, 10, 15, 20

Table 11. Simulation parameters configuration for Case-2, where the Gossiping property is interchangeably activated and deactivated

Configuration	diff.=5	diff.=10	diff.=15	diff.=20
Gossip activated	81	70	57	16
Gossip deactivated	92	98	100	67

Table 12. Results of Case-2, where the puzzle difficulty ranged from 5–20, and the Gossiping in FoBSim was interchangeably activated and deactivated

Configuration	T.D.=0	T.D.=5	T.D.=10	T.D.=15	T.D.=25
Gossip activated	12	18	14	26	33
Gossip deactivated	15	39	59	68	76

Table 13. Results of Case-2, where the transmission delay between neighbors ranged from 0–25 ms., and the Gossiping in FoBSim was interchangeably activated and deactivated

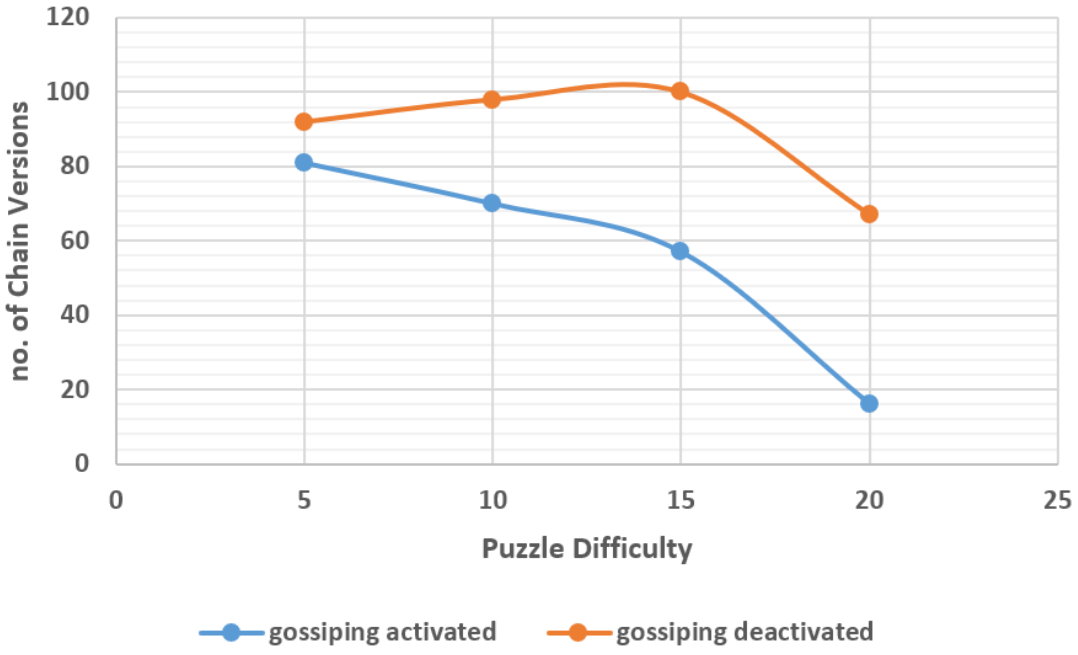

Figure 13. The effect of activating the gossiping protocol in FoBSim, on the number of chain versions at the end of PoW-based BC simulation runs, where the puzzle difficulty fluctuates from 5 to 20

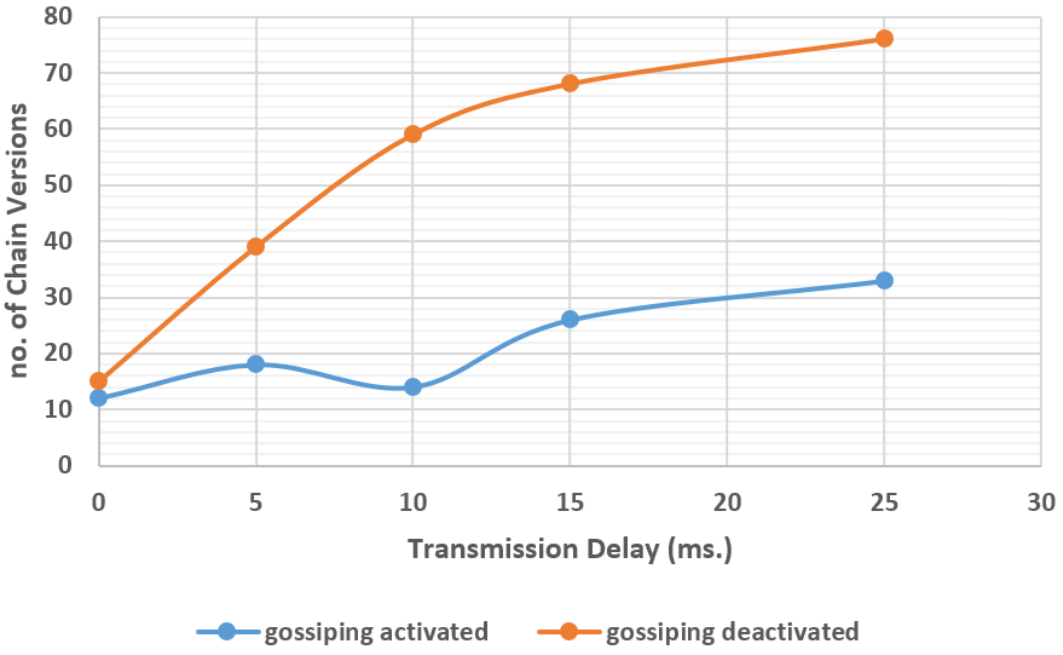

Figure 14. The effect of activating the gossiping protocol in FoBSim, on the number of chain versions at the end of PoW-based BC simulation runs, where the transmission delay between neighboring miners fluctuates from 0 to 25 ms.

7 CONCLUSIONS

In this paper, we proposed a novel simulation tool called FobSim that mimics the interaction between the entities of an integrated Fog-Blockchain system. We briefly described the architectural elements of Fog Computing (FC) and Blockchain (BC) technologies, and designed FoBSim in order to cover all the elements we described. We deployed three different consensus algorithms, namely PoW, PoS and PoA, and different deployment options of the BC in an FC architecture, namely the end-user layer and the fog layer. Additionally, we fine tuned the FoBSim modules so that various services, provided by FC and BC, can be adopted for any proposed integration scenario. The services that can be simulated are distributed Payment services, distributed Identity services, distributed Data storage and distributed Computational services (through Smart Contracts). In our paper, we described the modules of FoBSim, the transaction modelling, the Genesis block generation, the gossiping in FoBSim, the Consensus Algorithms, transaction and block validation, incentive mechanisms, and other FoBSim strategies. We validated FoBSim with two case studies: the first compares the average time consumption for block confirmation in different consensus algorithms, while the second analyzes the effect of gossiping on the consistency of the distributed ledger, in fluctuated puzzle difficulty and transmission delay configurations.

In the future releases of FoBSim, we are willing to make more CAs available, as well as enhancing the identity management scheme in FoBSim. We will further investigate adding the Reputation management service in a generalized and simple manner so that analysis can be provided, while proposed reputation management ideas, conditions, or properties can be easily implemented/modified.

ACKNOWLEDGEMENT

This research was supported by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793, and by the Hungarian Government under the grant number EFOP-3.6.1-16-2016-00008.

REFERENCES

[1] Smart Dubai Department. *BLOCKCHAIN*. 2020 (accessed October, 27, 2020). URL: https://www.smartdubai.ae/initiatives/blockchain.

[2] Global Times. *China launches blockchain-based smart city identification system*. 2019 (accessed October, 27, 2020). URL: https://www.globaltimes.cn/content/1168878.shtml.

[3] Smartcity Press. *China Taking A Big Leap With Blockchain*. 2019 (accessed October, 27, 2020). URL: https://www.smartcity.press/blockchain-technology-china/.

[4] Roman Beck et al. *Blockchain technology in business and information systems research*. 2017.

[5] Bitcoin.org. *Bitcoin is an innovative payment network and a new kind of money.* 2009 (accessed October, 27, 2020). URL: https://bitcoin.org/en/.

[6] Evangelos K Markakis et al. "EXEGESIS: Extreme edge resource harvesting for a virtualized fog environment". In: *IEEE Communications Magazine* 55.7 (2017), pp. 173–179.

[7] Pooyan Habibi et al. "Fog Computing: A Comprehensive Architectural Survey". In: *IEEE Access* 8 (2020), pp. 69105–69133.

[8] Amir Vahid Dastjerdi et al. "Fog computing: Principles, architectures, and applications". In: *Internet of things*. Elsevier, 2016, pp. 61–75.

[9] OpenFog Consortium et al. "OpenFog reference architecture for fog computing". In: *Architecture Working Group* (2017), pp. 1–162.

[10] Flavio Bonomi et al. "Fog computing: A platform for internet of things and analytics". In: *Big data and internet of things: A roadmap for smart environments*. Springer, 2014, pp. 169–186.

[11] Hamza Baniata and Attila Kertesz. "A Survey on Blockchain-Fog Integration Approaches". In: *IEEE Access* 8 (2020), pp. 102657–102668.

[12] Alberto Montresor and Márk Jelasity. "PeerSim: A scalable P2P simulator". In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. IEEE. 2009, pp. 99–100.

[13] Ioan Petri et al. “Blockchain for energy sharing and trading in distributed prosumer communities”. In: *Computers in Industry* 123 (2020), p. 103282.

[14] Jari Kreku et al. “Evaluating the Efficiency of Blockchains in IoT with Simulations.” In: *IoTBDS*. 2017, pp. 216–223.

[15] Sotirios Liaskos, Tarun Anand, and Nahid Alimohammadi. “Architecting blockchain network simulators: a model-driven perspective”. In: *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. 2020, pp. 1–3.

[16] Andras Markus and Attila Kertesz. “A survey and taxonomy of simulation environments modelling fog computing”. In: *Simulation Modelling Practice and Theory* 101 (2020), p. 102042.

[17] Zahra Nikdel, Bing Gao, and Stephen W Neville. “DockerSim: Full-stack simulation of container-based Software-as-a-Service (SaaS) cloud deployments and environments”. In: *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE. 2017, pp. 1–6.

[18] Tariq Qayyum et al. “FogNetSim++: A toolkit for modeling and simulation of distributed fog environment”. In: *IEEE Access* 6 (2018), pp. 63570–63583.

[19] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. “Edgecloudsim: An environment for performance evaluation of edge computing systems”. In: *Transactions on Emerging Telecommunications Technologies* 29.11 (2018), e3493.

[20] Ubaid Ur Rahman et al. “Nutshell—Simulation Toolkit for Modeling Data Center Networks and Cloud Computing”. In: *IEEE Access* 7 (2019), pp. 19922–19942.

[21] Rodrigo N Calheiros et al. “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms”. In: *Software: Practice and experience* 41.1 (2011), pp. 23–50.

[22] Ashkan Yousefpour et al. “All one needs to know about fog computing and related edge computing paradigms: A complete survey”. In: *Journal of Systems Architecture* 98 (2019), pp. 289–330.

[23] Harshit Gupta et al. “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments”. In: *Software: Practice and Experience* 47.9 (2017), pp. 1275–1296.

[24] Mohammed Islam Naas et al. “An extension to ifogsim to enable the design of data placement strategies”. In: *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*. IEEE. 2018, pp. 1–8.

[25] Ruben Mayer et al. “Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures”. In: *2017 IEEE Fog World Congress (FWC)*. IEEE. 2017, pp. 1–6.

[26] Antonio Coutinho et al. “Fogbed: A rapid-prototyping emulation environment for fog computing”. In: *2018 IEEE International Conference on Communications (ICC)*. IEEE. 2018, pp. 1–7.

[27] Márcio Moraes Lopes et al. “Myifogsim: A simulator for virtual machine migration in fog computing”. In: *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. 2017, pp. 47–52.

[28] Tanesh Kumar et al. “BlockEdge: Blockchain-Edge Framework for Industrial IoT Networks”. In: *IEEE Access* (2020).

[29] Hamza Baniata. “Fog-enhanced Blockchain Simulation”. In: *The 12th Conference of PhD Students in Computer Science (CS2)*. University of Szeged. 2020, pp. 83–87.

[30] Vysakh Anilkumar et al. “Blockchain Simulation and Development platforms: Survey, Issues and Challenges”. In: *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. IEEE. 2019, pp. 935–939.

[31] Ethereum. *Remix Platform*. 2020 (accessed October, 27, 2020). URL: `https://remix.ethereum.org/`.

[32] Truffle Blockchain Group. *TRUFFLE OVERVIEW*. 2020 (accessed October, 27, 2020). URL: `https://www.trufflesuite.com/docs/truffle/overview`.

[33] Arshdeep Bahga and Vijay Madisetti. *Blockchain applications: a hands-on approach*. Vpt, 2017.

[34] Bruno. *Explaining Ethereum Tools: What Are Geth and Mist?* 2018 (accessed October, 27, 2020). URL: https://bitfalls.com/2018/02/12/explaining-ethereum-tools-geth-mist/.

[35] Maher Alharby and Aad van Moorsel. "Blocksim: a simulation framework for blockchain systems". In: *ACM SIGMETRICS Performance Evaluation Review* 46.3 (2019), pp. 135–138.

[36] Carlos Faria and Miguel Correia. "BlockSim: Blockchain Simulator". In: *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2019, pp. 439–446.

[37] Bozhi Wang et al. "A simulation approach for studying behavior and quality of blockchain networks". In: *International Conference on Blockchain*. Springer. 2018, pp. 18–31.

[38] Arthur Gervais et al. "On the security and performance of proof of work blockchains". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 3–16.

[39] Raheel Ahmed Memon et al. "Modeling of blockchain based systems using queuing theory simulation". In: *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. IEEE. 2018, pp. 107–111.

[40] Fangyuan Zhao, Xin Guo, and Wai Kin Victor Chan. "Individual Green Certificates on Blockchain: A Simulation Approach". In: *Sustainability* 12.9 (2020), p. 3942.

[41] Pierre-Yves Piriou and Jean-Francois Dumas. "Simulation of stochastic blockchain models". In: *2018 14th European Dependable Computing Conference (EDCC)*. IEEE. 2018, pp. 150–157.

[42] Aditya Deshpande, Pezhman Nasirifard, and Hans-Arno Jacobsen. "eVIBES: Configurable and Interactive Ethereum Blockchain Simulation Framework". In: *Proceedings of the 19th International Middleware Conference (Posters)*. 2018, pp. 11–12.

[43] Bozhi Wang et al. "Security analysis on tangle-based blockchain through simulation". In: *Australasian Conference on Information Security and Privacy*. Springer. 2020, pp. 653–663.

[44] Ravi Kiran Raman et al. "A Scalable Blockchain Approach for Trusted Computation and Verifiable Simulation in Multi-Party Collaborations". In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. 2019, pp. 277–284.

[45] The Linux Foundation. *What is Hyperledger?* 2020 (accessed October, 27, 2020). URL: https://www.hyperledger.org/.

[46] Mozhdeh Farhadi et al. "A systematic approach toward security in Fog computing: Assets, vulnerabilities, possible countermeasures". In: *Software: Practice and Experience* 50.6 (2020), pp. 973–997.

[47] Stephane Herman Maes et al. *Orchestrating hybrid cloud services*. US Patent 9,882,829. Jan. 2018.

[48] Andrea Coladangelo and Or Sattath. "A Quantum Money Solution to the Blockchain Scalability Problem". In: *arXiv preprint arXiv:2002.11998* (2020).

[49] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. Manubot, 2019.

[50] Dejan Vujičić, Dijana Jagodić, and Siniša Ranđić. "Blockchain technology, bitcoin, and Ethereum: A brief overview". In: *2018 17th international symposium infoteh-jahorina (infoteh)*. IEEE. 2018, pp. 1–6.

[51] Sunny King and Scott Nadal. "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake". In: *self-published paper, August* 19 (2012), p. 1.

[52] Daniel Larimer. "Transactions as proof-of-stake". In: *Nov-2013* (2013).

[53] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. "Cryptocurrencies without proof of work". In: *International conference on financial cryptography and data security*. Springer. 2016, pp. 142–157.

[54] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. "A survey on long-range attacks for proof of stake protocols". In: *IEEE Access* 7 (2019), pp. 28712–28725.

[55] Rong Zhang and Wai Kin Victor Chan. "Evaluation of Energy Consumption in Block-Chains with Proof of Work and Proof of Stake". In: *Journal of Physics: Conference Series*. Vol. 1584. 1. IOP Publishing. 2020, p. 012023.

[56] JP Buntinx. "What is Proof of Elapsed Time". In: *The Merkle Hash. Available online: https://themerkle. com/what-is-proof-of-elapsed-time/(accessed on 8 August 2020)* (2017).

[57] Anushree A Avasthi and Ankur Saxena. "Two Hop Blockchain Model: Resonating between Proof of Work (PoW) and Proof of Authority (PoA)". In: *International Journal of Information Systems & Management Science* 1.1 (2018).

[58] J Manning. "Proof-of-work vs. proof-of-stake explained". In: *ETHNews, November. Available at: https://www. ethnews. com/proof-of-work-vs-proofof-stake-explained (Accessed: 6 January 2018)* (2016).

[59] Pranav Kumar Singh et al. "Managing smart home appliances with proof of authority and blockchain". In: *International Conference on Innovations for Community Services*. Springer. 2019, pp. 221–232.

[60] Lin Chen et al. "On security analysis of proof-of-elapsed-time (poet)". In: *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer. 2017, pp. 282–297.

[61] Hamza Baniata and Attila Kertész. "PF-BVM: A Privacy-aware Fog-enhanced Blockchain Validation Mechanism." In: *CLOSER*. 2020, pp. 430–439.

[62] Hamza Baniata, Ahmad Anaqreh, and Attila Kertesz. "PF-BTS: A Privacy-aware Fog-enhanced Blockchain-assisted Task Scheduling". In: *Information Processing and Management* 58 (2021).

[63] Hamza Baniata and Attila Kertesz. *FoBSim*. 2020 (accessed October, 27, 2020). URL: https://github.com/sed-szeged/FobSim.

[64] Hamza Baniata, Wesam Almobaideen, and Attila Kertesz. "A Privacy Preserving Model for Fog-enabled MCC systems using 5G Connection". In: *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2020, pp. 223–230.

[65] Piotr Fröhlich, Erol Gelenbe, and Mateusz P Nowak. "Smart SDN management of fog services". In: *2020 Global Internet of Things Summit (GIoTS)*. IEEE. 2020, pp. 1–6.

[66] Mazin Debe et al. "IoT public fog nodes reputation system: A decentralized solution using Ethereum blockchain". In: *IEEE Access* 7 (2019), pp. 178082–178093.

[67] Bastian Blywis et al. *A survey of flooding, gossip routing, and related schemes for wireless multi-hop networks*. Tech. rep. Berlin, Germany: Freie Universitat, 2011. URL: https://refubium.fu-berlin.de/bitstream/handle/fub188/18401/2010-Gossip-Routing.pdf?sequence=1.

[68] Xiaowei He, Yiju Cui, and Yunchao Jiang. "An Improved Gossip Algorithm Based on Semi-Distributed Blockchain Network". In: *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE. 2019, pp. 24–27.

[69] David Kempe, Alin Dobra, and Johannes Gehrke. "Gossip-based computation of aggregate information". In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* IEEE. 2003, pp. 482–491.

[70] Wilfried N Gansterer et al. "Scalable and fault tolerant orthogonalization based on randomized distributed data aggregation". In: *Journal of Computational Science* 4.6 (2013), pp. 480–488.

[71] Gábor Danner and Márk Jelasity. "Robust decentralized mean estimation with limited communication". In: *European Conference on Parallel Processing*. Springer. 2018, pp. 447–461.

[72] Caixiang Fan et al. "Performance Evaluation of Blockchain Systems: A Systematic Survey". In: *IEEE Access* 8 (2020), pp. 126927–126950.

[73] Jiang Lan et al. "Consistency maintenance in peer-to-peer file sharing networks". In: *Proceedings the Third IEEE Workshop on Internet Applications. WIAPP 2003*. IEEE. 2003, pp. 90–94.

[74] Nabhendra Bisnik and Alhussein A Abouzeid. "Optimizing random walk search algorithms in P2P networks". In: *Computer networks* 51.6 (2007), pp. 1499–1514.

[75] The_Python_Software_Foundation. *The Python Standard Library*. https://docs.python.org/3/library/multiprocessing.shared_memory.html. 2020 (Last accessed September 14, 2020).

[76] The_Python_Software_Foundation. *The Python Standard Library*. https://docs.python.org/2/library/multiprocessing.html. 2020 (Last accessed September 14, 2020).

[77] Arvind Narayanan et al. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[78] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. "Double-spending fast payments in bitcoin". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 906–917.

[79] Kevin Alarcón Negy, Peter R Rizun, and Emin Gün Sirer. "Selfish Mining Re-Examined". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2020, pp. 61–78.

[80] Ali Hassan Sodhro et al. "Towards Blockchain-Enabled Security Technique for Industrial Internet of Things Based Decentralized Applications". In: *Journal of Grid Computing* (2020), pp. 1–14.

[81] Umer Majeed, Kitae Kim, and Choong Seon Hong. "Mining Pool Selection Strategy in Blockchain Networks: A Probabilistic Approach". In: *KIISE Transactions on Computing Practices* 26.6 (2020), pp. 280–285.

[82] Robert G Sargent. "Verification and validation of simulation models". In: *Journal of simulation* 7.1 (2013), pp. 12–24.

[83] Stefano De Angelis et al. "PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain". In: ().

[84] Binance Academy. *Proof of Authority Explained*. 2020 (accessed October, 27, 2020). URL: https://academy.binance.com/en/articles/proof-of-authority-explained.

[85] Rafael Belchior et al. "A Survey on Blockchain Interoperability: Past, Present, and Future Trends". In: *arXiv preprint arXiv:2005.14282* (2020).

[86] Jelena Misic, Vojislav B Misic, and Xiaolin Chang. "Performance of Bitcoin network with synchronizing nodes and a mix of regular and compact blocks". In: *IEEE Transactions on Network Science and Engineering* (2020).

[87] Bin Cao et al. "Performance analysis and comparison of PoW, PoS and DAG based blockchains". In: *Digital Communications and Networks* (2020).