Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2022) 30: 392 – 405
© TÜBİTAK
doi:10.3906/elk-2105-184

BlockSim-Net: a network-based blockchain simulator

Prashanthi Ramachandran[1], **Nandini Agrawal**[1], **Osman Biçer**[2], **Alptekin Küpçü**[2]
[1]Computer Science Department, Ashoka University, Sonipat, India.
[2]Computer Engineering, Koç University,İstanbul, Turkey

Received: 22.05.2021 • **Accepted/Published Online:** 04.01.2022 • **Final Version:** 04.02.2022

Abstract: Since its proposal by Eyal and Sirer (CACM '13), selfish mining attacks on proof-of-work blockchains have been studied extensively. The main body of this research aims at both studying the extent of its impact and defending against it. Yet, before any practical defense is deployed in a real world blockchain system, it needs to be tested for security and dependability. However, real blockchain systems are too complex to conduct any test on or benchmark the developed protocols. Instead, some simulation environments have been proposed recently, such as BlockSim (Maher et al., SIGMETRICS Perform. Eval. Rev. '19), which is a modular and easy-to-use blockchain simulator. However, BlockSim's structure is insufficient to capture the essence of a real blockchain network, as the simulation of an entire network happens over a single CPU. Such a lack of decentralization can cause network issues such as propagation delays being simulated in an unrealistic manner. In this work, we propose BlockSim-Net, a modular, efficient, high performance, distributed, network-based blockchain simulator that is parallelized to better reflect reality in a blockchain simulation environment.

Key words: Bitcoin, blockchain network, simulator, decentralization

1. Introduction

Proposed by Nakamoto[1], blockchain has applications in cryptocurrencies (e.g., Bitcoin, Ethereum[2], Litecoin[3]), certificate transparency[4] [1], governmental services [2], etc. This is due to its transparency and immutability features [2–4]. Blockchains are maintained by peer-to-peer (P2P) networks, composed of nodes called "miners". Miners are incentivized by a competition through a process of mining, i.e., a trial-and-error process conducted via invested computational resources. Nakamoto's security assumption for Bitcoin has been that as long as a majority of the computational resources belongs to honest miners, the system will award each miner with their fair share. In other words, the reward that each miner receives will be proportional to their invested resources or "hashing power". [5] has shown that the blockchain shows immutability of the history if honest miners are majority. However, in 2013, Eyal and Sirer [6] showed that an adversarial miner can obtain more than their fair share even without having a majority of the total hashing power. Since then, there has been extensive research

[1] S. Nakamoto (2008). Bitcoin: A peer-to-peer electronic cash system. Website https://bitcoin.org/bitcoin.pdf [accessed 15 April 2021].

[2] V. Buterin (2013). Ethereum White Paper: A next-generation smart contract and decentralized application platform. Website https://ethereum.org/en/whitepaper/ [accessed 15 April 2021].

[3] Website https://litecoin.org/ [accessed 15 April 2021].

[4] Website http://www.certificate-transparency.org/ [accessed 15 April 2015].

to optimize this attack [7, 8], and to defend against it [9, 10][5]. The proposal of these improvements clarified the need for a dependable environment to conduct the tests. However, real blockchain systems are too large and complex for these purposes.

To clarify the effects of selfish mining attacks and defences, some general simulation environments have been developed in recent times [11–13]. Although it is possible to show these effects theoretically as well, it is generally reasonable to use tools for support. Also, sometimes we see that mathematical analysis is complicated to perform or error-prone. Among these environments, we focus on BlockSim [13], which is a versatile blockchain simulator. While BlockSim might suffice for certain security tests, since it simulates everything on a single CPU, it fails to be a network-based simulator. A network-based platform would provide a more realistic simulation of the real blockchain environments, since network delays will be automatically integrated under real network conditions. This can also help simulation of other attacks and issues associated with the network itself, e.g., network randomness due to propagation delays, and oracle and bold mining attacks [5]. Factoring these ideas, we propose BlockSim-Net, a simple, easy-to-use, highly-efficient, network-based blockchain simulator for use in blockchain security research on both attacks and defences. Additionally, our simulator can also be used for beta-testing and optimization benchmarking.

1.1. Our contributions

We build on top of an existing work, BlockSim [13], which simulates a blockchain on a local system. The contribution of this work is two-fold:

1. **From local to network:** Our main contribution is converting the BlockSim's [13] local simulation to a network simulation. As opposed to a local simulation, a network simulation is more distributed, decentralized, and comparable to a real blockchain, since multiple miners from different locations can be involved in the simulation of a blockchain network together. Such a simulator can also be used to effectively simulate network-based attacks.

2. **Efficiency:** Since simulation on the network comes with associated heavy communication costs, our work also includes multiple optimizations. The main optimizations are mentioned below: Switching to another miner's blockchain when a block is received: Since [13] simulates on a single system, miner nodes can easily access the local blockchains of other miner nodes. Over a network, however, to switch to another miner's blockchain, a node would need to receive the entire blockchain of that miner. This would mean high communication costs. Thus, instead of that, we propose the use of a local dictionary that stores all the blocks that the miner receives during the simulation. Most of the time, using this local dictionary, miners can recreate the chain that was supposed to be received with minimal communication. However, in the case where a required block is lost in communication, recreating the blockchain may not be possible. Consensus: In [13], since the entire simulation happens on a single system, the main thread accesses the longest local blockchains of all miners at the end of the simulation and decides on the longest chain. However, over a network of miners, each miner would have to send their longest local blockchain to the admin server for consensus (refer Section 3.7). However, the propagation of the entire blockchain would be very costly in terms of communication complexity. Instead, we propose that miners only send the last block of their longest local blockchain to the admin server for consensus. We elaborate on why the last block is enough for consensus in Section 3.7.

[5] O. Biçer and A. Küpçü (2020). FORTIS: Selfish Mining Mitigation by (FOR)geable (TI)me(S)tamps. Website https://eprint.iacr.org/2020/1290.pdf [accessed 15 April 2021].

1.2. Related work

Simulators. In the recent past, there have been several proposed blockchain simulators. SimBlock [12], proposed in 2019, is a blockchain simulator in Java that is mainly designed to study the impact of node behaviour on the network. Their experiments have proven that SimBlock provides a good picture of a real blockchain. Similarly, BlockSim [13], proposed in 2020, is another blockchain simulator in Python that captures the network, incentive, and consensus layers of a blockchain. It is a modular, intuitive, and easy-to-use framework that allows the user to simulate a blockchain network of a multitude of nodes. Talaria[6], proposed in 2021, is another simulator that extends the capability of BlockSim by including permissioned-blockchains. However, it is important to note that SimBlock, BlockSim, and Talaria simulate the whole network on a single CPU. This means that the complications associated with the network and communication between nodes cannot be realistically captured. While [12, 13] and Talaria are highly scalable, BlockSim-Net is better for security analyses because it is a closer approximation of an actual blockchain network.

Some more recent blockchain simulators include VIBES [11] and Blockchain Demo[7]. VIBES is a simulator that allows users to specify configurations to create and simulate custom blockchains. It works on real networks, and this makes the simulation bulky, complex, and not scalable. Blockchain Demo is a web-based simulator that allows a user to modify transactions and node configurations on the interface, and thereby understand the inner workings of blockchains. SimBlock and VIBES can be used to understand Blockchain networks visually. Table 1 provides a comparative summary of existing simulators.

Table 1. A qualitative comparison between state-of-the-art simulators. “Paral.” is an abbreviation for “Parallelizable.”

	Paral.	Propagation of Blocks	Simulation	Scalability
BlockSim	55	Simulation	Single CPU	Depends on the power of the CPU
SimBlock	55	Simulation	Single CPU	Depends on the power of the CPU
VIBES	51	Actual propagation	On a real blockchain	Bulky; hard to scale
Blockchain Demo	55	Simulated	Web-based visualization	N/A
Talaria	55	Simulated	Single CPU	Depends on the power of the CPU
BlockSim-Net	51	Over multiple servers	Over a network	Depends on the power of the servers

Selfish mining: In the original Bitcoin paper, Nakamoto works on the assumption that as long as the majority of the hashing power belongs to the honest miners, every miner gets a reward that is proportional to their work. However, Eyal and Sirer [6] have introduced the concept of selfish mining, which allows a miner to earn more than her fair share, even when the honest majority assumption is true. The primary idea behind selfish mining is that a seemingly-honest miner can deviate from the honest protocol by populating a private chain until it becomes longer than the existing longest chain in the network. By extending on a private chain, a miner can potentially eliminate blocks in the network and get rewarded more than one’s fair share. Several selfish mining defenses [7, 8, 14, 15] have also been proposed by various works against attacks proposed by [6, 9, 10, 16]. In the original Bitcoin implementation, in case multiple chains with the same number of blocks occur, miners choose the one that they receive first. However, [6] proposes choosing one of the chains uniformly at random. This defense prevents attackers with less than 25% of the total hashing power of the network from obtaining

[6] J. Xing, D. Fischer, N. Labh, R. Piersma, B. C. Lee, Y. A. Xia, T. Sahai, and V. Tarokh (2021). Talaria: A Framework for Simulation of Permissioned Blockchains for Logistics and Beyond. Website https://arxiv.org/abs/2103.02260 [accessed 15 April 2021].

[7] Website https://andersbrownworth.com/blockchain/, [accessed 30th April 2021].

high rewards. However, the later attack proposal by [7] on Optimal Selfish Mining reduces this value to 23.2%. [10] proposes the publish or perish scheme and this takes the value back to 25%. Given this trend of proposals of selfish mining and network-based attacks and defenses, blockchain security research needs to constantly work to improve with better defenses. In this scenario, there arises a need for simulators that can simulate a real network as realistically as possible in order for defenses to be tried and tested.

Lastly, a preliminary version of this paper was presented at the BAŞARIM 2020 conference, but it was not published in any proceedings.

2. Preliminaries

Proof-of-Work Blockchains. A blockchain is a chain of blocks that contain information about various transactions happening in the network. Each block in the chain consists of the hash of the previous block (i.e., the parent block) in the chain, the depth of the current block, a nonce, and a difficulty target. In some models (e.g., GHOST protocol [17]), a block may also refer to an uncle chain, which is a separate chain that comprises of orphaned blocks built on top of the parent block. Each miner in the network constantly attempts to mine a block. To mine a block, a miner should be able to create a block such that the hash of the block is lower than the difficulty target. The miner to achieve this first gets a reward. Creating a block such that the hash of the block is less than the difficulty target requires computational power and, consequently, a certain amount of time. The network is designed such that the miners adjust the difficulty target from time to time in order to ensure that the inter-block time is fixed even if the computational resources of the miners vary. For instance, in Bitcoin, the inter-block time is fixed to approximately 10 minutes. Therefore, every miner that is able to mine a block can be said to have done some ”work”, which is easy to verify. Since the network constantly updates the difficulty target based on the computational power of the miners, every miner that is able to mine a block is said to have demonstrated ”proof-of-work”.

BlockSim: BlockSim and BlockSim-Net deal with mainly three layers of blockchain: the *network layer*, the *consensus layer*, and the *incentive layer*. The network layer mainly copes with the propagation of blocks and transactions in the blockchain system. The consensus layer ensures the protocol for honest nodes to come into consensus in the main chain. The incentive layer deals with miners' incentives for playing the honest mining strategy by using transaction fees and block rewards. The main differences of this work from BlockSim are on the simulation of the network and the consensus layers. Algorithm 1 describes BlockSim's simulation of a blockchain network.

In Algorithm 1, `current_time` refers to the current time in the simulation at any given point. It starts with 0 at the beginning and goes up to `sim_time`, which marks the end of the simulation. At the beginning of the simulation, the `current_time` is set to 0, and the hash power of each of the `num_nodes` nodes are randomly initialized. Afterwards, a transaction pool and a genesis block are created centrally for the simulation. Further, this genesis block is appended to each node's local blockchain. Later, each node starts mining on top of the genesis block. In this simulation, mining is essentially the creation of block ”events”. Based on a node's hashing power, BlockSim calculates a time `t` (in seconds) that would be sufficient to show proof of work. Therefore, when a block is said to be mined, an ”event” is created with its timestamp set to `current_time + t`.

During the simulation, the block event that is scheduled for the earliest point in time is fetched from the queue by calling `get_next_event()`. The current time is then set to the timestamp associated with the event. If the block event is of a ”create” type, then a block object is created and appended to the miner's local blockchain if and only if the block is built on top of the last block in the miner's local blockchain. This block is

Algorithm 1 Simulation in BlockSim.

```
procedure $\mathcal{F}_{blocksim}$(num_nodes, sim_time)
    current_time = 0
    set_hash_power(num_nodes)
    create a transaction pool
    create genesis block and append to each node's local blockchain
    for node in all_nodes do
        mine(node, current_time)
    end for
    while (!empty(Queue) and current_time < sim_time) do
        event = get_next_event()
        current_time = event.time
        run_event(event)
        remove_from_queue(event)
    end while
    lb = get_longest_chain(all_nodes)
    set_global_blockchain(lb)
    print_consensus_stats()
end procedure
```

then sent to the other miners in the network. However, if this block event is of a ”receive” type, all the miners (except the miner of this block event) verify whether certain conditions are met. In the case that the block satisfies these conditions, the miners append it to their local blockchain and start mining on top of it. This is done repeatedly until the simulation end time is reached.

Once the simulation end time is reached, the longest blockchain is chosen centrally by comparing the local blockchains of all miners. Once the longest blockchain `lb` is found, the local blockchain of all miners is set to `lb`.

3. Simulation

BlockSim-Net is intended to be a simulator of a blockchain on a real network with different servers acting as miners. For implementation, we have split the structure of the simulator into two types of servers: 1) the *Admin Server*; and 2) the *Miner*. All miners on the network can deploy the miner code at their ends and run a simulation collectively. The admin server runs on a single node during the simulation.

3.1. The admin server (AS)

For our purposes, we have used an admin server (AS) that acts as a common point of contact to each of the miners. Every node connects to this server at the beginning of a simulation. AS, having received connections from the miners, collects information about them: miner ID, hash rate, IP, the port on which each of the miners host their server socket during the simulation. We call this information *miner-info*. Being the common point of contact for each of the miners, AS also shares the information of other miners in the network. Each node can communicate with each other for the purposes of the simulation. Although, in reality, such an admin server is not present in a blockchain, here we need it for the sake of efficiency and utilize it only at the start and end of the simulation.

We emphasize that AS only initiates the miners for the simulation and has no role in the creation and exchange of blocks in the simulation. AS assigns an ID to each of the miner nodes in order to make identification of the miner nodes easier. Once each of the nodes has been provided with miner-info, they are ready to run the simulation independently without any help from AS. The miner-info includes miner ID, the total hash rate of all the nodes in the network, and the IP and the port on which each of the miners will host their server socket during the simulation.

Apart from communicating information about other miners in the network with each node, AS is also responsible for the creation of the genesis block and creating a pool of transactions that are to be used by the miners for the duration of the simulation. As all miners have access to the genesis block and a transaction pool (with common transactions) in an actual network, the AS provides the genesis block and a pool of transactions to all the miners at the beginning of the simulation as a common starting point.

The next time that the AS is needed in the simulation is at the time of consensus, i.e., once the simulation is complete. While in a real blockchain, consensus happens as a result of communication between the miners themselves, here, it is important to remember that BlockSim-Net performs a timed simulation; a simulation has a start-time and an end-time. Due to this property of a simulator, there arises a need for an entity that can make up for a potential abrupt halt in the network. Thus, when the simulation end-time is reached, we use AS to ensure that the local blockchains of all miners are complete and uniform.

3.2. The Miner

The role of a miner is to create blocks on the network and propagate those to the other miners on the network. A miner also listens to the other miners on the network for blocks. Upon receiving blocks from other miners, a miner updates/maintains their blocks, based on some predefined rules. These rules differ from one another based on the type of blockchain or application.

Initially, each one of the miners connects to the admin server, AS. They learn miner-info from AS. As mentioned in the previous section, miner-info includes miner ID, the total hash rate of all the nodes in the network, and the IP and the port on which each of the miners hosts their server socket during the simulation. Once this basic information is acquired, the simulation timer starts running.

As soon as the simulation begins, AS creates a genesis block and a transaction pool and sends them to the miner nodes. Each miner node, upon receiving the genesis block and a pool of transactions, can start creating a new block at depth 1 (considering the genesis block is at depth 0). Each miner constantly listens for blocks from other miners on the network.

3.3. The overview of the simulation

The main steps of the simulation is shown in Figure. They can be summarized as follows:

1. Miner connects to the admin server.
2. Miner gets basic information from the admin server.
3. Miner starts the simulation.
4. Miner gets the Genesis Block and Transaction Pool from the admin server at the start of the simulation.
5. Miner starts the process of creating blocks and listening to blocks created by other miners on the network. With every new block (either created or received), Miner may update its local blockchain.
6. At the end of the simulation, Miner replaces the empty blocks (if any) in its longest chain with the blocks received.

7. Miner sends the last block of their longest chain to the admin server for consensus.

8. Based on the results of consensus, Miner gets the longest chain from the admin server (if its chain is not the longest).

In the rest of this section, We will describe them in detail.

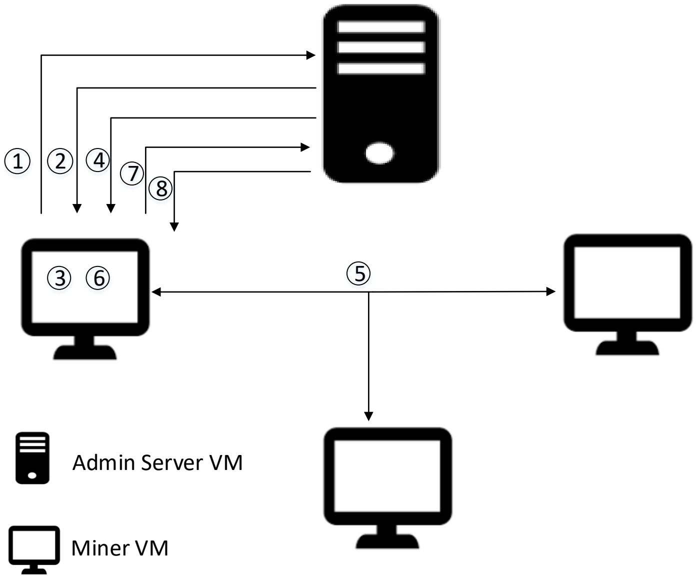

Figure. BlockSim-Net Simulation Flow. Descriptions of steps given inside circles are provided in Section 3.3.

3.4. Initialization

One of the main responsibilities of AS is to provide basic information to the miners in the network. This information includes the miner ID to uniquely identify a miner, IP addresses and ports of all the miners on the network so that they can communicate with each other, and the total hash rate of the network, which is computed by the admin server by adding the individual hash rates of each miner. This basic information is referred to as miner-info in our paper. Additionally, at the start of the simulation, AS also provides a pool of transactions to be used for a simulation period and the genesis block to the miners at the start of each simulation. On the miner end, each of the miners randomly initializes themselves with a hash rate (a random sample from 0 to 30). After receiving their IDs from the admin server, the miners also initialize themselves with that value.

At the start of the simulation, each miner gets a genesis block and a pool of transactions from AS. After this, they start mining on top of the genesis block. The created blocks are stored in a 'create queue'. The miners also listen for new blocks from other miners on the network. The received blocks are stored in a 'receive queue'.

In order to simulate proof of work in our blockchain network, we use the concept of "events" similar to [13]. In our implementation, each miner uses their (randomly generated) hash rate, h, in conjunction with the value of the total hash rate obtained from AS to calculate the time (in seconds) a miner would take to create a block. Thus, the creation of a block, in our context, means the creation of a block object that is scheduled

Algorithm 2 BlockSim-Net simulation algorithm for a miner.

```
procedure $\mathcal{F}$(txn_pool, genesis_block, node_info, sim_time)
    while current_time < sim_time do
        if (! empty(create_queue)) then
            block = getnextcreateblock()
            if current_time >= block.scheduled_time then
                update_local_blockchain(block,”create”)
            end if
        end if
        if (! empty(receive_queue)) then
            block = getnextreceiveblock()
            $\mathcal{F}_{update\_local\_blockchain}$(block,”recv”)
        end if
    end while
    build_full_chain()
    consensus()
end procedure
```

to be sent out into the network only when its blocktime (as calculated) is reached (or exceeded). As mentioned above, once created, these block objects are added to the ‘create queue’.

The miner iteratively checks for blocks in the ‘create queue’ and the ‘receive queue’. When a block is created, the miner appends it to its own blockchain if certain conditions are met (explained in the next section). On the other hand, if new blocks have been received, the miner checks if the blocks impose any further action to be taken on its blockchain. At the end of the simulation, the miner tries to build a full chain to account for any delays in the propagation of certain blocks. This has been explained in detail in Section 3.6. Finally, the miners send the last blocks in their local blockchains to the AS for consensus (explained in Section 3.7). This entire protocol has been illustrated in Algorithm 2.

3.5. Updating local blockchain

Any update to the local blockchain of a miner happens based on some predefined rules. A miner node may have to update its local blockchain in two cases: (a) when it creates a block, (b) when it receives a block from another miner node in the network. Algorithm 3 describes the protocol to update miners’ local blockchain. We now analyse the two cases:

1. When it creates a block: As discussed above, in this paper, ‘creation of a block’ actually corresponds to the scheduling of an event. When a block object is created, it is appended to the miner’s ‘create queue’. Any action on the created block is only taken once the block object’s blocktime has been reached or exceeded. As we have also discussed above, a miner needs to constantly check the blocks in its ‘create queue’ to see if the blocktime of the block (next_create_block) with the earliest timestamp in the queue has been reached (or exceeded). Once the blocktime of next_create_block has been reached (or exceeded), the miner looks at the last block (last_block) in its local blockchain. If the depth of last_block is less than that of next_create_block, the miner appends next_create_block to its local blockchain. After this process, next_create_block is removed from the queue. The miner can now start mining on top of next_create_block, which is now the last block of the updated blockchain. If a block with the same depth as next_create_block is received, it will be added to the miner’s uncle chain.

Algorithm 3 Local blockchain update procedure of BlockSim-Net.

```
procedure $\mathcal{F}_{update\_local\_blockchain}$(block, flag)
    if flag = ”create” then
        if ((myCH.last_block().depth < block.depth) and (myCH.last_block().id = block.previous)) then
            myCH.append(block)
            send_block(block)
            myCH.create_block_event()
        end if
    else
        if (block.previous = myCH.last_block().id) then // case 1: received block is built on my last block
            myCH.append(block)
            send_block(block)
            myCH.create_block_event()
        else// case 2: the received block is not built on my last block
            if (block.depth > myCh.depth) then
                myCH = construct_chain(block)
                myCH.create_block_event()
            else
                myUncleCH.append(block)
            end if
        end if
        receive_queue.remove(block)
    end if
end procedure
```

2. When it receives a block: When a miner receives a block from another miner in the network, the block is first added to the ‘receive queue’. During the simulation, the miner fetches the block (next_receive_block) with the earliest timestamp from the ‘receive queue’. If the depth of this block is greater than that of the last block in the miner’s local blockchain, the miner needs to update its blockchain. If not, next_receive_block is added to the uncle chain.

   If there is a need to update one’s local blockchain, the following rules have to be followed:

   - If a received block is built on top of the last block of the miner’s longest local blockchain, the miner appends it to its longest chain. If the miner was already trying to mine a block at the same depth as the received block, it will discard that block. The miner, having updated its local blockchain, can now start mining on top of the received block.

   - However, if the received block is not built on top of the last block of the miner’s longest chain, the miner looks at its depth. If the depth of the received block is less than or equal to that of the last block of the miner’s longest local blockchain, the miner adds the block to its uncle chain. However, if the depth of the received block is greater than that of the last block of the miner’s longest local blockchain, the miner needs to switch to the chain of the miner from whom the block was received.

In Algorithm 3, the procedure `update_local_blockchain` is described. This procedure receives a block and a flag associated with it as inputs. Here, `myCh` refers to a miner’s local blockchain, `block.depth` is the depth associated with a block, `myCH.last_block()` returns the last block in the miner’s local blockchain, `block.previous` refers to the block ID of the block that this block was mined on top of, `myUncleCH` refers to the uncle chain which contains the orphaned blocks of the miner. Lastly, `construct_chain` is a method that will allow the miner to efficiently switch to another miner's blockchain, without having to receive that miner's entire blockchain. The details of this method have been described in the next section.

3.6. Switching to another miner's blockchain

A simulation entails multiple exchanges of blocks. However, not all of the blocks involved (created or received) make it to the longest chain of a miner immediately. However, because of consensus, many of these blocks make it to the miner's longest chain at some point with a considerable probability. Thus, if some blocks are discarded because they do not make it to the miner's longest blockchain at this point, but they make it to the miner's longest blockchain at a point in the future, it does not make sense to discard them.

In our implementation, the miners locally store all the blocks that they create and receive from other miners on the network in a dictionary with the block IDs as keys and block objects as values. Thus, when a miner has to switch to another miner's chain, they trace back from the latest received block up to the genesis block. If the miner does not have some blocks in their local dictionary due to potential delays, we put temporary empty blocks up on their behalf. We then try to replace the empty blocks with the correct blocks, if available in the dictionary, once the simulation ends. This last step corresponds to the `build_full_chain` method given in Algorithm 2.

3.7. Consensus

At the end of the simulation, each miner sends the last block of their longest local blockchain to the admin server. The admin server determines the miner with the longest chain from the depth of these blocks. In case more than one miner has the longest blockchain, it considers the miner whose last block was created the earliest.

Once the admin server determines the miner with the longest chain, the admin server communicates with this miner to obtain its longest local chain. After that, the admin server sends it to all the other miners. If the miner with the longest chain has empty block objects that could not be replaced (due to the unavailability of those blocks) as part of its chain, that particular simulation is discarded.

This protocol has been described in Algorithm **??**. According to this algorithm, the admin server first determines the miner whose blockchain is the longest, using the method $\mathcal{F}_{determine\_accepted\_miner}$. It then requests this miner to send its chain and stores it in `global_chain`. Finally, `global_chain` is sent to all miners with the help of the method `send_chain()`. The admin server also prints the statistics of the consensus at the end.

4. Results

Our simulator codebase is available at: https://github.com/prashanthi-r/blocksim-net. Our implementation is in Python 3.6, and it uses the following packages: `NumPy` (version 1.20.3), `SciPy` (version 1.6.3), `math`, `random`, `pickle`, `threading`, and `socket`. We present the results of our simulation for both Bitcoin and Litecoin using the recent market shares.

For Bitcoin, we have deducted the mining statistics of the top 6 pools (F2Pool, Poolin, BTC.com, AntPool, Huobi.pool, and ViaBTC) for the time interval between 10 May 2020 and 10 May 2021 from btc.com[8]. We feed this information as the hashing powers for 7 miner nodes (6 pools + Others) for a 10,000-second simulation. The results have been presented in Table 2. Similarly, for Litecoin, we have deducted the mining statistics of 5 of its

[8] Website https://btc.com/stats/pools, [accessed 10 May 2021].

Algorithm 4 Consensus algorithm of BlockSim-Net executed by admin server

```
procedure $\mathcal{F}_{determine\_accepted\_miner}$(last_blocks)
    maxLength = -1 // length of the longest blockchain
    minerId = -1 // variable to store the ID of the miner with the longest chain
    for block in last_blocks do
        if (block.depth $\geq$ maxLength and block.timestamp $<$ current_time) then
            maxLength = block.depth
            timestamp = block.timestamp
            minerId = block.minerId
        end if
    end for
    return minerId
end procedure

procedure $\mathcal{F}_{consensus}$(last_blocks)
    minerId = $\mathcal{F}_{determine\_accepted\_miner}$(last_blocks)
    global_chain = get_chain(minerId)
    send_chain(global_chain)
    print_consensus_stats()
end procedure
```

pools (LTC.top, AntPool, F2Pool, Litecoinpool.org, and ViaBTC) from btc.com[9]. We feed this information as the hashing powers for 6 miner nodes (5 pools + Others) for a 10,800-second simulation. The results have been presented in Table 3. We have run both BlockSim and BlockSim-Net simulations, with the latter on different ports of an Intel Core i5 8th generation processor with 4 cores and 8GB RAM.

Table 2. Block percentage for major Bitcoin miners in a BlockSim-Net simulation. Expected block interval = 5 seconds; Simulation time = 10,000 seconds; Expected number of blocks for the simulation = 2000; Total number of blocks mined in BlockSim: 1559; Total number of blocks mined in BlockSim-Net: 1979.

Mining Pools and Hash Percentage	Real mining share(%)	% of total blocks mined	
		BlockSim	BlockSim-Net
F2Pool	16.7	15.26	17.58
Poolin	13.7	15.06	12.43
BTC.com	11.5	12.95	11.41
AntPool	11.1	11.54	11.67
Huobi.pool	8.7	7.63	8.74
Binance Pool	8.7	7.44	7.57
Others	29.6	30.06	30.52

In BlockSim-Net simulations, we observe that when the expected number of mined blocks in the simulation is higher, the deviation from the expected fair share (i.e., the hash power of each miner) is lower. Both Tables 2 and 3 demonstrate that every miner gets a fair share (total block percentage) that aligns with their hashing power. Note that Table 2 induces less deviation from the expected rewards compared to Table 3 because of a higher block interval. The expected block interval and simulation times affect the result as they affect the

[9] Website https://miningpools.com/litecoin/, [accessed 10 May 2021].

Table 3. Block percentage for major Litecoin miners in a BlockSim-Net simulation. Expected block interval = 12.42 seconds; Simulation time = 10800 seconds; Expected number of blocks for the simulation = 869; Total number of blocks mined in BlockSim: 801; Total number of blocks mined in BlockSim-Net: 833.

Mining Pools and Hash Percentage	Real mining share(%)	% of total blocks mined	
		BlockSim	BlockSim-Net
LTC.top	22.0	22.32	24.60
AntPool	20.0	20.07	21.24
F2Pool	19.0	21.07	16.44
Litecoinpool.org	14.0	13.84	13.20
ViaBTC	12.0	10.47	12.24
Others	13.0	12.09	12.12

expected number of blocks found in a simulation. As a result, for future experiments, we recommend a small expected block interval (typically 5 seconds) for fast simulation. We also recommend a simulation with at least 2,000 expected blocks for low deviation.

Comparison with BlockSim. To compare with BlockSim simulations, our results give similarly close rewards to hashing powers of miners. To quantify closeness, we use a standard deviation type of metric *reward deviation* or $\mathcal{R}_{dev}$ that we define as follows:

$$\mathcal{R}_{dev} = \sum_{i=1}^{N} (\alpha_i - \mathcal{R}_i^{sim})^2$$

where N is the total number of miners in the network, α_i is the hashing power of the i-th miner, and $\mathcal{R}_i^{sim}$ is the reward share that the i-th miner receives in a simulation. The lower this value, the more consistent will the reward be to the hashing power. By this metric, we calculate that for the Bitcoin simulation in Table 2, BlockSim-Net results in $\mathcal{R}_{dev} = 4.85$, and BlockSim results in $\mathcal{R}_{dev} = 9.16$, and for the Litecoin simulation in Table 3, BlockSim-Net results in $\mathcal{R}_{dev} = 13.29$, while BlockSim results in $\mathcal{R}_{dev} = 7.59$. Since $\mathcal{R}_{dev}$ for BlockSim and BlockSim-Net are of the same order in the two simulations, we infer that one can expect BlockSim and BlockSim-Net to give similar rewards for miners with given hashing powers.

Additionally, we define a metric *block ratio* or λ, which is the ratio of the number of blocks mined in a simulation to the expected number of blocks for the specified simulation time and block interval. This metric is a measure of the closeness of the mining process in a simulation to a *real* blockchain. Note that the higher the block ratio, the closer a simulation is to the real mining process. For the Bitcoin simulation, BlockSim produces $\lambda = 77.95$, while BlockSim-Net produces $\lambda = 98.95$. Further, for the Litecoin simulation, BlockSim produces $\lambda = 92.06$, while BlockSim-Net produces $\lambda = 95.74$. One can observe that BlockSim-Net captures the essence of real blockchain mining better than BlockSim in both simulations. We encapsulate the comparison between BlockSim and BlockSim-Net for both simulations in Table 4 based on the defined metrics: reward deviation and block ratio.

5. Conclusion

In this paper, we propose BlockSim-Net, a simulation framework for blockchain technologies on a real network with multiple communication optimizations. Our codebase in Python, as an extension of the implementation of BlockSim, has been divided into two aspects: the admin server and the miner. Future work can focus on

Table 4. Comparison of the performance of BlockSim and BlockSim-Net based on reward deviation ($\mathcal{R}_{dev}$) and block ratio (λ) in the simulations in Tables 2 and 3.

Metrics	$\mathcal{R}_{dev}$		λ	
	BlockSim	BlockSim-Net	BlockSim	BlockSim-Net
Simulation in Table 2	9.16	4.85	77.95	98.95
Simulation in Table 3	7.59	13.29	92.06	95.74

improving the scalability of the system. Furthermore, while BlockSim-Net provides a codebase for proof-of-work blockchain networks, our work could be extended to other blockchain systems and simulate possible attacks and defence mechanisms on these systems.

Acknowledgment

We acknowledge support from TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 119E088.

References

[1] Etemad M, Küpçü A. Efficient Key Authentication Service for Secure End-to-end Communications. ProvSec 2015. doi: 10.1007/978-3-319-26059-4_10

[2] Allessie D, Sobolewski M, Vaccari L. Blockchain for digital government. Publications Office of the European Union 2019. doi: 10.13140/RG.2.2.34874.85449

[3] Singh P K, Singh R, Nandi S K, Nandi S. Managing Smart Home Appliances with Proof of Authority and Blockchain. Innovations for Community Services 2019; (pp.221-232). doi: 10.1007/978-3-030-22482-0_16

[4] Mashamba-Thompson T P, Crayton E D. Blockchain and Artificial Intelligence Technology for Novel Coronavirus Disease 2019 Self-Testing. Diagnostics 2020; 10 (198). doi: 10.3390/diagnostics10040198

[5] Garay J A, Kiayias A, Leonardos N. The Bitcoin Backbone Protocol: Analysis and Applications. EUROCRYPT 2015. doi: 10.1007/978-3-662-46803-6_10

[6] Eyal I, Sirer E G. Majority is Not Enough: Bitcoin Mining is Vulnerable. Commun. ACM 2013; 61 (7). doi: 10.1145/3212998

[7] Sapirshtein A, Sompolinsky Y, Zohar A. Optimal Selfish Mining Strategies in Bitcoin. Financial Cryptography 2017. doi: 10.1007/978-3-662-54970-4_30

[8] Nayak K, Kumar S, Miller A, Shi E. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. IEEE EuroS&P 2016. doi: 10.1109/EuroSP.2016.32

[9] Heilman E. One Weird Trick to Stop Selfish Miners: Fresh Bitcoins, A Solution for the Honest Miner. Financial Cryptography 2014 (Poster Abstract). doi: 10.1007/978-3-662-44774-1_12

[10] Zhang R, Preneel B. Publish or Perish: A Backward-Compatible Defense Against Selfish Mining in Bitcoin. CT-RSA 2017. doi: 10.1007/978-3-319-52153-4_16

[11] Stoykov L, Zhang K, Jacobsen H A. VIBES: fast blockchain simulations for large-scale peer-to-peer networks. ACM/IFIP/USENIX Middleware Conference: Posters and Demos 2017. doi: 10.1145/3155016.3155020

[12] Aoki Y, Otsuki K, Kaneko T, Banno R, Shudo, K. SimBlock: A Blockchain Network Simulator. IEEE INFOCOM 2019. doi: 10.1109/INFCOMW.2019.8845253

[13] Alharby M, van Moorsel A. BlockSim: An Extensible Simulation Tool for Blockchain Systems. Frontiers in Blockchain 2020; 3. doi: 10.3389/fbloc.2020.00028

[14] Gervais A, Karame G O, Wüst K, Glykantzis V, Ritzdorf H, Capkun S. On the Security and Performance of Proof of Work Blockchains. ACM Conference on Computer and Communications Security 2016. doi: 10.1145/2976749.2978341

[15] Wang T, Liew S C, Zhang S. When blockchain meets AI: Optimal mining strategy achieved by machine learning. Int. J. Intell. Syst. 2021. doi: 10.1002/int.22375

[16] Bissias G, Levine B. Bobtail: Improved Blockchain Security with Low-Variance Mining. The Network and Distributed System Security Symposium 2020.

[17] Sompolinsky Y, Zohar A. Secure High-Rate Transaction Processing in Bitcoin. Financial Cryptography 2015. doi: 10.1007/978-3-662-47854-7_32