Demo: Proof-of-Work Network Simulator for Blockchain and Cryptocurrency Research

Simeon Wuthier
University of Colorado, Colorado Springs
swuthier@uccs.edu

Sang-Yoon Chang
University of Colorado, Colorado Springs
schang2@uccs.edu

***Abstract*—Blockchain and the proof-of-work (PoW) distributed consensus protocol rely on peer-to-peer (P2P) networking. We build a PoW P2P simulator for the modeling and analyses of permissionless blockchain networking. Our simulator utilizes a built-in randomness generator for the simulations, has an easy-to-use interface and intuitive visualization, supports dynamic/programmable control and modifications, and can generate simulation data for further processing. We publish our simulator in open source to facilitate its use for blockchain and P2P networking research and especially recommend it for scalability research or preliminary testing. To highlight its features and capabilities, we demonstrate the simulator use in this paper to analyze the recent blockchain security research, including 51% attack, eclipse, partitioning, and DoS attack.**

I. Introduction

Blockchain technologies are built upon peer-to-peer (P2P) networking and provide a reliable, decentralized, and trustless system for blockchain applications. Within such P2P systems, nodes behave as both the client and the server in information delivery and form a distributed and non-hierarchical network. Having multiple, diverse, and locally selected peer connections helps with the security and reliability of blockchain networking. The P2P connections in blockchain are critical because they deliver information responsible for driving the blockchain consensus protocol and the ledger synchronization.

Proof of work (PoW) is the most popular mechanism for blockchain distributed consensus protocol. PoW achieves consensus without relying on the identity-based trust because cryptocurrencies require permissionless and anonymous applications, prohibiting other identity- and voting-based mechanisms for consensus protocols, e.g. practical byzantine fault tolerance (PBFT). Because PoW is based on race and competition, the timely networking and block delivery are critical.

Our simulator models and analyzes P2P networking, PoW consensus protocol, and the interrelations between the two. It makes use of a random number generator—with the option of choosing between SHA-256 hash brute-forcing and JavaScript-built-in random function—for mining the PoW blocks and generating the P2P networking packet occurrences/transmissions. We design and build our simulator based on our knowledge and previous hands-on experience with Bitcoin, the most popular cryptocurrency. We build flexible and dynamic control of the relevant parameters to enable such analyses, provide an easy-to-use interface, and the capability to generate and log the data for further analyses. Our simulator is designed to facilitate and aid in blockchain networking R&D as well as conduct the preliminary analyses for new permissionless blockchains/cryptocurrencies.

II. Technical Contributions of the Simulator

We model, design, and build our simulator based on our hands-on experience with a Bitcoin prototype (Bitcoin Core) capable of mining and networking in the Mainnet [1], [2]. The layout of our simulator is comprised of two parts: P2P communication/networking and the PoW protocol. While PoW is dependent upon P2P, the networking portion of our simulator was built to accommodate a wider range of applications. For example, PoW assumes all connections are bi-directional in nature while our simulator allows control over the direction of traffic. This section describes such implementation-based approaches used to model P2P and PoW scenarios.

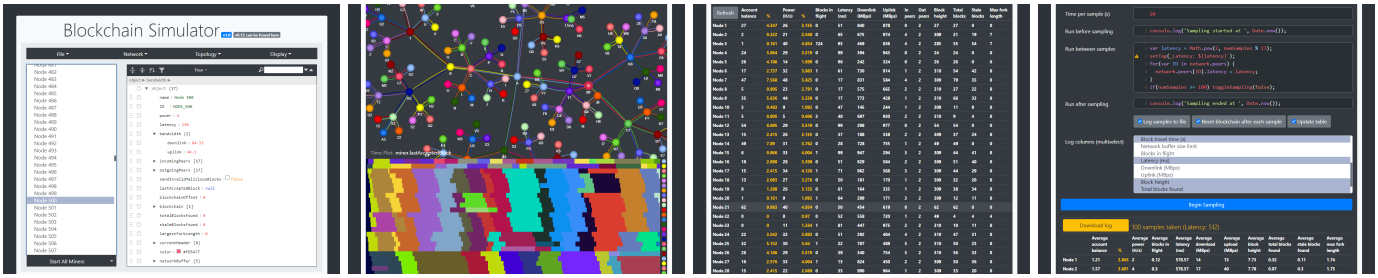
A. Software Implementations

Figure 1 displays the screenshots of our simulator interface. The software is implemented as a JavaScript web application, consisting of four main classes: the *Network*, *Miner*/node, *NetworkBuffer*, and the *Block/Header*. The *Network* class encapsulates all other class instances and is responsible for maintaining the networking-level parameters of P2P. The *Miner*/node class maintains the consensus-level parameters of PoW, as well as a *NetworkBuffer* to act as the router connecting the nodes. Blockchain ledgers are stored within the *Miner* and are implemented as an array/linked list of block headers. Nodes are built with functions to mine, broadcast information, send/receive block headers, and maintain the block-linking structure of the blockchain.

We enable the saving and loading of the *Network* class by serializing it to the user's file system in the local hard drive, the browser's built-in storage (*LocalStorage*), or to the clipboard via copy-and-paste. To accommodate larger networks, the serialization process uses LZ-based UTF-16 string compression which reduces the storage size. For example, it saves 87.8% of storage from 9.9MB to 1.2MB for a network of 10,000 nodes.

B. Modeling PoW

Due to the permissionless requirement of cryptocurrencies prohibiting identity-based or voting-based distributed consensus protocol, cryptocurrencies use proof of work (PoW). Bitcoin [3], which is the first and currently the most popular cryptocurrency invented the use of random PoW for blockchain synchronization and consensus protocol. Within this protocol,



(a) Node management screen (b) Network visualization screen (c) Node statistics screen (d) Sampling/logging screen

Fig. 1: A screenshot of the simulator, split into its four components. Figure 1a controls and manages the nodes and their parameters. Figure 1b visualizes the network using the *vis.js* framework, followed by a canvas that plots any portion of the simulator over time (with a textbox defining which object/portion). In this screenshot, the canvas is visualizing the blockchain synchronization, where each node is given a row, and the colors represent that node's last accepted block, i.e. what block it is currently mining from. Figure 1c visualizes the node data in numerical format, and Figure 1d provides the sampling interface of the network, as discussed in Section III-2.

nodes connect to one another and relay block/header messages containing parameters necessary for PoW consensus [4]. The process of mining a block in PoW consists of repeatedly guessing a unique nonce value within the block header and double hashing the header using SHA-256 until the hash is within a numerical lower bound/difficulty threshold. Blocks are found/mined randomly, and the average duration between blocks is ten minutes by the Bitcoin consensus protocol design and, more specifically, by the adaptive mining difficulty control. To capture the hash computation power/rate capabilities, we use the built-in JavaScript function SETINTERVAL as an artificial rate limiter to handle mining for the active miners. The asynchronous nature of this function allows multiple instances/miners to mine simultaneously. Because miners work independently, it becomes possible that two miners find blocks of the same height, thus creating a fork and the need to prioritize one over the other. The blocks that are not accepted are *stale blocks* while the blocks that become part of the chain are *accepted blocks*. Since the hash-function-based randomness generator has a one-to-one correspondence with the nonce data in a block, by default, our simulator verifies the validity of a block by comparing the nonce to the lower-bound block difficulty, rather than hashing. This is done for performance optimization; however, we also include the Boolean property "*useSHA256*" to activate the traditional block verification and random number generator used in Bitcoin.

Simulator Parameters We enable control over the following PoW-based parameters: network difficulty, network reward for accepted blocks, mining hash rate per node, toggling of the mining thread per node, as well as enabling a node to maliciously send invalid blocks (see Section IV-3 for more information). The PoW parameters that we measure include: block height, stale blocks mined, miner balances/accepted blocks mined, and the number of stale blocks in a fork.

C. Modeling P2P Networking

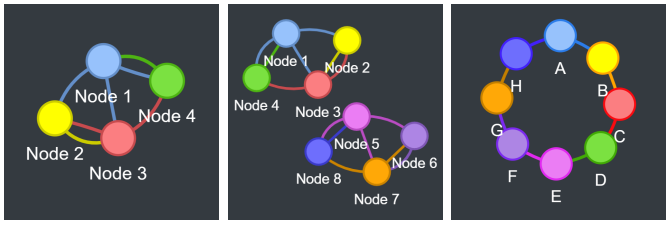
Blockchain technologies are dependent upon the underlying network infrastructure that ensure blocks are transmitted and distributed to all nodes in the network. Every node maintains a list of outgoing and incoming connections which enable the modeling of network topologies, and flow of network packets. We identify each node in the network with a unique identifier/address, along with a *NetworkBuffer* object that models the routing of blocks between entities. This object uses JavaScript's built-in SETTIMEOUT asynchronous function to induce an artificial delay/latency on every networking packet. We give structure to the packet transmission by holding a single queue/buffer within each node's *NetworkBuffer* object which limits the upper-bound rate at which packets can be sent and received. Additionally, this also requires defining an artificial packet size for the speed by which a packet can travel through the network buffer. To limit this, we define *bufferSizeLimit* that prevents a node's queue/buffer from exceeding a threshold size. Optionally, setting *bufferSizeLimit* to JavaScript's global INFINITY property enables an unlimited network buffer.

Simulator Parameters We provide direct control over the following P2P-based parameters: network incoming and outgoing connections/topology, transmission bandwidth/uplink and downlink rate (controlled by block size), link latency, and buffer/queue size limit. The P2P-based parameters that we measure include the number of packet hops, the duration for a packet to reach its destination, the current size of the buffer/queue, and the list of blocks in-flight for each node.

III. SIMULATOR FEATURES

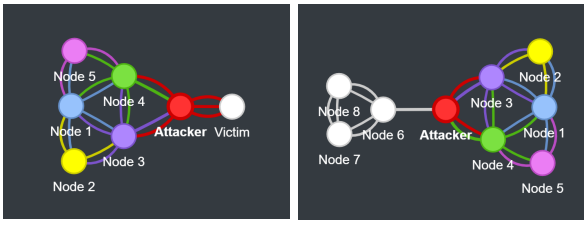
We aim to provide a vibrant set of functionalities that allow the studying of various P2P and PoW-based components. In this section, we highlight the notable capabilities including the P2P networking topologies and control, as well as the dynamic logging of the simulator for analytics and visualization.

1) P2P Networking Topologies and Parameter Control: To simulate distributed P2P networking, we enable control of the networking topology in each pairwise connection. For example, Figure 2a shows a very simple four-node scenario where the number of outgoing connections for Node 1, Node 2, Node 3, and Node 4 are 3, 1, 2, and 1, respectively. Figure 2b duplicates this topology to show how we can



(a) Network of four nodes (b) Disconnected network (c) Ring of eight nodes

Fig. 2: Example of networking topologies



(a) Modeling the eclipse attack (b) Modeling the partitioning attack

Fig. 3: Networking topologies of Eclipse and Partitioning

simulate disconnected networks, where Nodes 1 through 4 share a ledger, while Nodes 5 through 8 share a different ledger. Furthermore, we also enable the building of layouts from known P2P networking topologies [5] such as Figure 2c where eight nodes are arranged in a Ring topology, thus enforcing a single path from one node to another. The closest to blockchain-based cryptocurrencies is the Mesh topology.

The simulator can also control the delay/latency and bandwidth between any two peer nodes. For example, we test our simulator to model Bitcoin networking by importing the network size (number of peer nodes) and the channel latency between the peer nodes from the Bitnodes dataset, a regularly updated dataset measuring the Bitcoin P2P networking [6]. We can model and sample the channel latencies as random variables using the empirical distribution given from the Bitnodes dataset and compare the peer-connection statistics of the simulator to that of our Bitcoin implementation based on Bitcoin Core code and connected to Mainnet. In this preliminary Bitcoin modeling, we find that the number of connections for the simulator varies between 10 and 36 while the number of connections for a public Bitcoin Core (version 0.21.0) node connected to the Mainnet is 32 on average; more comparisons and details are omitted due to space constraint.

2) Dynamic Logging and Analytics Visualization: We also add the ability to modify any portion of the network in real-time, as defined by our logging and sample generation portion of the simulator. As shown in Figure 1d, the interface has an option to begin sampling, with parameters for the time per sample (s), code to execute before sampling, between samples, and after samples, whether or not to reset the blockchain between samples, generate a downloadable file to export the simulator data into a CSV file, and a list of columns to generate in the CSV log file. If the option is selected to reset the blockchain after each sample, then all blocks that are currently in-flight are stopped, a snapshot is taken of the current blockchain state, and each node's blockchain is reset back to the genesis block.

Granting the ability to execute user-defined JavaScript before, between, and after samples enables dynamically and conditionally setting node/simulation parameters. For example, to increment the computing power of a node with ID: X, adding "X.POWER += 1" to execute between each sample will increment the node's computing power every sample. More complex statements can be used to provide finer control.

IV. Using the Simulator for Analyses

In this section, we explore the impact of decentralized networking by modeling the 51% attack, eclipse attack, partitioning attack, DoS attack, and Bitcoin topology. While this is by no means comprehensive, we aim to give a starting point for further modeling and analyses.

1) 51% Attack: When a node's computational power becomes greater than the rest of the network's combined power, the node gains full control of the distributed ledger. We build on the networking topology from Figure 2a by adding an Attacker node connected to Node 1. The attacker node ignores the other node's blocks to create an intentional fork and reverse the other chain/blocks by Nodes 1 through 4.

Simulation Experiment and Results We set the computing power of the four nodes to 25 hashes per second each, and the Attacker node to 101 hashes per second (50.25% of the network's power). We measure the mining balance (number of accepted blocks) per node, excluding the attacker's balance which monotonically increases. Because the attacker has greater computational power, every time the attacker's block height exceeds the rest of the network, the network switches to the attacker's blockchain to continue mining in a new fork, however, none of the mined blocks are permanently accepted unless the attacker decides to accept them. Since we simulate the most simple scenario where the attacker ignores the network, this results in the rest of the network mining blocks at a consistent rate, but all blocks resetting repeatedly, as shown in Figure 4.

2) Eclipse and Partitioning Attack: The eclipse attack occurs when a malicious node is able to tamper with another node's list of potential connections and replace the identities with the attacker's own spoofed identities [7]. While the victim node can be oblivious to this behavior, the attacker will have control over the information that is sent and received by the victim. Similarly, the partitioning attack occurs when a connected network becomes disconnected through the strategic intervening of connections that are necessary for the transmission of messages between two partitions of a network [8].

While the two attacks are conducted differently, the outcomes of both the eclipse and partitioning attack result in the control of a node's blockchain, which we analyze.

Simulation Experiment and Results We simulate the eclipse attack (Figure 3a), and partitioning attack (Figure 3b) by using the Attacker node to cause intentional forks. We set all nodes to have equal resources. In the partitioning attack, the attacker

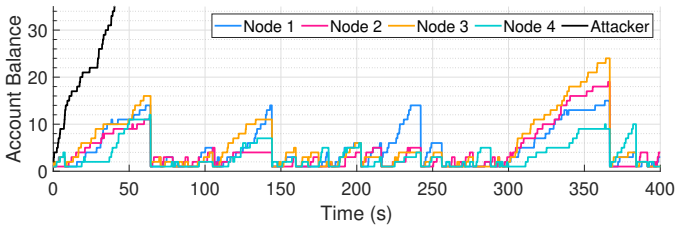


Fig. 4: Block reward balance vs. time. The 51% attacker dominates the blocks in the chain (yielding increasing block reward for the attacker) so that the rest of the nodes' blocks get revoked (the reward balance decreasing to zero). The attacker's balance continues to increase and becomes 418 at 400 seconds.

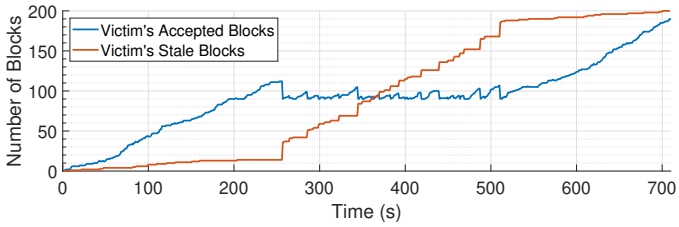


Fig. 5: The victim's number of accepted and rejected blocks during the DoS attack. While the victim continues to find blocks (same slope), they get replaced due to network congestion and the inability to broadcast newly found blocks.

targets Node 6, Node 7, and Node 8. We begin by letting the network's block height reach 100. In our experiment, 11 of these blocks were mined by Node 6. We then use the Attacker node to disconnect the partition of victim nodes and let the block height reach 200 on the victim's partition. At this point, Node 6 has mined 46 blocks. However, the larger partition (Nodes 1 through 5) generated a block height of 234, so when the attacker re-connects the network, all block data generated between height 100 to 200 become stale on the smaller partition, and the balance of Node 6 falls back to 11.

3) DoS Attack and Congestion: In cases where there is more networking than what is supported by the channel link capacity, for example, due to a DoS attacker flooding the link, the node's networking availability will get disrupted. While this may occur non-maliciously, the denial of service (DoS) attack takes advantage of this by flooding unnecessary information that wastes computing power, networking bandwidth, memory, or other system resources.

Simulation Experiment and Results We model this attack by conditionally enabling and disabling an attacker node using the dynamic code execution feature discussed in Section III-2, and the network topology in Figure 2a, with all nodes having a bandwidth of 100 MBps and a buffer size of 1000 MB. When this buffer size threshold is exceeded, the incoming and outgoing packets are dropped due to memory overflow.

The DoS attacker targets Node 1. We begin by making the attacker passive and, at 200 seconds, we activate the attacker to flood invalid packets to the victim until 500 seconds, before becoming passive again. Figure 5 shows how the attack disables the victim's ability to broadcast blocks to the network. We show that the rate of stale blocks increases while the rate of accepted blocks remains unchanged. The victim continues to mine valid blocks during the attack, however, due to the networking limitations from the attack, these blocks become overwritten by Nodes 2 through 4 (not under the DoS attack) before Node 1 has time to broadcast them to the other nodes and claim the rewards. Node 1 therefore loses the forking race and any recent blocks it mined during the DoS attack at 257 seconds become stale/invalid due to the longest-chain rule.

V. Conference Demo

Because the simulator is software-based and since ICDCS is held online this year, we do not need additional equipment for the conference accommodations and presentation. In fact, the simulator is currently available[1] with the source code[2], which we plan to continue to maintain and update. At the conference, we plan to provide a dynamic demo presentation of our simulator and its potential use for distributed computing/networking research. The demo presentation will facilitate the dissemination and the use of the simulator and help gather feedback for improving the simulator for the next version.

VI. Conclusion

Our simulator presents a wide range of capabilities and features to enable research in networking and mining in cryptocurrency. We use the simulator to analyze concrete scenarios to highlight the features and capabilities. Even though our simulator has not been advertised before this paper, it is already garnering interests among researchers to show its potential for blockchain research. We are in active communications with other researchers, including those using our simulator for a thesis dissertation research, which inspired us to write this paper and share our simulator.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1922410.

References

[1] J. Kim, M. Nakashima, W. Fan, S. Wuthier, X. Zhou, I. Kim, and S.-Y. Chang, "Anomaly detection based on traffic monitoring for secure blockchain networking," in *International Conference on Blockchain*, 2021.

[2] S.-Y. Chang and S. Wuthier, "Dynamic power control for rational cryptocurrency mining," in *Proceedings of the 3rd Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, 2020, pp. 47–52.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[4] "Block headers." [Online]. Available: https://developer.bitcoin.org/reference/block_chain.html

[5] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of IEEE INFOCOM'96. Conference on Computer Communications*, vol. 2. IEEE, 1996, pp. 594–602.

[6] A. Yeow, "Bitnodes," 2021. [Online]. Available: https://github.com/ayeowch/bitnodes

[7] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 129–144.

[8] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 375–392.

[1] https://simewu.github.io/blockchain-simulator

[2] https://github.com/simewu/blockchain-simulator