CBlockSim: A Modular High-Performance Blockchain Simulator

Xuyang Ma*, Han Wu†, Du Xu*, Katinka Wolter‡,

*School of Information and Communication Engineering, University of Electronic Science and Technology of China
Chengdu, China
xuyangm7@gmail.com, xudu@uestc.edu.cn
†School of Computing, Newcastle University, Newcastle Upon Tyne, United Kingdom
han.wu@newcastle.ac.uk
‡Department of Mathematics and Computer Science, Freie Universität Berlin, Berlin, Germany
katinka.wolter@fu-berlin.de

Abstract—**Blockchain has attracted much attention from both academia and industry since emerging in 2008. Due to the inconvenience of the deployment of large-scale blockchains, blockchain simulators are used to facilitate blockchain design and implementation. We evaluate state-of-the-art simulators applied to both Bitcoin and Ethereum and find that they suffer from low performance and scalability which are significant limitations. To build a more general and faster blockchain simulator, we extend an existing blockchain simulator, i.e. BlockSim. We add a network module integrated with a network topology generation algorithm and a block propagation algorithm to generate a realistic blockchain network and simulate the block propagation efficiently. We design a binary transaction pool structure and migrate BlockSim from Python to C++ so that bitwise operations can be used to accelerate the simulation and reduce memory usage. Moreover, we modularize the simulator based on five primary blockchain processes. Significant blockchain elements including consensus protocols (PoW and PoS), information propagation algorithms (Gossip) and finalization rules (Longest rule and GHOST rule) are implemented in individual modules and can be combined flexibly to simulate different types of blockchains. Experiments demonstrate that the new simulator reduces the simulation time by an order of magnitude and improves scalability, enabling us to simulate more than ten thousand nodes, roughly the size of the Bitcoin and Ethereum networks. Two typical use cases are proposed to investigate network-related issues which are not covered by most other simulators.**

Index Terms—**blockchain, simulator, network**

I. Introduction

Bitcoin [1] started in 2008 and was the first successful system using a blockchain as core technology. Bitcoin, like several other similar systems, has an underlying peer-to-peer network composed of many connected nodes. Each node holds a copy of the ledger, which is an ordered sequence of blocks. The nodes periodically reach consensus on the next valid block using a consensus protocol. Hash pointers connecting the blocks guarantee immutability of the block contents, i.e. the transactions. While transactions in Bitcoin are rather simple, Ethereum [2] supports smart contracts, Turing complete programs, as transactions.

With many attractive features including tamper resistance, high security, decentralization and smart contracts, blockchain systems have received a lot of attention by academia and industry. We found that blockchain technology has been widely used in many different fields such as finance [3], [4], internet of things (IoT) [5], [6], the energy market [7], [8], and healthcare [9], [10]. While researchers have a huge interest in blockchain, popular blockchains like Bitcoin and Ethereum do not support flexible modification and configuration. Therefore, many blockchain simulators, from early Bitcoin Simulator [11] to recent BlockSim [12] and SimBlock [13], have been proposed to simulate blockchain.

We have evaluated state-of-the-art blockchain simulators and noticed their limited performance and scalability. Some simulators such as Bitcoin Simulator and SimBlock do not explicitly include transactions in order to accelerate the simulation. But some research requires simulation for transactions to estimate the run time of smart contract [14] or the throughput of blockchain [15]. Other simulators such as BlockSim:Faria [22] and BlockSim:Alharby [12] aim at including much more detail but suffer from high run time and low scalability. Therefore recent research experimented in different ways rather than using existing blockchain simulators. Some work [14], [15] deployed a small-scale blockchain on a few computers or servers which is very different from the real blockchain environment. Other work [16], [17] ran experiments on Ethereum testnets (Rinkeby or Ropsten) where the configuration is unknown and blockchain settings can not be adjusted. The work in [18], [19] did not apply experiments in a real blockchain but instead used theoretical analysis.

Contributions: To maintain a detailed simulation and achieve high performance and scalability, we extend Block-Sim:Alharby to be a flexible high-performance blockchain simulator named CBlockSim. Our contributions are as follows.

1) We add a network module integrated with a network generation algorithm and a Dijkstra-based algorithm to generate a realistic blockchain network and simulate information propagation efficiently.

2) We rebuild BlockSim:Alharby in C++ and design a binary transaction pool data structure so that we can adopt bitwise operations in C++ to accelerate the simulation. The experiments compare CBlockSim with other state-of-the-art simulators, indicating that CBlockSim shortens the run time by an order of magnitude and increases scalability by an order of magnitude.
3) We modularize the simulator based on five blockchain processes. The simulator can be configured to represent different blockchains, i.e. different consensus protocols, different information propagation algorithms and finalization rules implemented in individual modules.
4) We propose two typical use cases towards network-related issues that most other simulators do not cover due to the lack of precise network simulation.

II. RELATED WORK

Since Bitcoin and Ethereum are the two most popular blockchains, most blockchain simulators aim to simulate either or both of them. Bitcoin Simulator [11] built in C++ and ns3 are the most widely used simulators for Bitcoin. VIBES [20] is another large-scale Bitcoin simulator built in Scala. An Ethereum simulator eVIBES [21] was proposed inspired by VIBES. SimBlock [13] is a simulator for Bitcoin built in Java, supporting compact block relay. It is also expected to support simulation for Ethereum in the future. BlockSim [22] proposed by Faria and Correia aims to simulate both Bitcoin and Ethereum with high flexibility. Another simulator also named BlockSim [12] proposed by Alharby and van Moorsel is built in Python and can simulate both blockchains as well.

All these simulators are analyzed in [23]. Some simulators cannot simulate several of the blockchain components. VIBES, eVIBES and BlockSim:Alharby adopt a fixed information propagation delay so that they can not simulate realistic networks. Bitcoin Simulator and SimBlock ignore simulation of transactions but concentrate on blocks. Some simulators have limited performance. BlockSim: Faria is only used to simulate a small-scale blockchain. BlockSim:Alharby suffers from high run time when simulating transactions. Our proposed CBlockSim aims at performance and scalability and simulates a significant number of blockchain components.

III. CBLOCKSIM ARCHITECTURE

As illustrated in Figure 1, CBlockSim is built on a discrete-event simulation model. A queue $eventPool$ is adopted to manage two types of events, GENERATE_BLOCK event and RECEIVE_BLOCK event. The global timestamp $clock$ which is no more than SIM_TIME (the simulation time set by the user) updates with the change of the timestamp of each event. Process A deals with GENERATE_BLOCK event and process B deals with RECEIVE_BLOCK event. We modularize CBlockSim based on the following five blockchain processes [24].

A. Block Proposal

Block Proposal (step A3 in Figure 1) is responsible for block generation according to different consensus protocols.
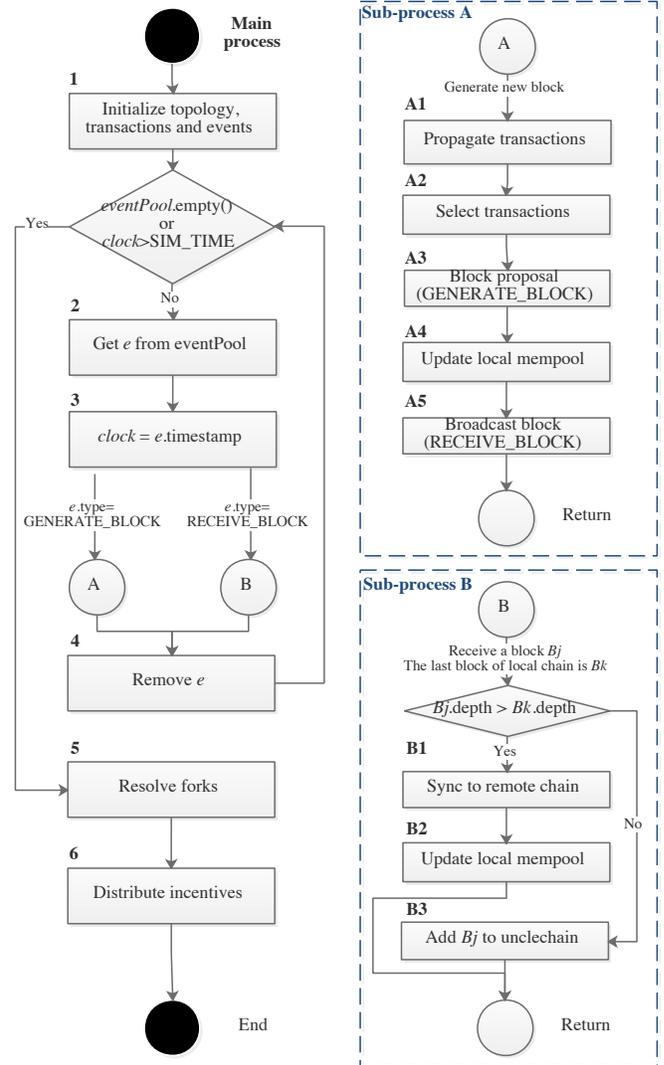

Fig. 1. The flowchart of CBlockSim

It is an individual module for simulating the generation time of every generated block. We implement an algorithm to simulate two mainstream consensus protocols, PoW and PoS. The PoW simulation algorithm is consistent with the algorithm in BlockSim:Alharby. Suppose there are N miners where $\mathbf{N}=\{1, 2, \cdots, N\}$. The hash power of a miner i is denoted by H_i. The fraction of blocks generated by miner i should be proportional to $\frac{H_i}{\sum_{j \in \mathbf{N}} H_j}$. Therefore the exponential distribution is adopted to simulate the time between blocks as shown in PoW function in Algorithm 1.

PoW is the most common consensus protocol of many blockchains but it is often accused of consuming too much energy. PoS is a promising energy-saving alternative consensus protocol. Miners compete with each other based on their holding stakes instead of computing power. We simulate the PoS protocol as in PeerCoin [25] which considers both the stake owned by the miner and the duration that the miner has held on to it. Suppose the stake owned by miner i is S_i and the

duration that the miner has held the stake is denoted by Age_i. Then the fraction of blocks that miner i generates should be proportional to $\frac{S_i \cdot Age_i}{\sum_{j \in \mathbf{N}} S_j \cdot Age_j}$. The PoS function in Algorithm 1 presents the simulation of this process.

Algorithm 1 Block Proposal

    $BlockInterval$: the time interval between two blocks;
1: **function** POW(H_i, $currentTimestamp$)
2:     $\lambda \leftarrow \frac{H_i}{\sum_{j \in \mathbf{N}} H_j} \cdot \frac{1}{BlockInterval}$;
3:     The time took to mine a block: $T \sim Exp(\lambda)$;
4:     The generation time of the next block: $genTimestamp \leftarrow currentTimestamp + T$;
5:     **return** $genTimestamp$;
6: **end function**

7: **function** POS(S_i, Age_i, $currentTimestamp$)
8:     $\lambda \leftarrow \frac{S_i \cdot Age_i}{\sum_{j \in \mathbf{N}} S_j \cdot Age_j} \cdot \frac{1}{BlockInterval}$;
9:     The time took to mine a block: $T \sim Exp(\lambda)$;
10:     The generation time of the next block: $genTimestamp \leftarrow currentTimestamp + T$;
11:     **return** $genTimestamp$;
12: **end function**

B. Information Propagation

While the underlying network is a key part of a blockchain and could seriously affect the performance of the blockchain [26], [27], in many simulators there is no realistic network model. BlockSim:Alharby adopted the exponential distribution to simulate the block propagation delay. Instead, we add a network module to simulate information propagation in detail (step 2 in Figure 1). It first generates a realistic blockchain network topology and then calculates the block propagation delay.

The Bitcoin network is an unstructured peer-to-peer network which can be simulated using a random graph model [28]. The Ethereum network is a structured peer-to-peer network and has been proven to have the small-world property [29]–[31]. It can be simulated using the Watts-Strogatz model [32]. We adopt the method proposed in [33] to generate a random network and a small-world network. The average degree d and a topology controlling parameter β are used to control the generation of different networks. We set d to 12 [34] and β to 1 to generate a random graph (Bitcoin), and set d to 19.747 and β to 0.24 to generate a small-world network (Ethereum) as observed in [31]. As presented in Algorithm 2, the $GetProbability$ function calculates the probability of a connection between node i and node j. The $GenerateNetwork$ function compares the generated random number and the probability to determine whether two nodes are connected or not.

We propose an algorithm based on the Dijkstra algorithm to calculate the block propagation delay as shown in Algorithm 3. Step A5 in Figure 1 broadcasts the generated block according to the calculated block propagation delay. The difference between the calculation of the delay of Bitcoin and Ethereum is due to the difference of transmission protocols. (a)(b)(c)

Algorithm 2 Network Generation

    N: the number of nodes;
    β: topology controlling parameter;
    d: average number of connections;
    G: graph containing all the nodes;
1: **function** GETPROBABILITY(N, β, d, i, j)
2:     $dis \leftarrow |i - j|$;
3:     $edgeDensity \leftarrow \frac{d}{N-1}$;   ▷ p_0 in [33]
4:     $maxDistance \leftarrow \lfloor \frac{N}{2} \rfloor$;   ▷ D_{max} in [33]
5:     **if** $edgeDensity - \frac{min(dis, N-dis)}{maxDistance} \geq 0$ **then**
6:         **return** $\beta \cdot (edgeDensity - 1) + 1$;
7:     **else**
8:         **return** $\beta \cdot edgeDensity$;
9:     **end if**
10: **end function**

11: **function** GENERATENETWORK(N, β, d)
12:     Create a graph G with N nodes and no edge;
13:     **for** $i, j \in \mathbf{N}$ **do**
14:         $p \leftarrow GetProbability(N, \beta, d, i, j)$;
15:         $r \leftarrow random(0, 1)$;
16:         **if** $i = j$ or $r > p$ **then**
17:             $G.edge(i, j) \leftarrow 0$;
18:         **else**
19:             $G.edge(i, j) \leftarrow 1$;
20:         **end if**
21:     **end for**
22:     **return** G;
23: **end function**

---

in Figure 2 illustrate three block transmission modes in the Bitcoin network, where (a) is the legacy relaying mode. The node sends the inv message to notify all its peers of the arrival of a new block. The peer replies with the $getdata$ message to request the new block. (b) and (c) are the high bandwidth relaying and low bandwidth relaying modes proposed in Compact Block Relay (CBR) [36].

In the high bandwidth relaying mode the node sends $cmpctblock$ message to notify the peer of the arrival of a new block. If the peer has all the transactions contained in the new block, it will reconstruct the block locally and no further transmission is needed. But if some transactions are not available, a $getblocktxn/blocktxn$ roundtrip is needed to obtain remaining transactions.

In the low bandwidth relaying mode a peer receives an inv message and returns a $getdata(CMPCT)$ message to request the header and short transaction IDs. As in the high bandwidth relaying mode, the peer will try to reconstruct the block locally unless necessary transactions are not available. Thus, in the best case only one $cmpctblock$ message is sent so that one latency is needed. In the worst case five messages are sent and the latency is counted five times. In our simulator we count the latency three times which is the average case and ignore the message size because in most cases small-size messages instead of the entire block are sent to request
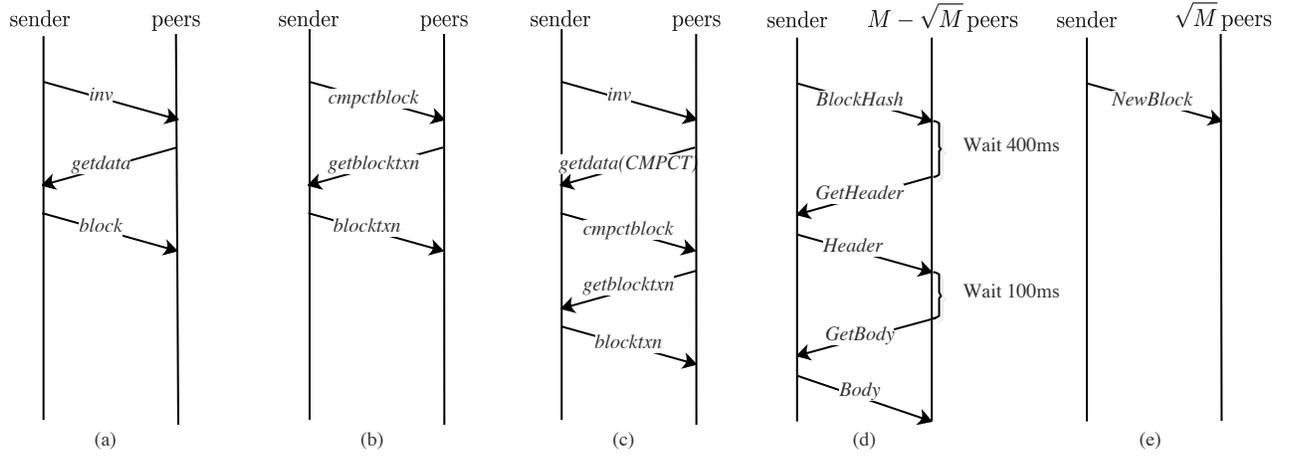
Fig. 2. (a)(b)(c) present the block transmission in Bitcoin; (d)(e) present the block transmission in Ethereum

---

Algorithm 3 Information Propagation

    Input:
    $sender$: the sender of the transaction or block;
    G: graph containing all the nodes;
    Output:
    D: transmission delays from $sender$ to other nodes;
1: Set priority queue $Q \leftarrow \varnothing$, updated node vector $V \leftarrow \varnothing$;
2: **for** $v \in G$ **do**
3:     $D[v] \leftarrow \infty$;
4:     **if** v is a neighbor of $sender$ **then**
5:         $Q.add(v)$;
6:         Calculate $D[v]$ according to Equation 1 or 2;
7:     **end if**
8: **end for**
9: $D[sender] \leftarrow 0$;
10: $V.add(sender)$;
11: **while** Q is not empty **do**
12:     $u \leftarrow Q.pop()$;
13:     $Q.remove(u)$;
14:     **for** each neighbor v of u **do**
15:         Calculate d_{uv} (delay between u and v) according to Equation 1 or 2;
16:         $D[v] \leftarrow min(D[v], d_{uv})$;
17:     **end for**
18: **end while**

---

missing transactions. So the transmission delay between two adjacent nodes can be denoted by Equation 1. D indicates the transmission delay. L indicates the latency. $size$ indicates the block size and pd indicates the block processing delay for 1 Mb block.

$$D = 3 \cdot L + pd \cdot size. \qquad (1)$$

(d) and (e) illustrate the transmission in Ethereum network. A node sends the complete block only to a small fraction of its connected peers (usually the square root of the total number of peers) and sends the hash of the new block to other peers. A peer which receives the hash at first waits for 400 ms. Then it sends the $GetHeader$ message to request the block header. After receiving the block header, it waits for 100 ms and sends the $GetBody$ message to request the block body. The sender and its peer send messages five times in total. So the transmission delay can be described by Equation 2. B denotes the bandwidth and M denotes the number of connected peers. We use the $random$ function to simulate the random choice of peers of an Ethereum node.

$$D = \begin{cases} L + size \cdot (B^{-1} + pd), & random(1, M) \leq \sqrt{M} \\ 5 \cdot L + size \cdot (B^{-1} + pd), & random(1, M) > \sqrt{M} \end{cases} \qquad (2)$$

C. Block Validation

When receiving a new block from other nodes, the receiver needs to check the generation proof and validate and execute all the included transactions, causing a considerable block processing delay which affects the block propagation delay (Equation 1 and 2). We adopt the linear model in [28] to estimate this delay. Compared to the constant or random variable used in other simulators, this active model can reflect the impact of block size on the block propagation delay better.

BlockSim:Faria used 229 ms and 240 ms as the average overall block processing delays of Ethereum and Bitcoin based on measurements. We take the two values to estimate block processing delays on each node. First, Considering the generated Bitcoin topology in Section III(B) of which the average path length is about 4 and the Bitcoin block size is 1.22 MB, the block processing delay for 0.1 MB Bitcoin block will be $240/4/1.22/10 \approx 5$ ms, which is close to the theoretical estimated block processing delay (4 ms) in [28]. Then using the same way to infer the block processing delay of Ethereum, the average path length of the generated Ethereum topology is 3.8 which is close to the estimated value (3.7) in [30]. The average Ethereum block size in 2020 is about 0.0225

MB. So the estimated processing delay for 0.1 MB Ethereum block is $229/3.8/0.0225/10 \approx 268$ ms.

D. Block Finalization

Block finalization reaches the agreement on the acceptance of blocks. Most blockchains adopt the longest chain rule to determine the accepted chain but there are still some other choices such as GHOST which is used in the early version of Ethereum. While BlockSim:Alharby supports the longest chain rule, we include the GHOST rule in our simulator as well. Process B in Figure 1 illustrates the process of the longest chain rule.

E. Incentive Mechanism

The incentive mechanism decides the way to distribute block rewards and transaction fees. Bitcoin only rewards the miners who generate blocks in the longest chain. Ethereum rewards the miners who generate blocks in the accepted chain or uncle blocks. Step 9 in Figure 1 distributes rewards according to different incentive mechanisms.

IV. TRANSACTION POOL DESIGN

Besides the above block-related processes, there are some significant transaction-related steps that seriously affect the performance of the simulator. First, the number of transactions is much larger than the number of blocks. Further, the execution times of transaction-related operations such as steps B2 and B5 in Figure 1 will increase as the number of nodes increases. Suppose there are N nodes in total, every generated new block will be propagated to all other nodes according to the Algorithm 3 so each GENERATE_BLOCK event will trigger N-1 RECEIVE_BLOCK events. Then the frequency of executing step B2 or B5 will increase with the growth of the number of nodes. Therefore, we make a special design on the transaction pool structure to avoid massive iterative operations, using bitwise operation in C++.
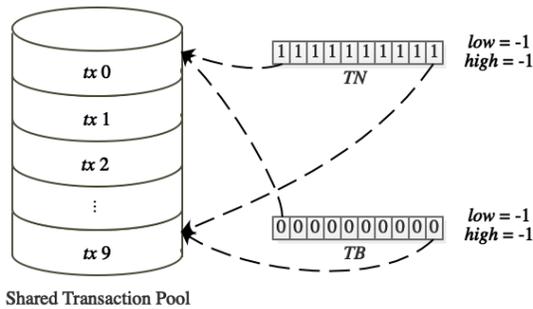

Fig. 3. Transaction storage structure

As illustrated in Figure 3, we first store the complete transactions in a transaction pool which is shared by all the nodes. These complete transactions will be stored in memory only once and will be sorted by the generation timestamp. Every node has a *mempool* to record received transactions which are not included in the local chain. We use the binary sequences, TN (transactions in the mempool of the node) and

TB (transactions in the block), to represent transactions in each node and block. Two pointers, low and $high$, are used to determine the valid range of transactions. The length of the two binary sequences is equal to the total number of transactions generated during simulation. Only the transactions between low and $high$ are available.

For example, if ten transactions are generated during simulation, TN and TB will have ten bits. The leftmost digit represents the first generated transaction and the rightmost digit represents the last generated transaction. If a digit is set to 1 and locates between low and $high$, it means the block or the mempool of the node contains this transaction. At the beginning, low and $high$ in every node and block are set to -1 which means no transactions are included in any mempool or block. All the bits of TN are initialized to 1 while all the bits of TB are initialized to 0.
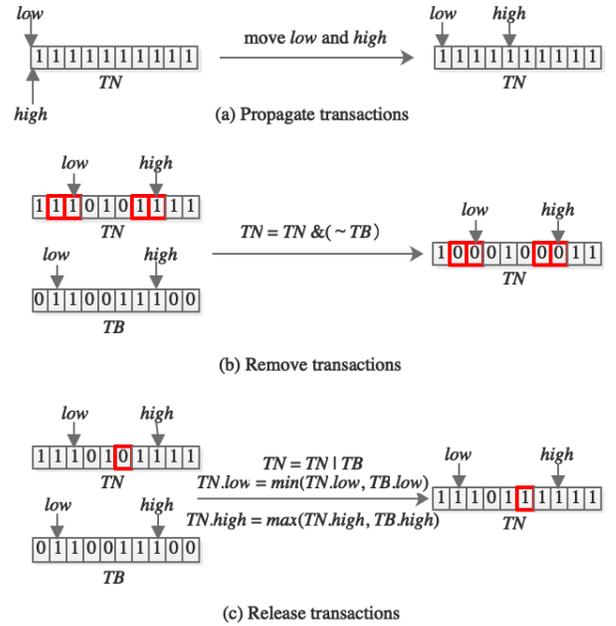

Fig. 4. (a) Move low and $high$ to propagate transactions (b) Remove transactions included in a block from the mempool (c) Release transactions back to the mempool

Suppose the number of nodes is N and the number of transactions is M. Three following operations benefit from the transaction pool structure design:

1) Transaction propagation (step A1 in Figure 1): as illustrated in Figure 4(a), before generating a new block, transactions that were generated earlier than the block generation timestamp should be propagated to all nodes. We only need to move low and $high$ to include transactions that were generated before current timestamp.

2) Transaction removal (step A4 and B2 in Figure 1): when a block is included in the local chain of a node, the node needs to remove the transactions that are included in this block from the mempool. Figure 4(b) illustrates this process which is a bitwise NAND operation.

3) Transaction release (step B5 in Figure 1): when a node receives a block from another node with a longer chain, the receiver needs to synchronize its local chain with the longer chain. Some blocks in the local chain will become stale or uncle blocks. The transactions included in these blocks should be released back to the mempool to be included again later. Figure 4(c) illustrates this process which is a bitwise OR operation.

V. CBLOCKSIM VALIDATION

We validate CBlockSim by comparing the simulation results generated by CBlockSim with the observed realistic data of Bitcoin and Ethereum. To deal with different application scenarios, CBlockSim can be configured to enable or disable network simulation. Users who are not willing to simulate network details can use an exponential distribution to simulate a delay as in BlockSim:Alharby.

We first simulate Bitcoin using data from 2020 and compare the result with the realistic data presented in [23]. Then we simulate Ethereum observed in [31] and compare the result with the data obtained from Etherscan[1] and Ethstats[2]. The experiment is performed ten times and we take the average value as the result. The simulation configuration is presented in Table I.

TABLE I
SIMULATION CONFIGURATION

	Bitcoin	Ethereum
#Nodes	11,000	8,223
Simulation Time (s)	600,000	86,400
Block Interval (s)	600	13.05
Geographical Node Distribution	[23]	[31]
Average Degree	12	19.747
Topology Controlling Parameter	1	0.24
Bandwidth/Latency Distribution	[13]	[13]
Hash Power Distribution	[35]	[35]

Table II compares real data and the simulation output including the average block size, the block propagation delay and the stale/uncle rate. When disabling the network module, the fixed average block propagation delay and block size are used as the input parameter and the result is consistent with the simulated result of BlockSim:Alharby. We set the average block propagation delay to 0.7 seconds for both Bitcoin and Ethereum. In this condition the simulated 50th percentile of the block propagation delay is approximately 0.5 seconds which is consistent with real data.

The simulator output is close to the real observed data. When enabling the network topology simulation, either the simulated block propagation delay of Ethereum or Bitcoin is close to the observed one which means the network models we used are relatively precise.

TABLE II
COMPARISON OF AVERAGE BLOCK SIZE s_B, 50TH AND 90TH PERCENTILE OF BLOCK PROPAGATION DELAY t_{BPD}, AND STALE/UNCLE RATE r

	s_B(MB)	t_{BPD}^{50th} (s)	t_{BPD}^{90th} (s)	r
Bitcoin[m]	1.22	0.5	3.3	0.06%
Bitcoin[a]	1.21±0.02	0.49±0.001	1.61±0.003	0.11%±0.006%
Bitcoin[b]	1.22±0.01	0.35±0.005	1.34±0.002	0.08%±0.001%
Ethereum[m]	0.023	0.5	1.75	5.36%
Ethereum[a]	0.024±0.001	0.49±0.001	1.61±0.002	4.80%±0.15%
Ethereum[b]	0.024±0.002	0.51±0.002	1.72±0.007	5.15%±0.16%

[m] Observed data.
[a] Disable network topology simulation.
[b] Enable network topology simulation.

VI. PERFORMANCE EVALUATION

In order to evaluate the performance of the proposed simulator, we compare the run time and memory usage of CBlockSim, BlockSim:Faria and BlockSim:Alharby while varying the number of nodes and simulation time. All three simulators perform both Bitcoin and Ethereum simulation and have complete simulation functions. Other popular simulators such as Bitcoin Simulator and SimBlock can only simulate Bitcoin and are short of simulation for transactions. All the experiments are performed to simulate Ethereum as observed in [31] on a Linux virtual machine with 1 CPU and 32 GB memory. We use the Ethereum configuration in Table I.

We first set the number of nodes to $100, 200$ and 300, respectively. The simulation time is set to $6, 12, 18$ and 24 hours, respectively. Figure 5 presents the results. Block-Sim:Faria can only simulate a small scale blockchain with short simulation time. The memory usage of BlockSim:Faria is much higher than of the other two simulators. For 200 nodes and a simulation time of more than 12 or 300 nodes and a simulation time of more than 6, it will run out of memory (more than 32 GB). Both BlockSim:Alharby and CBlockSim can simulate a blockchain well when the number of nodes varies from 100 to 300 and the simulation time varies from 6 to 24 hours. The difference in memory usage is significant because BlockSim:Faria simulates many details of a blockchain while the other two simulators simplify some components. BlockSim:Alharby shares transactions among all the nodes and uses the exponential distribution to simulate latency. CBlockSim stores complete transactions only once and adopts the binary encoding to represent transactions in each node and block. It further combines a realistic network topology and the shortest path algorithm to simulate network communication.

With increasing number of nodes, a fast growth of the time overhead of BlockSim:Alharby is observed from minutes to hours. In opposition, the time overhead of CBlockSim is low and it increases much slower, from seconds to a few minutes. In Figure 6 we evaluate CBlockSim individually when setting the number of nodes to $1,000$ and $10,000$ because the time overhead of BlockSim:Alharby is too large in this condition. The high time overhead impairs the application of
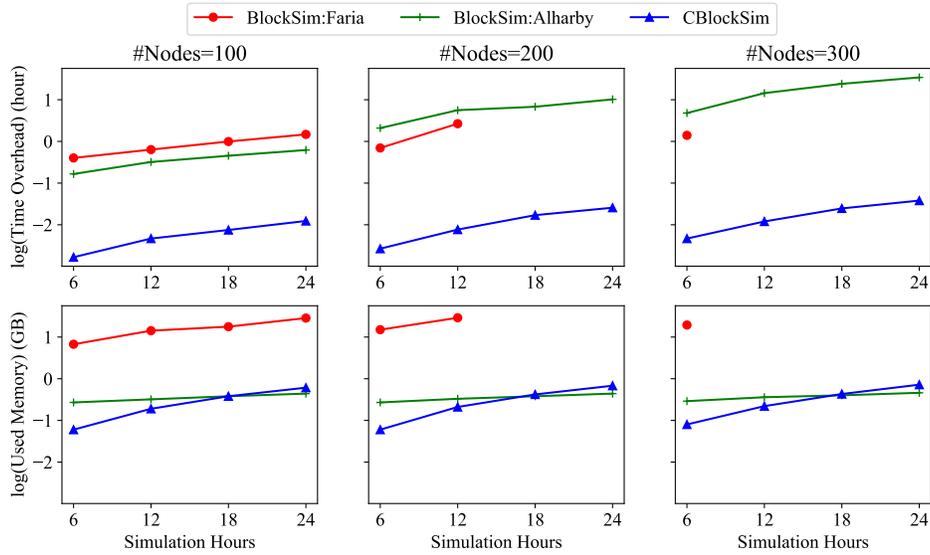
Fig. 5. Comparison of time overhead and memory usage

BlockSim:Alharby. The scale of $10,000$ nodes is close to the real blockchain, either Bitcoin or Ethereum. When simulating a blockchain with $10,000$ nodes for 24 hours, CBlockSim takes about one and half hours and 4.5 GB of memory. This demonstrates that CBlockSim has low time overhead and can be applied to a large-scale blockchain with ten thousand nodes.

Fig. 6. Evaluation of CBlockSim's scalability

VII. USE CASES

Using CBlockSim we can investigate some network-related issues that many other simulators do not cover. In this section we propose two use cases, investigating the effect of different transmission protocols and the impact of different average node degree on the blockchain network.

A. Compact Block Relay v.s. Eth Wire Protocol

Bitcoin uses a compact block relay to reduce the delay caused by increased block size while Ethereum applies the *Ethereum wire* protocol to propagate blocks. The two protocols

are illustrated in Figure 2. To compare the two protocols, we use the Ethereum configuration in Table I, vary the block size from 0.02 MB to 0.1 MB and load the two different protocols in the information propagation module. Figure 7 presents the 50th and 90th percentile of the block propagation delay. Figure 8 presents the uncle rate when using the two protocols.
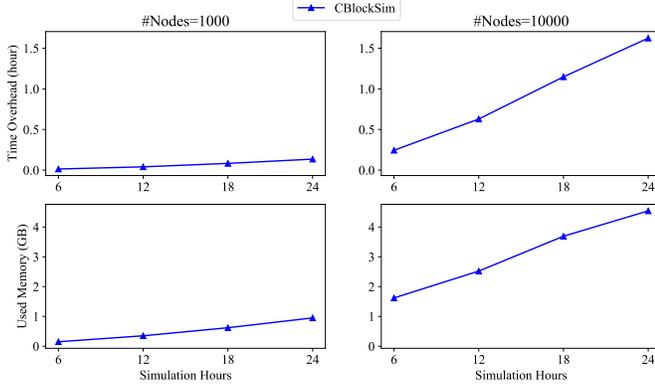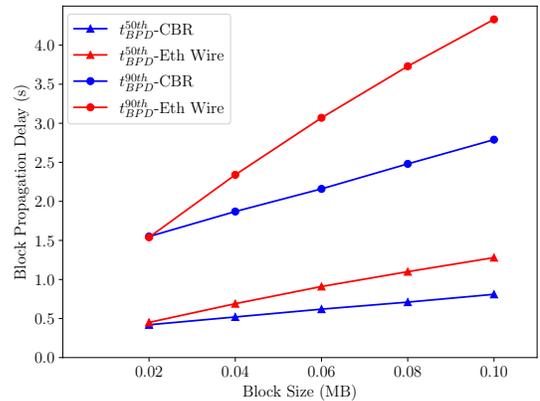

Fig. 7. The 50th and 90th percentile of block propagation delay, using CBR and Eth Wire Protocol with different block sizes

When the block size is 0.02 MB which is close to the average block size of Ethereum observed in [31] (March 2020), the two protocols have almost identical block propagation delay. As the block size increases, the block propagation delay and uncle rate grow and CBR outperforms the Ethereum Wire Protocol increasingly. Obviously, the block size has a significant impact on the block propagation delay and uncle rate. If the block size of Ethereum continues to increase, the transmission protocol should be improved in the future. For researchers or developers who aim to build an Ethereum-like
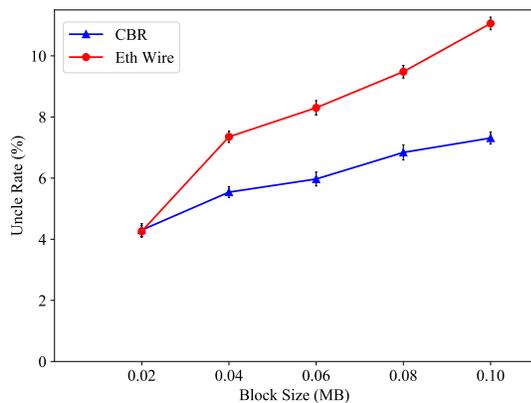
Fig. 8. The uncle rate, using CBR and Eth Wire Protocol with different block sizes

Fig. 10. Impact of the average node degree on the uncle rate

blockchain, it is necessary to consider using CBR to improve the blockchain performance if the block size is rather large.

B. The Impact of Average Degree

Besides the transmission protocols the average node degree also affects the blockchain network significantly. Increasing the average node degree reduces the block propagation delay but results in more network traffic. In this use case we use CBlockSim to investigate the impact of the average node degree on the blockchain performance. We vary the average node degree of Ethereum from 10 to 100 and the simulated block propagation delay and uncle rate are presented in Figures 9 and 10.
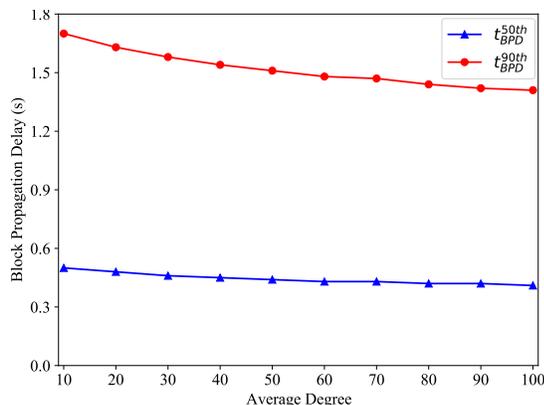

Fig. 9. The impact of average degree on block propagation delay

A continuous decrease of the block propagation delay and uncle rate can be observed as the average node degree increases. As the average degree increases, the uncle rate drops fast at first and then gradually slows down. When the average degree increases from 10 to 50, the uncle rate has dropped by about 0.6 percent. But when the degree increases from 50 to 100, the uncle rate has only dropped by about 0.3 percent.

VIII. Conclusion

The paper presented CBlockSim which is a modular high-performance blockchain simulator extended from Block-
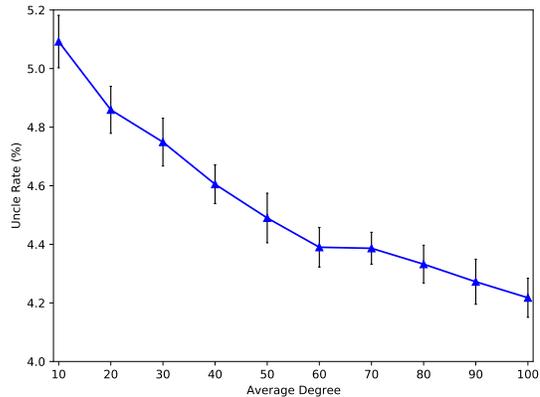
Sim:Alharby. We publish our code on Github[3]. Various blockchain components including different consensus protocols, network transmission protocols and finalization rules are integrated into the simulator in the form of individual modules. Users can combine existing modules flexibly or extend the simulator by adding new modules to simulate different blockchains. The design of the transaction pool structure which uses a binary sequence and bitwise operation improves the simulation performance. The simulation time is reduced by an order of magnitude and the scale of the network can be extended up to ten thousand nodes or more. With the proposed network module the simulator can be applied to network-related research which is not covered by many other simulators. Two typical network-related issues are investigated as use cases.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008, [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] V. Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform," 2014, [Online]. Available: https://ethereum.org/en/whitepaper/

[3] F. Schär, "Decentralized finance: On blockchain-and smart contract-based financial markets," *Federal Reserve Bank of St. Louis Research Paper Series*, 2021.

[4] M. Kowalski, Z.W. Lee, and T.K. Chan, "Blockchain technology and trust relationships in trade finance. Technological Forecasting and Social Change," *Technological Forecasting and Social Change*, vol. 166, 2021.

[5] X. Yang, X. Yang, X. Yi, I. Khalil, X. Zhou, D. He, X. Huang, and S. Nepal, "Blockchain-Based Secure and Lightweight Authentication for Internet of Things," *IEEE Internet of Things Journal*, early access, July 19, 2021. DOI: 10.1109/JIOT.2021.3098007.

[6] T. Li, W. Liu, A. Liu, M. Dong, K. Ota, N.N. Xiong, and Q. Li, "BTS: A Blockchain-based Trust System to Deter Malicious Data Reporting in Intelligent Internet of Things," *IEEE Internet of Things Journal*, early access, May 31, 2021. DOI: 10.1109/JIOT.2021.3085004.

[7] J. Yang, A. Paudel, H.B. Gooi, and H.D. Nguyen, "A Proof-of-Stake public blockchain based pricing scheme for peer-to-peer energy trading," *Applied Energy*, vol. 298, 117154, 2021.

[8] M. Foti, C. Mavromatis, and M. Vavalis, "Decentralized blockchain-based consensus for Optimal Power Flow solutions," *Applied Energy*, vol. 283, 116100, 2021.

---

[3]https://github.com/xuyangm/CBlockSim

[9] K. Miyachi and T.K. Mackey, "hOCBS: A privacy-preserving blockchain framework for healthcare data leveraging an on-chain and off-chain system design," *Information Processing & Management*, vol. 58, 102535, 2021.

[10] R. Zou, X. Lv, and J. Zhao, "SPChain: Blockchain-based medical data sharing and privacy-preserving eHealth system," *Information Processing & Management*, vol. 58, 102604, 2021.

[11] A. Gervais, G.O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3-16, 2016.

[12] M. Alharby, A.V. Moorsel, "Blocksim: a simulation framework for blockchain systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, pp. 135-138, 2019.

[13] Y. Aoki, K. Otsuki, T. Kaneko, R. Banno, and K. Shudo, "Simblock: A blockchain network simulator," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops*, pp. 325-329, 2019.

[14] B. Putz, M. Dietz, P. Empl, and G. Pernul, "Ethertwin: Blockchain-based secure digital twin information management," *Information Processing & Management*, vol. 58, 102425, 2021.

[15] K.P. Yu, L. Tan, M. Aloqaily, H. Yang, and Y. Jararweh, "Blockchain-enhanced data sharing with traceable and direct revocation in IIoT," *IEEE transactions on industrial informatics*, vol. 17, pp. 7669-7678, 2021.

[16] R. Kumar, B. Palanisamy, and S. Sural, "BEAAS: Blockchain Enabled Attribute-Based Access Control as a Service," in *2021 IEEE International Conference on Blockchain and Cryptocurrency*, pp. 1-3, 2021.

[17] R.A. Mishra, A. Kalla, A. Braeken, and M. Liyanage, "Privacy protected blockchain based architecture and implementation for sharing of students' credentials," *Information Processing & Management*, vol. 58, 102512, 2021.

[18] J.A. Chang, M.N. Katehakis, J.J. Shi, Z. Yan, "Blockchain-empowered Newsvendor optimization," *International Journal of Production Economics*, vol. 238, 108144, 2021.

[19] C. Piao, Y. Hao, J. Yan, and X. Jiang, "Privacy preserving in blockchain-based government data sharing: A Service-On-Chain (SOC) approach," *Information Processing & Management*, vol. 58, 102651, 2021.

[20] L. Stoykov, K. Zhang, H.A. Jacobsen, "Vibes: fast blockchain simulations for large-scale peer-to-peer networks," in *18th ACM/IFIP/USENIX Middleware Conference*, pp. 19-20, 2017.

[21] A. Deshpande, P. Nasirifard, and H.A. Jacobsen, "eVIBES: configurable and interactive ethereum blockchain simulation framework," in *19th International Middleware Conference*, pp. 11-12, 2018.

[22] C. Faria and M. Correia, "BlockSim: blockchain simulator," in *2019 IEEE International Conference on Blockchain*, pp. 439-446, 2019.

[23] R. Paulavičius, S. Grigaitis, and E. Filatovas, "A Systematic Review and Empirical Analysis of Blockchain Simulators," *IEEE Access*, vol. 9, pp. 38010-38028, 2021.

[24] Y. Xiao, N. Zhang, W. Lou, and Y.T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, pp. 1432-1465, 2020.

[25] W. Zhao, S. Yang, X. Luo, and J. Zhou, "On PeerCoin Proof of Stake for Blockchain Consensus," in *2021 The 3rd International Conference on Blockchain Technology*, pp. 129-134, 2021.

[26] W. Hao, J. Zeng, X. Dai, J. Xiao, Q. Hua, H. Chen, K. Li, and H. Jin, "BlockP2P: Enabling fast blockchain broadcast with scalable peer-to-peer network topology," in *International Conference on Green, Pervasive, and Cloud Computing*, pp. 223-237, 2019.

[27] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *The 1st ACM Conference on Advances in Financial Technologies*, pp. 199-213, 2019.

[28] Y. Shahsavari, K. Zhang, and C. Talhi, "A theoretical model for block propagation analysis in bitcoin network," *IEEE Transactions on Engineering Management*, pp. 1-18, 2020.

[29] Y. Gao, J. Shi, X. Wang, Q. Tan, C. Zhao, and Z. Yin, "Topology measurement and analysis on ethereum p2p network," in *2019 IEEE Symposium on Computers and Communications*, pp. 1-7, 2019.

[30] T. Wang, C. Zhao, Q. Yang, S. Zhang, and S.C. Liew, "Ethna: Analyzing the Underlying Peer-to-Peer Network of Ethereum Blockchain," *IEEE Transactions on Network Science and Engineering*, early access, May 07, 2021. DOI: 10.1109/TNSE.2021.3078181.

[31] S. Maeng, M. Essaid, C. Lee, S. Park, and H. Ju, "Visualization of Ethereum P2P network topology and peer properties," *International Journal of Network Management*, e2175, 2021.

[32] D.J. Watts and S.H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol.393, pp. 440-442, 1998.

[33] H.F. Song and X.J. Wang, "Simple, distance-dependent formulation of the Watts-Strogatz model for directed and undirected small-world networks," *Physical Review E*, vol. 90, 062801, 2014.

[34] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering bitcoin's public topology and influential nodes," 2015. [Online]. Available: https://www.cs.umd.edu/projects/coinscope/coinscope.pdf.

[35] R. Nagayama, R. Banno, and K. Shudo, "Identifying impacts of protocol and internet development on the bitcoin network," in *2020 IEEE Symposium on Computers and Communications*, pp. 1-6, 2020.

[36] *BIP-0152*, 2016. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki/.