Simulation of a P2P Cryptocurrency Network

Phool Roshan Jha (24D0366), K. R. Ramya Shri Shakthi (24D0369), Ritika Jaiswal (24M0855)

February 8, 2025

Abstract

This report presents the design and implementation of a discrete-event simulator for a peer-to-peer (P2P) cryptocurrency network. The simulator models key aspects such as transaction generation, block mining with Proof-of-Work (PoW), block propagation, and blockchain tree maintenance. In this report, we have mentioned the details of the choices behind using exponential distributions for interarrival times and queuing delays, loop-less transaction forwarding, blockchain tree maintenance and visualization approach, and analyzed how various network parameters (e.g., node speed, CPU power) affect the network's behavior.

1 Introduction

Cryptocurrency networks, such as Bitcoin, operate in a decentralized environment where nodes generate transactions, mine blocks, and propagate information across a distributed network. In this project, we simulate these dynamics by modeling:

- Transaction generation by peers with interarrival times sampled from an exponential distribution.
- Block mining using a Proof-of-Work (PoW) mechanism, where each node mines at a rate proportional to its hashing power.
- Network latencies that affect the propagation of transactions and blocks.
- Blockchain tree maintenance, capturing forks and orphaned blocks.

2 Simulation Architecture

2.1 Key Components

- **Transaction Class:** Each transaction includes a sender, receiver, amount, timestamp, and a unique transaction ID generated via hashing. This class models both standard transactions and coinbase transactions (mining rewards).
- **Block Class:** A block encapsulates a set of transactions, its own unique ID, a pointer (hash) to the previous block, a timestamp, and a nonce used in the PoW process. Blocks have size constraints (a maximum of 1 MB and a minimum size of 250 bytes for empty blocks).
- **Blockchain Class:** This class maintains the main chain as well as orphaned blocks and potential forks. It handles block validation, chain extension, and fork resolution by selecting the longest valid chain.

- **Peer Class:** Peers simulate network nodes with attributes such as unique ID, speed (fast or slow), CPU type (high or low), a mempool for transactions, and hashing power. They generate transactions, propagate them, and mine blocks. We have also ensured loopless forwarding by tracking which transactions and blocks have been sent or received from each peer.

3 Transaction Generation

Each peer generates transactions with interarrival times sampled from an **exponential distribution** with mean T_{tx}.

Why Exponential Distribution? The exponential distribution is used for transaction interarrival times because it models a Poisson process, where events occur randomly and independently over time. It has the memoryless property, meaning the next transaction's arrival time does not depend on the previous one, making it ideal for decentralized networks. This distribution also realistically simulates the bursty nature of transactions in real blockchain networks. Additionally, it is computationally efficient to generate and helps model network congestion effectively.

4 Simulating Network Latencies

The latency L_{ij} between two connected peers i and j is modeled as:

$$L_{ij} = \rho_{ij} + \frac{|m|}{c_{ij}} + d_{ij},$$

where:

- ρ_{ij} is a minimum delay chosen uniformly between 10 ms and 500 ms (representing speed-of-light propagation delays).
- $|m|$ is the message size in bits.
- c_{ij} is the link speed (set to 100 Mbps if both nodes are fast, and 5 Mbps if either node is slow).
- d_{ij} is the queuing delay, sampled from an exponential distribution with mean $\frac{96\,\text{kbits}}{c_{ij}}$.

Reason for Inverse Relationship of d_{ij} and c_{ij}

The queuing delay d_{ij} is inversely proportional to the link speed c_{ij} based on queuing theory. In an M/M/1 queue, the average delay is given by:

$$d_{ij} = \frac{1}{\mu - \lambda}$$

where μ is the service rate, which depends on c_{ij}. Since

$$\mu = \frac{c_{ij}}{S},$$

substituting this gives:

$$d_{ij} \approx \frac{S}{c_{ij}}$$

for low arrival rates. This shows that as c_{ij} increases, d_{ij} decreases, proving the inverse relationship. Also, higher link speeds allow messages to be transmitted faster, reducing congestion and waiting times. This reflects real-world network behavior where higher bandwidth links incur lower delays.

5 Loop-less Transaction Forwarding

To prevent transactions from circulating in endless loops, a node forwards any received transaction to a connected peer only if it has not already sent that transaction to that peer and has not received it from that peer. This mechanism ensures efficient propagation without redundancy.

6 Proof-of-Work (PoW) and Block Propagation

6.1 Block Mining and Tree Maintenance

- **Genesis Block:** All nodes begin with a common genesis block.
- **Block Creation:** When a node receives a block that extends its longest chain, it forms a new block by selecting unconfirmed transactions from its mempool. The block includes a coinbase transaction granting a reward (e.g., 50 coins) to the miner.
- **Mining Time T_k:** Each node k draws a mining time T_k from an exponential distribution with mean $\frac{I}{h_k}$, where I is the average interarrival time between blocks (e.g., 600 seconds) and h_k is the node's fraction of total hashing power. High CPU nodes have 10 times the hashing power of low CPU nodes, resulting in shorter T_k.
- **Block Validation and Propagation:** Upon receiving a block, nodes validate that all transactions are valid (ensuring no negative balances) and that the block size does not exceed 1 MB. If valid, the block is added to the node's blockchain tree and propagated to its connected peers.
- **Fork Resolution:** Nodes maintain a complete tree of received blocks and choose the longest valid chain as the canonical blockchain. Orphaned blocks (blocks not in the longest chain) are maintained for record-keeping and further analysis.

6.1.1 Explanation for the Choice of Mean Block Interarrival Time in PoW Simulation

In our simulation, we set the mean block interarrival time to $\mathbf{I = 600}$ **seconds**, similar to Bitcoin, ensuring a controlled and realistic blockchain growth rate. This choice aligns with the **security and stability of decentralized networks**, as a longer interarrival time reduces stale blocks and minimizes network congestion.

Each miner draws its mining time

$$T_k \sim \text{Exp}\left(\frac{I}{h_k}\right),$$

where h_k represents its fraction of total hashing power. High CPU nodes have 10 times more hash power than low CPU nodes, meaning they mine blocks faster. This ensures a fair distribution of blocks and mimics real-world mining conditions, where computationally stronger nodes mine blocks more frequently.

Thus, choosing $\mathbf{I = 600s}$ balances security, fairness, and network efficiency while preventing excessive forks and orphan blocks in the simulation.

7 Blockchain Tree Maintenance and Visualization

7.1 Tree Maintenance

Each node maintains a complete tree of all blockchains it hears from the start of the simulation. For every block, the node records:

- The block's unique ID.
- The time of arrival.

At the end of the simulation, this information is written to a file for analysis.

7.2 Visualization

We have used **NetworkX with Matplotlib** as our visualization tool to study the blockchain tree structure.

7.3 Experimental Observations and Insights

By varying parameters such as the number of peers n, the percentage of slow nodes z, transaction interarrival time T_{tx}, queuing delay d_{ij}, and mining time T_k, the following insights were observed:

- **Node Performance:** Fast, high-CPU nodes consistently contribute a higher fraction of blocks to the longest chain compared to slow, low-CPU nodes.
- **Forking Dynamics:** Increased network latency and lower link speeds lead to more frequent forks, resulting in a blockchain tree with many short, orphaned branches.
- **Impact of Parameters:** A higher T_{tx} (longer transaction interarrival) or increased queuing delay d_{ij} reduces the number of transactions per block and overall block creation rate, making the blockchain tree less dense.

For one of the result we got, the ratio of the number of blocks generated by each node in the longest chain of the tree to the total number of blocks it generates at the end of the simulation for each peer is as follows:

- **Peer 9**: 2/5
- **Peer 8**: 1/5
- **Peer 1**: 1/5

Total number of blocks in the longest chain of the blockchain is 5 and one of the blocks out of the five is the genesis blocks.