Blockchain Arena

Simulating Mining Wars and Network Attacks

Project 1

Simulation of a P2P Cryptocurrency Network

Release Date: June 13, 2025

Due Date: 23:59hrs, June 25, 2025

Blockchain Arena
Simulating Mining Wars & Network Attacks
Build. Battle. Secure. Join the Arena!

Objective

In this assignment, you will build your own **discrete-event simulator** for a **P2P cryptocurrency network**. This simulation will be built upon your first assignment where you have created a P2P network topology.
Discrete-Event Simulator: It maintains an **event-queue** from which the earliest event is executed. This event may create further future events that get added to the queue. For example, an `send_block` event in which one node sends a block to connected peers will create future events of `receive_block` at its peers.

Suggestions

- I recommend using Python or C++ for this assignment. You can use any other programming language, but ensure that the code is well-structured and modular.
- If you include code from a publicly available source, it should be clearly mentioned in the comments. Don't copy-paste code from online GitHub repositories if available.
- Use of Generative AI tools like ChatGPT, Gemini, etc. is allowed for template code, but make sure to understand the code and modify it as per the requirements. **Don't** copy-paste code generated by these tools without understanding it.

Network and Node Properties

The cryptocurrency network must have the following properties:

1. There are n peers, each with a unique ID, where n is a parameter set at the time of initiation of the network. Some of these nodes (say z_0 percent, where z_0 is a command line simulation parameter) are labeled `slow` and the others `fast`. In addition, some of these nodes (say z_1 percent, where z_1 is again a command line simulation parameter) are labeled `low CPU` and the others `high CPU`. We will use this classification below. **Tip:** Create a class `Peer` with required attributes.

2. Each peer generates transactions randomly in time. The interarrival time between transactions generated by any peer is chosen from an exponential distribution whose `mean time` (T_{tx}) can be set as a parameter of the simulator.

   - (Mention in report) What are the theoretical reasons of choosing the exponential distribution for interarrival time sampling?

3. Each transaction has the format: "TxnID: ID_x pays ID_y C coins". You must ensure that C is less than or equal to the coins currently owned by ID_x (ID of the peer that generates the transaction) before including it in a block. ID_y should be the ID of any other peer in the network to which the transaction is destined (ID_y should be chosen randomly). The size of each transaction is assumed to be 1 KB. **Tip:** Create a class `Transaction` with attributes like `TxnID`, `ID_x`, `ID_y`, `C` etc.

4. **(Network Topology)** This is the same P2P network that you have created as part of the first assignment. The P2P network should satisfy the following properties:

- Each peer is randomly connected to between 3 and 6 other peers.
- The network is undirected and connected.

Tip: Use the `Network` class that you have implemented in the first assignment to represent the P2P network and maintain a list of peers.

5. Simulate latencies L_{ij} between pairs of peers i and j connected by a link. Latency is the time between which a message m was transmitted from the sender i and received by another node j. Choose the latency to be of the form $\rho_{ij} + |m|/c_{ij} + d_{ij}$, where ρ_{ij} is a positive minimum value corresponding to the speed of light propagation delay, $|m|$ denotes the length of the message in bits, c_{ij} is the link speed between i and j in bits per second, and d_{ij} is the queuing delay at node i to forward the message to node j. d_{ij} is randomly chosen from an exponential distribution with mean $96kbits/c_{i,j}$. Note that $d_{i,j}$ must be randomly chosen for each message transmitted from i to j. ρ_{ij} can be chosen from a uniform distribution between 10ms and 500ms at the start of the simulation. c_{ij} is set to 100 Mbps if both i and j are fast, and to 5 Mbps if either of the nodes is slow.

   - (Mention in report) Why is the mean of d_{ij} (queuing delay) inversely related to c_{ij} (link speed)? Give justification for this choice.

6. A node forwards any transaction it receives from one peer to another connected peer, as long as it has not already sent the transaction to that peer and has not received it from that peer. This prevents transactions from circulating in endless loops (loop-less forwarding).

Simulating Proof-of-Work (PoW)

7. All nodes have the genesis block at the start of the simulation. Each block must have a unique ID, say BlkID (use a suitable method to generate unique block Id). Any peer, say peer k, maintains a tree of blocks as in bitcoin. When it receives a block from another peer, it validates all its transactions (no balance of any peer should go negative), and if the block is valid it adds the block to its tree.

   When a received block creates a new longest chain at peer k (longer than the previous longest chain), say at time t_k, then we simulate PoW mining of a new block as follows. Peer k forms a block at t_k by selecting a subset of the transactions received so far and not included in any blocks in the longest chain. After forming a block, node k generates a random variable T_k, at time t_k, as follows:

   Suppose the interarrival time between blocks on average is I (for example, I is 600 sec in Bitcoin), and node k has fraction h_k (where $0 < h_k < 1$) of the total hashing power. Then T_k is drawn from an exponential distribution with mean equal to I/h_k. If peer k is a high CPU node, then it is assumed to have 10 times higher hashing power than a low CPU node. (Note: ensure that the $\sum_k h_k = 1$ in the simulation).

   If peer k has the same longest chain at time $t_k + T_k$ then it broadcasts a new block. (If it no longer has the same longest chain at $t_k + T_k$ then the event scheduled for mining a block at $t_k + T_k$ is terminated.) This block lists the BlkID of the last block in the longest received chain of the node and the list of transactions included in it. Once broadcast, the node starts creating a new block from this terminal block which is now the `longest_chain_tip`. The block is assumed to contribute **50** coins to k (i.e. ID_k) as a mining fee, and this is included as a coinbase transaction "TxnID: ID_k mines 50 coins". On the contrary, if node k receives the block from another node in the network, it validates all its transactions. Valid transactions are those where the sender has sufficient balance to perform the transaction. After validation, the node again starts block creation, as explained above.

The block propagates in the network just like individual transactions. A block can have size at max 1 MB (8×10^6 bits) and actual size depends on the number of transactions encapsulated in the block. An empty block (with no transactions besides the coinbase) is assumed to be of size 1KB. Ensure that you do not add too many transactions in the block such that the size of the block exceeds the maximum permitted size i.e. 1 MB. Make sure proper resolution of forks is done, i.e., the longest chain is always maintained.

- (Mention in report) The theoretical reason for choosing the exponential distribution for T_k and the choice of its values set by you during experiments.

Tip: Create a `block` class to manage block creation, block validation, mining, etc.

8. Each node maintains a tree of all blockchains heard since the start of the simulation. The node stores the time of arrival of every block in its tree. This information is written to a file at the end of the simulation.
   Tip: Use a class `BlockchainTree` to manage the tree of blocks. The tree should have methods to add blocks, validate blocks, and find the longest chain.

Hint: How to build the Simulator? 🤔

I recommend to build a centralized simulator around a single event queue. The simulator's core job is to process events from this queue in chronological order based on time.

To guide your design, consider the following tips:

- Create a central `Simulator` class to manage the global time and the event queue.
- An `Event` class could represent different actions. Using an `enum` for event types (e.g., `GENERATE_TRANSACTION`, `RECEIVE_BLOCK`) is a good way to identify them.
- The main loop would dequeue the earliest event and dispatch it to the correct node that will handle the event based on its type.
- A key concept is that handling one event can create new, future events that are added back to the queue. For example, a node receiving a block will then create future events for its peers to receive that same block.

Analysis and Experimentation

Use an appropriate visualization tool to study the blockchain tree. Experiment with choosing different values for different parameters (n, z_0, z_1, T_{tx}, mean of T_k etc.). Find 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. How does this ratio vary depending on whether the node is fast, slow, low CPU, high CPU power etc.? How long are branches of the tree measured in number of blocks? Give detailed insights to explain your observations and findings in the report.

Submission Guidelines

Submit the GitHub repository link containing the following files:

1. Source code for simulator and visualization tool that you have used.
2. `README.md` file with instructions for compiling and running.
3. A report that includes the explnations as per the assignment, detailing your findings along with pictures of typical blockchain trees and appropriate insights.

Submission Link: Google Form for Submission
Deadline: 23:59hrs, June 25, 2025.

Be patient and Enjoy building the simulator!