CS765 Project Part-1

Simulation of a P2P Cryptocurrency Network
Report

Saksham Rathi (22B1003), Kavya Gupta (22B1053), Mayank Kumar (22B0933)
Department of Computer Science,
Indian Institute of Technology Bombay

Point 2

Theoretical reasons of choosing the exponential distribution for interarrival time sampling are:-

- The most important theoretical reason for using the exponential distribution for inter-arrival times is its **memoryless property**, which means that the probability of an event occurring in the future is independent of the past. Specifically:

  $$P(X > s + t | X > s) = P(X > t)$$

  This is particularly helpful as this makes the future transactions independent of the past which is true as all transactions are independently randomly generated.

- It's **correspondence to the Poisson** distribution. The Poisson distribution is about the most basic count process, with constant event rate and no memory. This can help model the generation of transactions appropriately. Whenever a Poisson models the distribution of events then an Exponential models the distribution of inter-event times.

Point 5

The mean of $d_{i,j}$ (queuing delay) is inversely related to $c_{i,j}$ (link speed) because a higher link speed allows messages to be transmitted more quickly, reducing the time they spend in the queue. For example, if a link has higher capacity, it can process and forward packets at a faster rate, reducing the time packets spend in the queue. Conversely, a lower link speed results in congestion, as packets arrive faster than they are transmitted, increasing queuing delays. Therefore delay is inversely proportional to link speed.

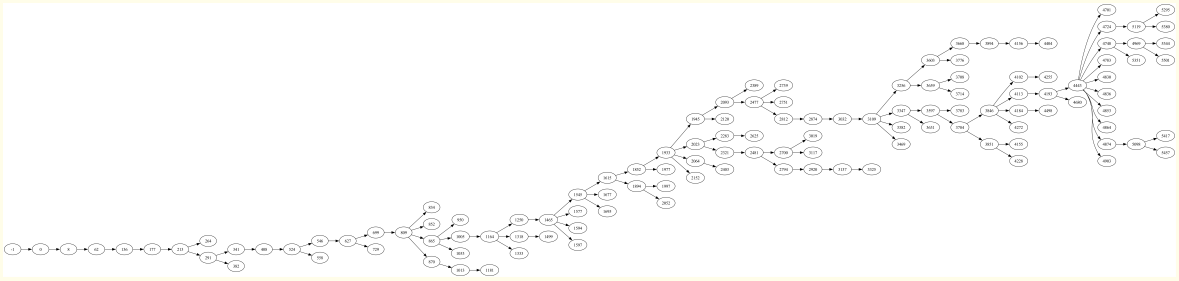
Point 7

We chose the mean block arrival time (I) as 600 seconds. The reasons for it are as follows:-

- Choosing a very high value would result in a very low number of blocks being generated in the network, which would not be a good representation of a real-world cryptocurrency

network. In our simulation, we found out that (for certain parameters) I = 600 sec will have ≈ 10 times the longest chain height than I = 60000 sec in half the time. For I = 6000000 sec, the simulation didn't give any meaningful results, due to too many transaction events.

- Choosing a very low value would likely lead to many instances of forking, orphaning a lot of blocks and hence a very branched tree. For Ethereum, it is 15 seconds and for that our simulation returns this dense tree:-



- For Bitcoin blockchain, this value is 600 seconds as well, hence choosing this value would be likely provide simulations closer to the actual things.

Flowchart of Event Execution

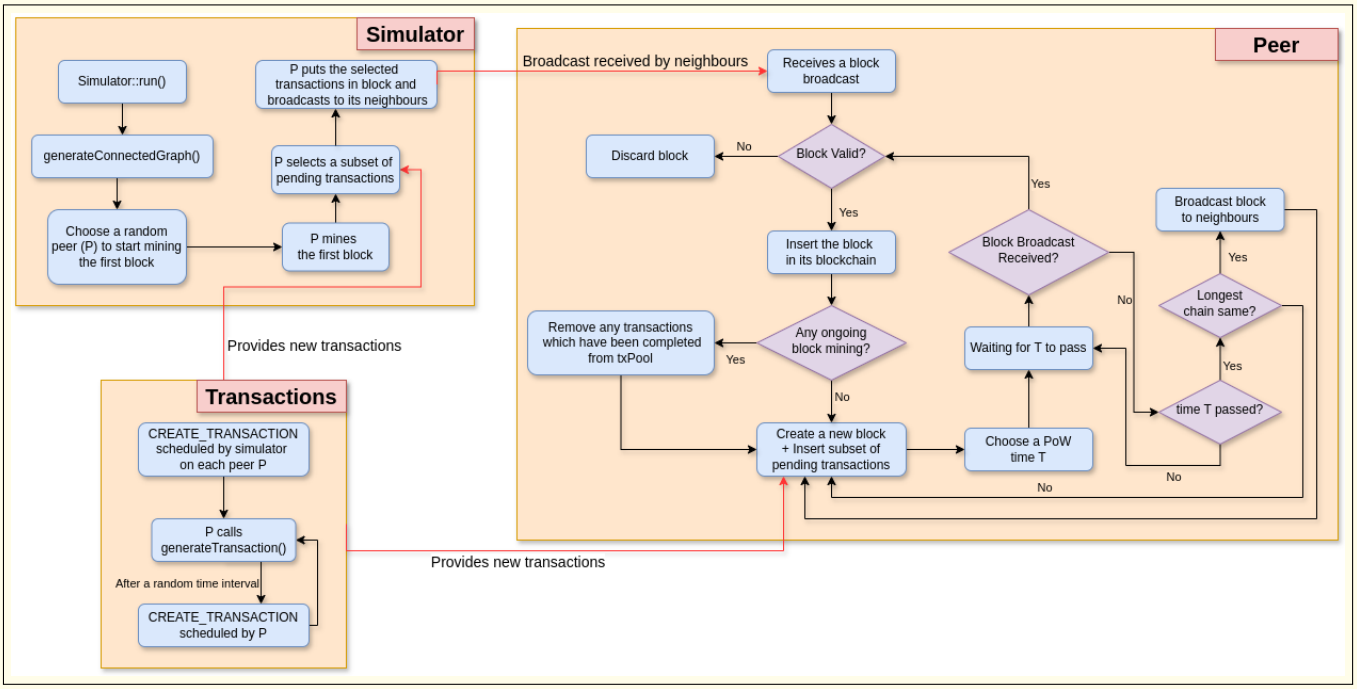


Figure 1: Flow of events

Analysis

We are analyzing the following ratio:

$$\text{Ratio}_i = \frac{\text{Total number of transactions in the longest chain from peer } i}{\text{Total number of transactions produced by peer } i \text{ in the network}}$$

Here is the default list of parameters which we have chosen for our analysis:

- Number of peers: 100
- Percentage of slow peers: 50%
- Percentage of low CPU peers: 50%
- Mean transaction interarrival time: 10 seconds
- Mean block arrival time: 600 seconds

Some general inferences:

- The ratio is higher for fast nodes as compared to the slow nodes. Because, across a link with both fast nodes, the latency is less, and hence the peer nodes are able to communicate their blocks faster. (The blocks which come later are invalid, and hence the ratio is higher for fast nodes).
- The ratio is lesser in case of high CPU nodes. Because of their higher hashing power, they complete their mining before the blockchain gets stabilized, and thus produce forks, and a decreased ratio.

In the graph shown below, we vary one of these parameters and deduce inferences from the graph.

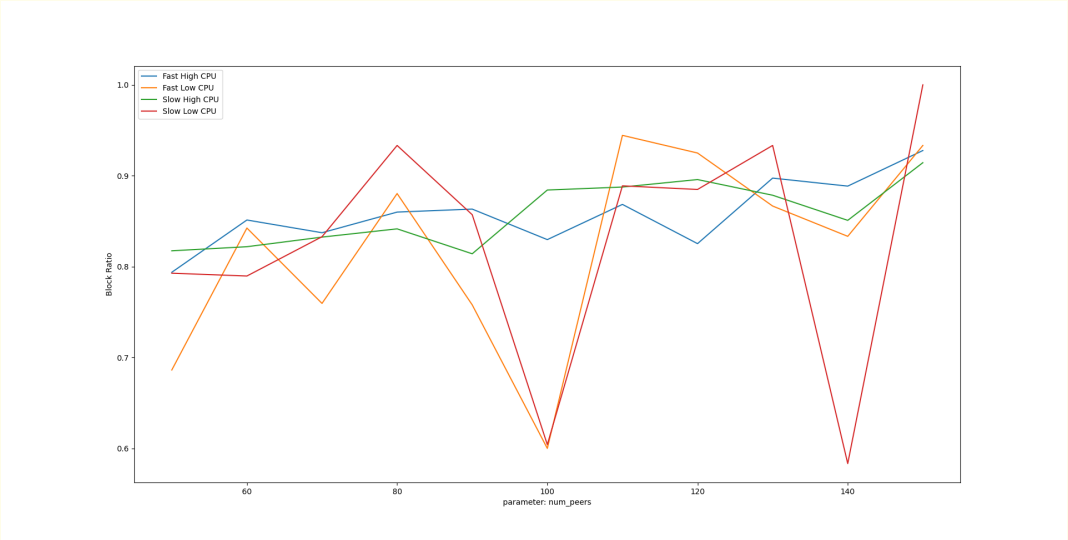


Figure 2: Ratio v/s Number of peers

The above is the graph of the ratio across various types of peer nodes vs the number of nodes in the network. The ratio is in general higher for fast nodes as compared to the slow nodes, because across a link where both the nodes are fast nodes, the latency is quite less, and hence the peer nodes are able to communicate their blocks faster. Rest, there are some outliers in the graph too.

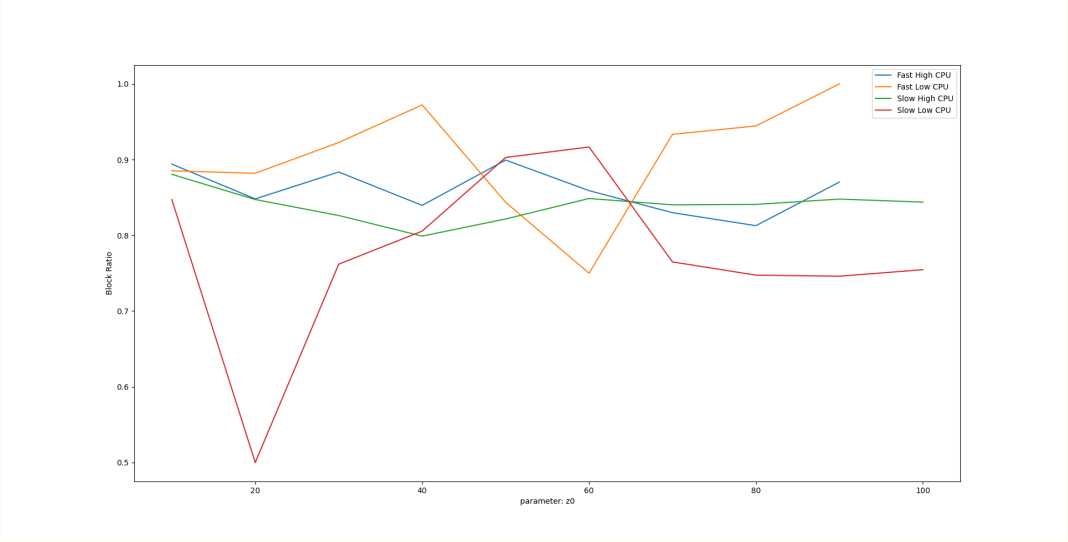


Figure 3: Ratio v/s Percentage of Slow Peers

The above graph is the ratio vs the percentage of slow nodes in the graph. The ratio is again higher for fast nodes. Overall, there is neither an upward nor a downward trend in the graph, which shows that the percentage of slow nodes does not affect the ratio much (for a link to have lower latency, both the nodes need to be fast).

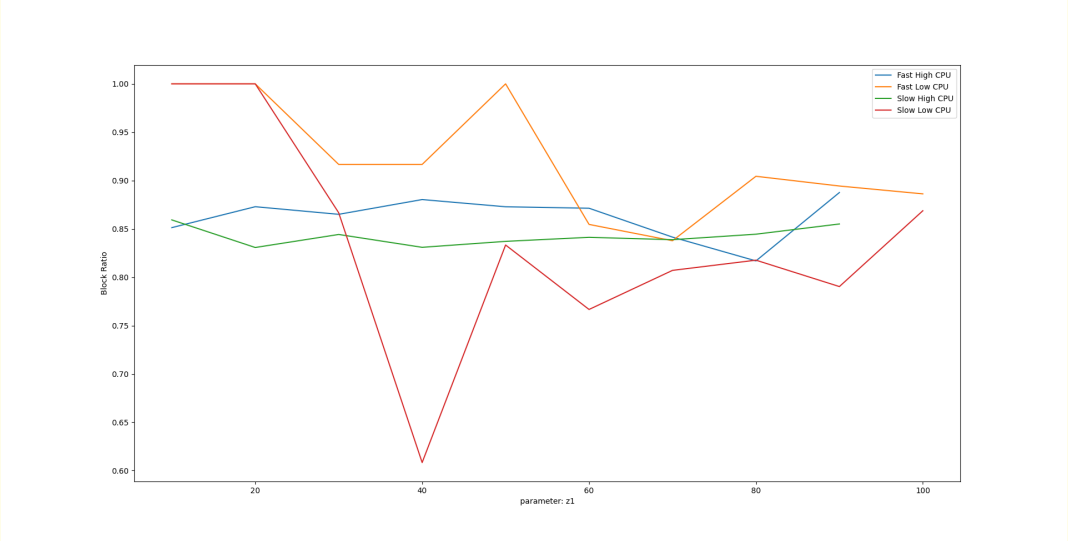


Figure 4: Ratio v/s Percentage of Low CPU Peers

The above graph is the ratio vs the percentage of low CPU nodes in the graph. The ratio is similar across various types of nodes, which shows us that the network which we have created gives proper chance to all types of nodes. However, the number of blocks produced by the fast CPU nodes is higher, and because of similar ratios, the number of blocks in the blockchain is also higher for fast CPU nodes.

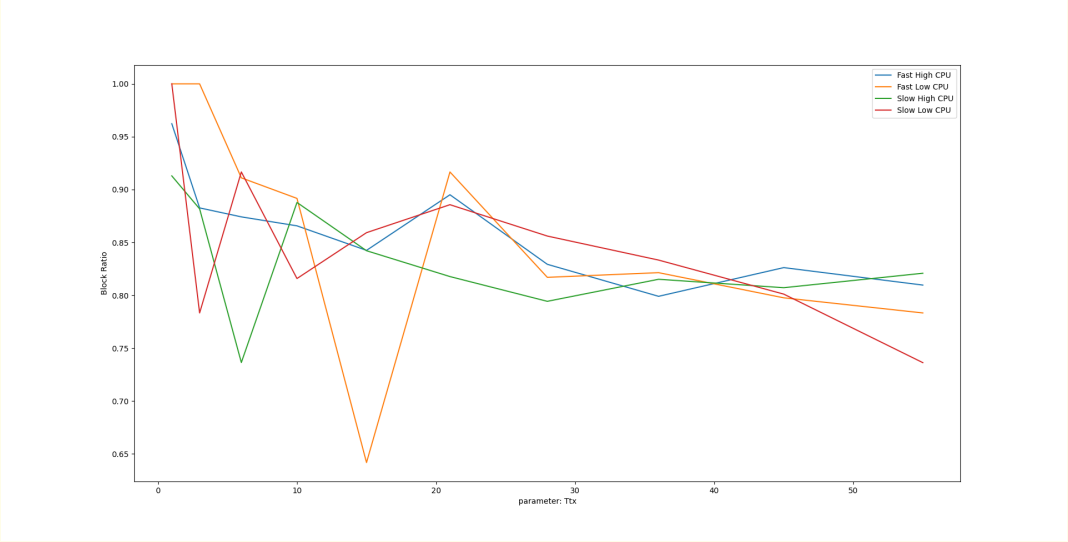


Figure 5: Ratio v/s Mean Transaction Interarrival Time

The above graph is the ratio vs the mean transaction interarrival time. Overall, the ratio shows a downwards trend as the mean transaction interarrival time increases. This is because the number of transactions produced in the network reduces, as the mean inter arrival time increases. So, the amount of overlap between various blocks (in terms of the transaction sets they contain) increases, and a larger fraction of blocks are declared to be invalid.

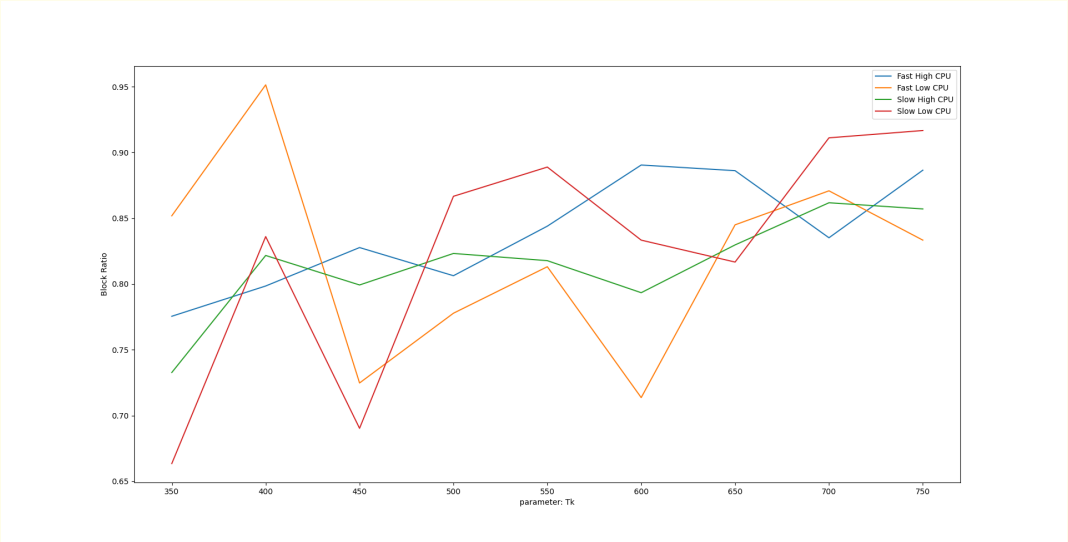


Figure 6: Ratio v/s Mean Block Arrival Time

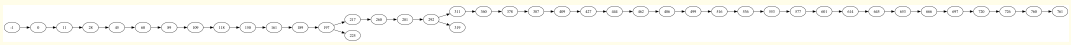
The above graph is the ratio vs the mean block arrival time. Overall, there is an upwards trend in the graph as the mean block arrival time increases. This is because the number of blocks produced in the network reduces, as the mean block arrival time increases. So, the amount of overlap between various blocks (in terms of the transaction sets they contain) decreases, and a smaller fraction of blocks are declared to be invalid.

Some sample blockchain trees

We will show the blockchain trees for the following set of parameters:

- Number of peers: 20 ; Percentage of slow peers: 50% ; Percentage of low CPU peers: 50%

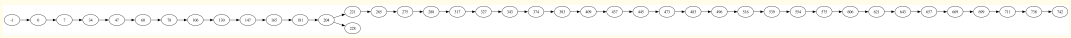
; Mean transaction interarrival time: 10 seconds ; Mean block arrival time: 600 seconds ; Time of execution: 10 seconds



Here are the branch heights for the above tree:

```
Branch Heights:- 33 42 42 48 50 52 64 64 66 93 104 106 106 107 108 110 110
```

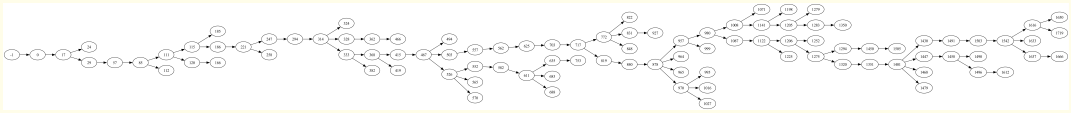
- Number of peers: 20 ; Percentage of slow peers: 50% ; Percentage of low CPU peers: 50% ; Mean transaction interarrival time: 10 seconds ; Mean block arrival time: 6000 seconds ; Time of execution: 10 seconds



Here are the branch heights for the above tree:

```
Branch Heights:- 32
```

- Number of peers: 20 ; Percentage of slow peers: 50% ; Percentage of low CPU peers: 50% ; Mean transaction interarrival time: 10 seconds ; Mean block arrival time: 6 seconds ; Time of execution: 10 seconds



Here are the branch heights for the above tree:

```
Branch Heights:- 6 7 7 7 9 10 11 12 13 14 15 17 17 18 19 20 21 21 21 22 23 24
```

As we can see that on varying the inter arrival block time, the number of branches in the tree also varies. For a very low block arrival time, the tree is very dense and has a lot of branches. For a very high block arrival time, the tree is very sparse and has very few branches.

Class Structure

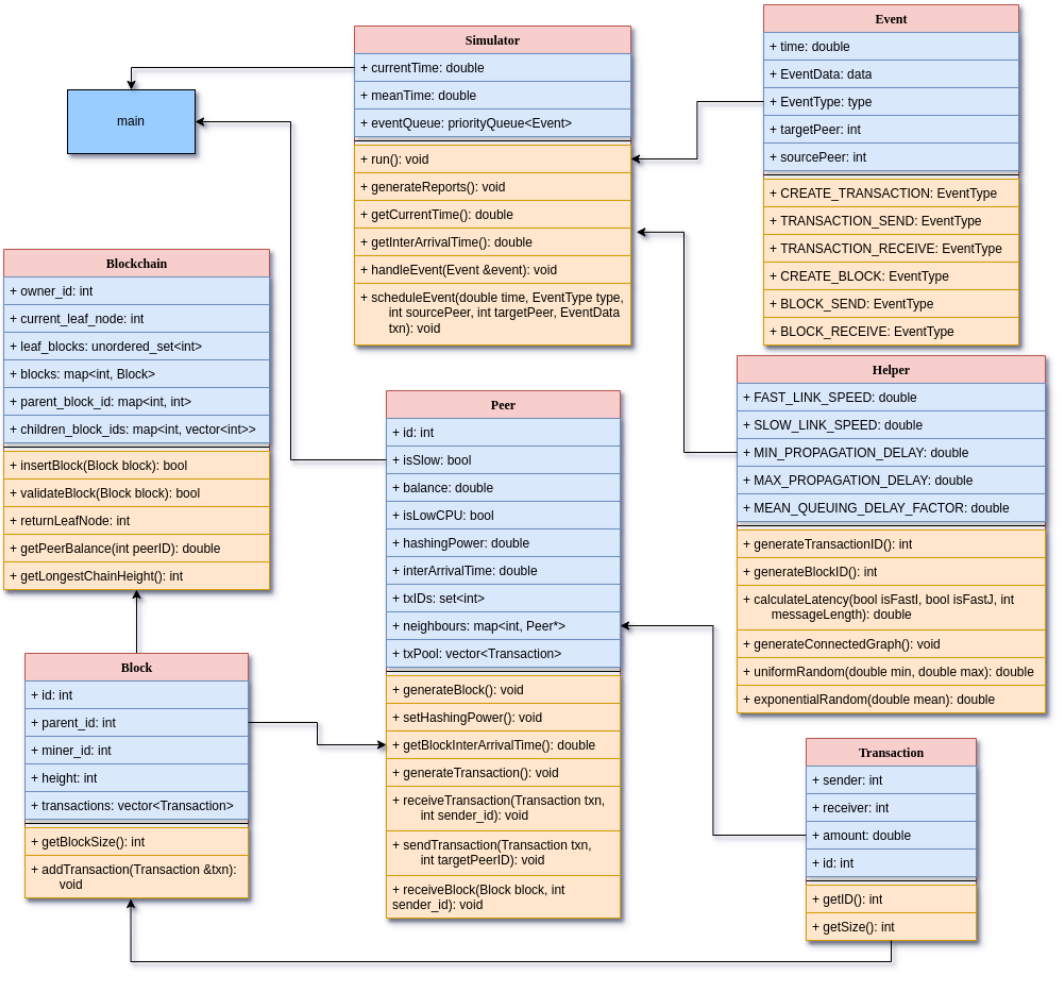


Figure 7: Ratio v/s Mean Block Arrival Time

Simulator

- The simulator calls the `run` function to start the simulation.
- It first calls the `generateConnectedGraph` function to create a connected graph of nodes.
- Each node then creates a genesis block using the `createGenesisBlock` function.
- The simulator schedules the creation of transactions by calling `scheduleEvent` with an inter-arrival time, event type, and node index.

```
void Simulator::run() {
 generateConnectedGraph();

 for (int i = 0 ; i < num_nodes ; i++ ) {
  peers[i]->createGenesisBlock();
 }

 for (int i = 0 ; i < num_nodes ; i ++ ) {
  double interArrivalTime = getInterArrivalTime();
  scheduleEvent(interArrivalTime, CREATE_TRANSACTION, i, -1, {});
 }
}
```

- A random peer is selected to start mining by scheduling a `MINING_START` event.
- The simulator processes events from the event queue by updating the current time and handling each event accordingly.

```
 int random_peer = uniformRandom(0, num_nodes - 1);
 scheduleEvent(currentTime, MINING_START, random_peer, -1, {});
 while ( !eventQueue.empty() ) {
  Event current = eventQueue.top();
  eventQueue.pop();
  currentTime = current.time;
  handleEvent(current);
 }
```

Transactions

A peer creates a transaction by calling the `generateTransaction` function.

- The peer checks its balance using the `getPeerBalance` function.
- If the balance is greater than zero, it selects a random target peer.
- A transaction is created and added to the transaction pool.
- The transaction is then broadcasted to all neighboring peers.

```
void Peer::generateTransaction() {
 double our_balance = blockchain->getPeerBalance(id);
 if (our_balance <= 0) {
  return;
 }

 int targetPeerID = id;
 while (targetPeerID == id) {
  targetPeerID = uniformRandom(0, num_nodes - 1);
 }
 Transaction txn = Transaction(id, targetPeerID,uniformRandom(1, our_balance));
 txPool.insert(txn);
 for (auto& [id, peer] : neighbours) {
  sendTransaction(txn, peer->id);
 }
}
```

After the simulator first schedules the event `CREATE_TRANSACTION` at each node:

- Each node generates a transaction and then a new time (`newTime`) is calculated by adding the current time to a value obtained from an exponential distribution using the function `getInterArrivalTime()`.
- The event `CREATE_TRANSACTION` is then recursively scheduled on the same node with the newly calculated `newTime`.

```
case CREATE_TRANSACTION:
 peers[event.sourcePeer]->generateTransaction();
 newTime = currentTime + getInterArrivalTime();
 scheduleEvent(newTime, CREATE_TRANSACTION, event.sourcePeer, -1, event.data);
 break;
```