Blockchain Assignment 1

Atharva Tambat - 210070014
Shubham Hazra - 210100143

February 2023

Contents

1 Introduction

This aim of this assignment is to build a discrete-event simulator for a P2P cryptocurrency network. We do this by using a centralized event priority queue which stores 4 different types of events sorted in ascending order of their execution times. A graph of a given number of nodes is created to model the network. The graph is created such that each node is connected to atleast 4 and atmost 8 other nodes. All the nodes have been given different attributes like cpu power, speed, hashing capabilities etc. to model them as peers. The latencies between any two connected nodes is also modelled by using an appropriate formula. All the peers act independently and mine on a local copy of their own blockchains. They perform 4 main functions corresponding to the 4 different events in our event queue i.e. they create and forward transactions and mine and forward blocks. The simulator is initialized by seeding a few transaction generation events and a few mine block events at the start after which they are generated automatically. For more precise implementation details check the Design section of the report below.
NOTE: The block numbers of the consecutive blocks differ by $\sim$ 100 in the following diagrams. This is due to the algorithm that is used to number the blocks. The blocks are numbered by a node when it starts mining on the block irrespective of whether it would be successful in that. Since, in a "round" of mining, only about a couple of 100s of blocks will be successful in solving PoW, the numbering of the consecutive blocks are not almost consecutive.
The problem statement only states that the Block IDs must be distinct (not consecutive), which is still satisfied.

2 Flow of Code

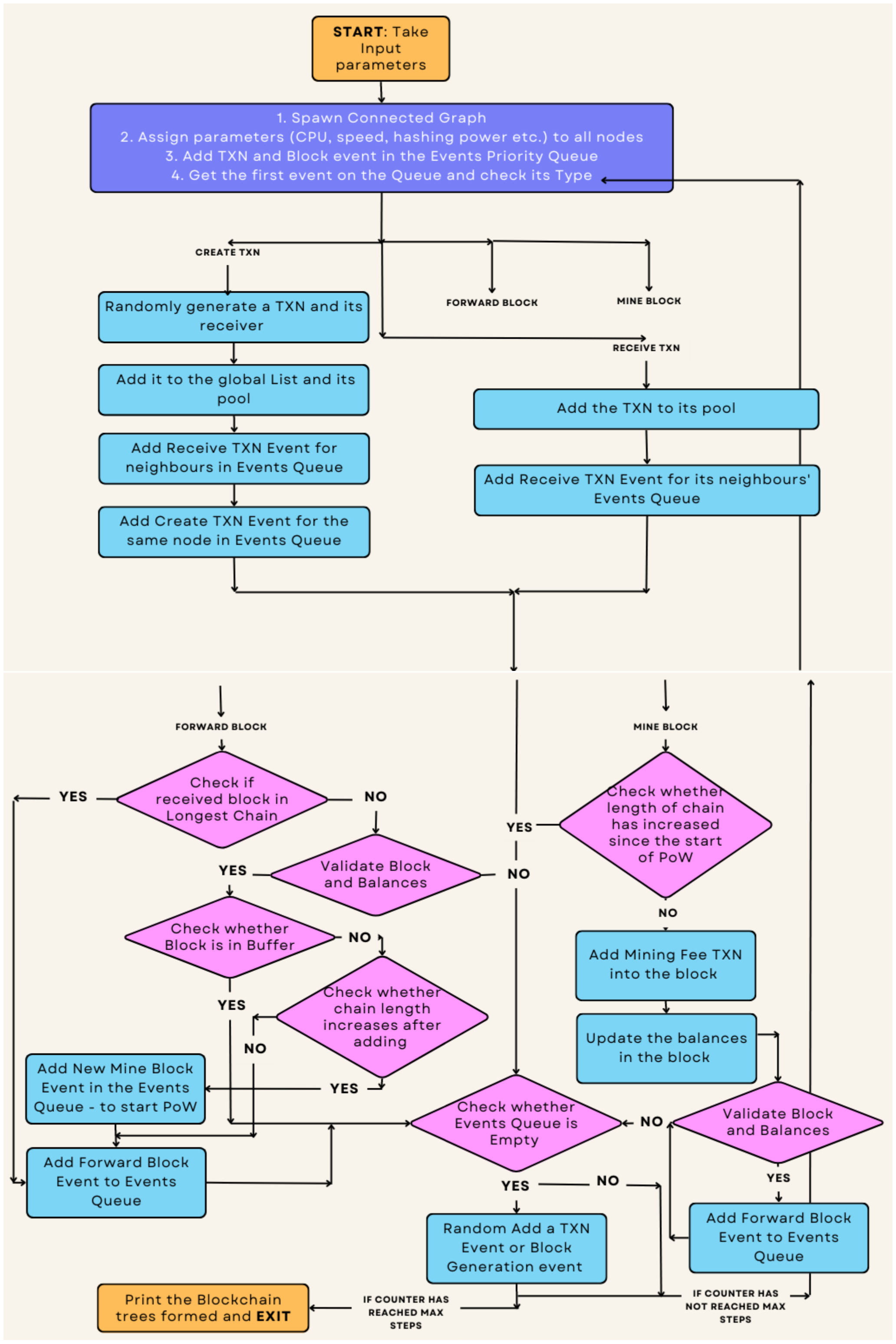

START: Take Input parameters
1. Spawn Connected Graph
2. Assign parameters (CPU, speed, hashing power etc.) to all nodes
3. Add TXN and Block event in the Events Priority Queue
4. Get the first event on the Queue and check its Type
CREATE TXN
FORWARD BLOCK
MINE BLOCK
RECEIVE TXN
Randomly generate a TXN and its receiver
Add it to the global List and its pool
Add Receive TXN Event for neighbours in Events Queue
Add Create TXN Event for the same node in Events Queue
Add the TXN to its pool
Add Receive TXN Event for its neighbours' Events Queue
FORWARD BLOCK
MINE BLOCK
Check if received block in Longest Chain
YES
NO
Validate Block and Balances
YES
NO
Check whether Block is in Buffer
NO
YES
Check whether chain length increases after adding
NO
YES
Check whether length of chain has increased since the start of PoW
YES
NO
Add Mining Fee TXN into the block
Update the balances in the block
Validate Block and Balances
YES
NO
Add New Mine Block Event in the Events Queue - to start PoW
Add Forward Block Event to Events Queue
Check whether Events Queue is Empty
YES
NO
Random Add a TXN Event or Block Generation event
Add Forward Block Event to Events Queue
Print the Blockchain trees formed and EXIT
IF COUNTER HAS REACHED MAX STEPS
IF COUNTER HAS NOT REACHED MAX STEPS

Figure 1: Program Logic Flow Chart

3 Problem Statement Answers

Q. What are the theoretical reasons of choosing the exponential distribution?

The theoretical reasons for choosing exponential distribution for the inter-arrival time between transactions generated by any peer are:

- All the miners have to solve a puzzle (bring the hash of block header below the threshold). Since the hash function used (SHA-256) is Hiding and Puzzle-Friendly solving it is only possible by using random values. Since the exponential function, models the inter-arrival time of random processes, it is the most suitable
- Due to virtue of the exponential distribution's memory less property, transactions generated do not depend on the history of transactions and hence each peer acts independently to create transactions
- Inter-arrival time being sampled from an exponential distribution implies that the transaction generation process forms a **Poisson distribution** which is a very good approximation of the real bitcoin blockchain

Q. Why is the mean of d_{ij} inversely related to c_{ij} ? Give justification for this choice.

The mean of d_{ij} is inversely related to c_{ij} because:

- d_{ij} is the queuing delay and it depends on the link speed (c_{ij}) as link speed determines the number of packets to be sent per second and the number of seconds a packet is not sent is its queuing delay.
- For the same number of packets higher link speeds will pump out more packets per second reducing the queuing delay for other packets and lower link speeds pump out packets slowly increasing the queuing delay for other packets. Hence d_{ij} is inversely related to c_{ij}

Q. Explanation for the choice of a particular mean of T_k.

The choice of mean inter-arrival block time is found to be dependent on the following two factors:

- Network Delays: Changing the Inter arrival time of the blocks (since the network delays are fixed), it was found that for block inter arrival delay $<$ 5 seconds, forking begins and grows when the network delays become a significant number as compared to the mean block inter-arrival time
- Mean Inter-TXN generating time: Since we have initially spawned a TXN for every node, to ensure that there are enough TXNs with every node to make a block, so that the simulation does not stop pre-maturely, the final structure of the blockchain is somewhat less dependant on the mean inter-TXN generating time

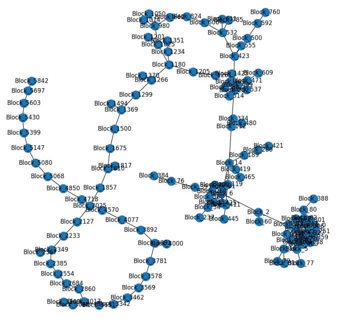
As far as the expressing is concerned, it is taken from an exponential distribution with mean $\frac{I}{h_i}$ (I is a parameter and h_i is the hashing power of the block). This is justified because, the more the hashing power h_i less should the time that the node has to wait for the completing PoW.

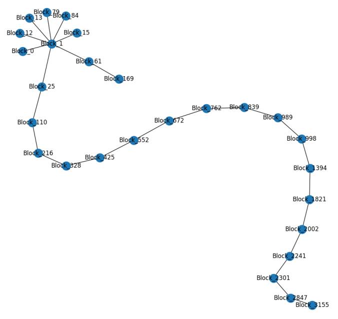
4 Observations

We have experimented with a network of 100 nodes, with varying z_0, z_1, T_{tx} and I. The following are our observations:

1. **Changing the average block inter-arrival time (I)**: To measure the effect of I, we kept the $z_0 = 10$, $z_1 = 10$, $T_{tx} = 0.1$ and ran the simulation for 50000 steps. The following immediate observations can be made from the graphs:

   - **Length of blockchain:** As the average inter-arrival time of the blocks increases, since the simulation is run for a fixed number of steps, the number of blocks in the longest chain of the final blockchain decreases, as less number of blocks are generated on an average
   - **Amount of forking in the blockchain:** As I increases, the amount of forking in the blockchain also decreases. When I is comparable to the network delays, there is a higher possibility of two blocks being mined at the same time, leading to forks the in blockchain
   - **Length of forks:** It can be observed that as I decreases, the length of forks also increases. This is because if blocks are coming at a high rate, then higher is the chance that a fork once created, both the individual chains of the block are simultaneously elongated (so that one remains the longest chain for some fraction of nodes and same for the other)


(a) Blockchain for I = 0.001


(b) Blockchain for I = 0.05

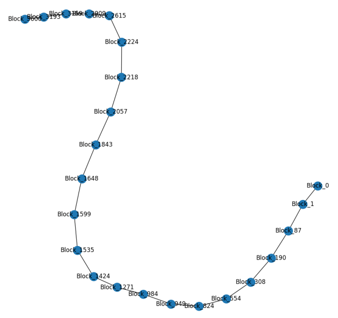
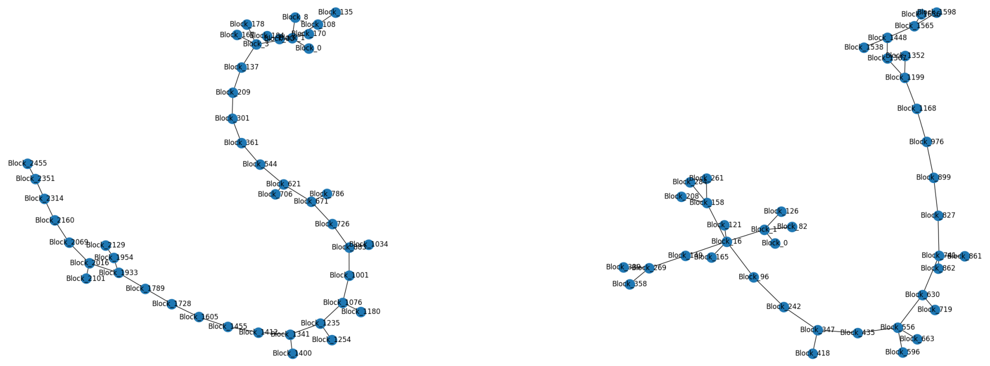

(c) Blockchain for I = 20

Figure 2: Blockchains of a particular node for different I values

2. **Changing the percentage of nodes having low speed (z_0)**: To measure the effect of z_0, we kept the $z_1 = 10$, $T_{tx} = 100$, $I = 10$ and ran the simulation for 10000 steps. The following immediate observations can be made from the graphs:

   - **Generation Ratio ($r_{z_0=z_0'}$):** defined as - 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. So, the ration $r_{z_0=0} \sim \frac{32}{45}$ and $r_{z_0=100} \sim \frac{20}{42}$. So, $r_{z_0=0} > r_{z_0=100}$. Lower r means that larger fraction of blocks have been orphaned. This is expected because when larger fraction of nodes are slow, then the network delay between them increases, which causes more forking (as explained above)
   - **Length of the blockchain:** Since the propagation delay of $z_0 = 0$ case is less than the $z_0 = 100$ case, and a larger fraction of blocks get into the main chain (as shown above), the length of main chain is on an average greater when number of slow nodes are less - less propagation delay, means that all the blockchain get updated in a time < average PoW delay (Tk)
   - **Amount of forking in the blockchain & Length of forks:** Both the amount of forking and length of the largest fork is greater when a greater proportion of nodes are slow (as explained above in the generation ratio part)



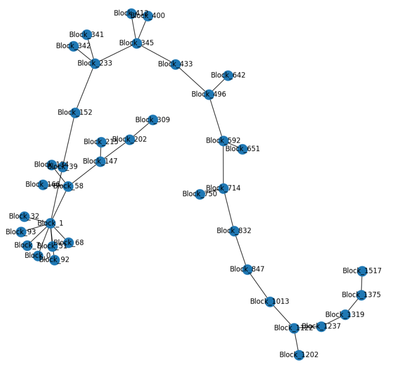
(a) Blockchain for $z_0 = 0$ (b) Blockchain for $z_0 = 100$

Figure 3: Blockchain of a node with high and low speed

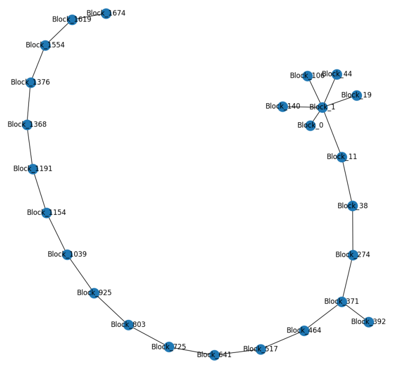
3. **Changing the percentage of nodes having low cpu (z_1)**: To measure the effect of z_1, we kept the $z_0 = 10$, $T_{tx} = 100$, $I = 10$ and ran the simulation for 10000 steps. The following immediate observations can be made from the graphs:

   - **Generation Ratio ($r_{z_1=z_1'}$):** defined as above. So, the ration $r_{z_1=0} \sim \frac{18}{39}$ and $r_{z_1=100} \sim \frac{20}{24}$. So, $r_{z_1=0} < r_{z_1=100}$. Lower r means that larger fraction of blocks have been orphaned. This is expected because when larger fraction of nodes are low cpu, the mean inter-block arrival delay ($T_k = \frac{I}{h_i}$ and $h_i \propto$ CPU power) is high, so, the chances of two blocks getting created almost at the same time is less.

- **Length of the blockchain:** It was observed, lower the proportion of nodes with low CPU, smaller is the blockchain. The reason for this might be that even if larger number of blocks are being produced in the case where majority of nodes are faster, a large proportion of them are orphaned - not contributing to the length of longest chain
- **Amount of forking in the blockchain & Length of forks:** Both the amount of forking and length of the largest fork is greater when a lesser proportion of nodes have low CPU (as explained above in the generation ratio part)



(a) Blockchain for $z_1 = 0$



(b) Blockchain for $z_1 = 100$

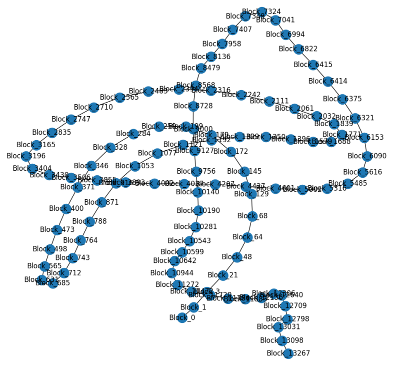
Figure 4: Blockchain of a node for high and low CPU values

4. **Changing the total number of nodes in the graph (n):** To measure the effect of n, we kept the z_0= 10, $z_1 = 10$, $T_{tx} = 100$, $I = 10$ and ran the simulation for 50000 steps. (NOTE: The inter-block arrival time is enough to cause no forking) The following immediate observations can be made from the graphs:

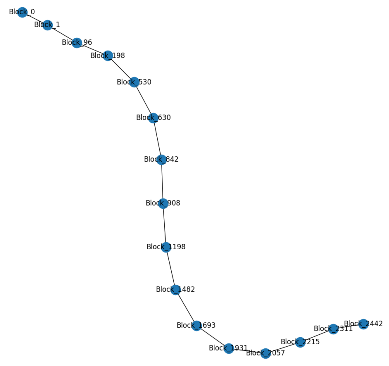
   - **Length of the blockchain:** Since the smaller graph has smaller number of nodes:

     (a) Lesser time (here, number of steps) is required to propogate both TXNs and newly created block. Thus, more number of steps are left for more TXN and block creation events

   - **Amount of forking in the blockchain & Length of forks:** The inter-block arrival time is enough to cause no forking



(a) Blockchain for n = 15

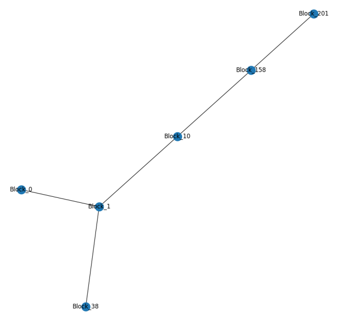


(b) Blockchain for n = 100

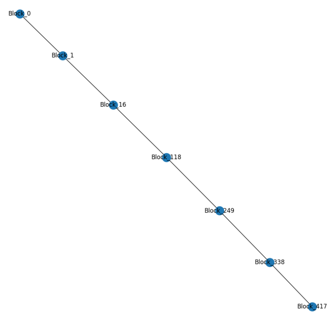
Figure 5: Blockchain of a node for different sizes of network

5. **Changing the mean inter TXN generation time (T_{tx})**: To measure the effect of T_{tx}, we kept the $z_0 = 10$, $z_1 = 10$, $I = 10$ and ran the simulation for 10000 steps. The following immediate observations can be made from the graphs:

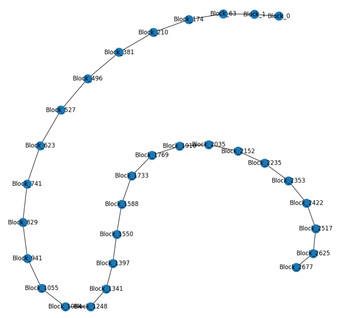
   - **Length of the blockchain:** The change observed was not linear/ monotonic. After the initial round of TXNs get over (spawned at the start), inter-TXN arrival time plays a role.
     (a) Very small T_{tx} (0.001): TXN generation events flood the Event Queue and since the simulation runs for finite steps, less Block Creation events are executed
     (b) Intermediate T_{tx} (1): More number of blocks, but no significant increase
     (c) Intermediate T_{tx} (10000): Sweet point - balance between TXN creation time and Block creation time
     (d) Intermediate T_{tx} (100000): All the new TXN generation events are so delayed that when new blocks are formed, TXN pool of many miners are empty. Therefore less number of blocks again.
   - **Amount of forking in the blockchain & Length of forks:** No significant difference was found in the blockchains in this respect, which is expected again, because generation of TXNs is independent (to a large extent, when enough are available) of Block creation and propogation.



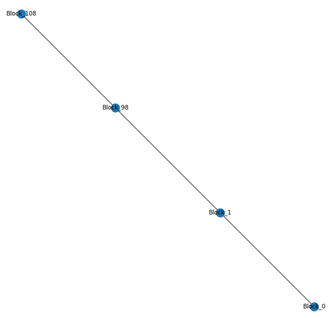
(a) Blockchain for T_{tx} = 0.001



(b) Blockchain for T_{tx} = 100



(c) Blockchain for T_{tx} = 10000



(d) Blockchain for T_{tx} = 100000

Figure 6: Blockchain for different inter-TXN creation time

5 Design

5.1 Network

The network is created using the **Networkx** package in python. The network is created as follows:

- First a random degree sequence is generated with each of the degrees being in between 4 and 8
- Then this sequence is sorted and edges are added such that the degrees of the nodes correspond to this sequence
- Then the edges are swapped for a generous number of times to simulate randomness while maintaining the connected-ness of the graph

After the graph (network) has been formed, we add a set of attributes to each of the nodes (peers) in the network. The attributes added are:

- cpu (high or low) - Set a randomly chosen z_1% of the nodes to low cpu and others to high. Where z is a command line argument
- speed (high or low) - Set a randomly chosen z_0% of the nodes to low speed and others to high. Where z is a command line argument

- l (latency in seconds) - Chosen from a uniform distribution between 10ms and 500ms for each node
- c (link speed in Mbps) - Set to 5 if node is low cpu else set to 100
- d (queuing delay) - It is randomly chosen from an exponential distribution with some mean 96kbits/c
- hashing_power (hashing power in fraction) - Set to a number h_k for node k. Where $h_{high\_cpu} = 10 * h_{low\_cpu}$ and $\sum_k h_k = 1$

5.2 Blocks

The block is simulated by a python class with the following attributes:

- block_id - Unique ID for each block (Set to 0 for genesis block)
- creator_id - The ID of the creator of the block
- previous_id - The ID of the previous block it points to
- created_at - The time of creation of the block
- transactions - List of all transactions (IDs) present in the block
- length - Distance from the genesis block in a chain
- received - A boolean array keeping track of the peers who have seen this block

The size of a block depends on the number of transactions in it and is given by:

$$\text{Size of block} = \text{Size of coinbase} + \text{Total size of transactions in the block}$$

- The size of the coinbase is assumed to be equal to 1KB (=8000 bits)
- The size of a block can not exceed 1MB (= 1000 KBs)

5.3 Transactions

The transactions are generated at intervals of time sampled from an exponential distribution with some mean T_k which is a command line parameter. The transactions are created at each node at the start of the simulation with a randomly chosen node as the receiver. The amount of coins in the transaction is between 0 and 1.2 times the amount of coins the sender has. The range is chosen as such so that most of the transaction will be valid while also creating some invalid transactions.
The transactions are modelled by a python class with the following attributes:

- txn_id - Unique transaction id (Number) (Negative if it is a coinbase transaction)
- id - Unique transaction id (Of the form Txn_$\langle$ txn_id $\rangle$)

- sender_id - Unique id of the sender
- receiver_id - Unique id of the receiver
- amount - Amount of coins in the transaction
- message - A message of the form: "TxnID: ID_x pays ID_y C coins" or of the form "TxnID:ID_k mines C coin"

The size of a single transaction has been assumed to be equal to 1KB (=8000 bits)
The transactions are propagated throughout the network and each peer receives the transaction with a delay corresponding to the sum of latencies in the path the transaction took to reach it.
The latency between two peers is simulated as:

$$\text{latency}_{ij} = \rho_{ij} + |m|/c_{ij} + d_{ij} \text{ where,}$$

- ρ_{ij} - A positive minimum value corresponding to speed of light propagation delay
- m - The length of the message in bits
- c_{ij} - The link speed between i and j in bits per second
- d_{ij} - The queuing delay at node i to forward the message to node j

5.4 Nodes

A node in the network represents a peer and these peers are assumed to act independently to generate transactions and mine blocks. Each node maintains its own blockchain tree and mines on its longest chain. The nodes also listen for transactions and are responsible for forwarding it to their neighbours.
Upon receiving a block it first validates the transactions in the block and only after validation it adds it to the chain. If while mining a block it receives another block and the longest chain changes then the block it was mining is dropped.
The node is a python class with the following attributes:

- pid - Unique Id of the peer
- cpu - CPU speed of the peer
- hashing_power - Hashing power of the peer
- speed - Speed of the peer
- peers = Storing the pointer for function to put events in Queues of peers
- BTC = Initial balance of the peer
- blockchain_tree = Blockchain tree of the peer
- blockchain = Blockchain of the peer - stores the block objects, initially the genesis block is added

- longest_chain =Longest chain of the peer as a list of block ids
- max_len = Length of the longest chain
- txn_list = List of transactions that the peer has seen but not included in any block
- included_txn = List of transactions that the peer has included in a block
- block_buffer = List of blocks that the peer has heard but not added to its blockchain because parent block is not yet added
- blocksReceiveTime = List of time at which the peer receives a block

5.5 Events

Events are the most important component of the simulation. These are the objects that are placed in the simulation queue and are responsible for all the transactions and block creations.
The **Event** class in python acts as a base class for the different types of events and has the following attributes:

- node_id - Node on which the event is to be run
- creator_id - Node which created the event
- create_time - Time at which the event was created
- run_time - Time at which the event is to be run

The events considered in this simulation are of 4 broad types:

- CreateTXN - This event is responsible for creating a transaction with a random amount and to a random receiver. This event when executed seeds a ReceiveTXN and another CreateTXN with appropriate delays
- ReceiveTXN - This event is responsible for listening to a transaction and then forwarding it to its neighbors. This event when executed seeds another ReceiveTXN. It is only called for those peers that have not seen the transaction yet so as to not form an endless loop
- MineBlock - This event is responsible for mining a block at a node with a random set of transactions included and pointing to its longest chain. This event when executed seeds another MineBlock and ForwardBlock event
- ForwardBlock - This event is responsible for listening for a block and then forwarding it to its neighbors.The node first validates the block and forwards the block to its neighbors only if the block is valid. This event when executed seeds another ForwardBlock

5.6 Simulator

The simulator is the main component which makes the priority queue and executes the different events for each time setp. It takes in all the command line arguments and runs the simulation for a given number of steps.
It initializes the event queue with a few number of transactions using CreateTXN event and a few block mining events using MineBlock, after that the simulation is automatic. The simulator is a python class with the following attributes:

- N - Network of n nodes
- z0 - Percentage of slow nodes
- z1 - Percentage of low CPU nodes
- Ttx - Mean transaction interarrival time
- I - Mean block interarrival time
- txn_id - Unique ID for each transaction
- block_id - Unique ID for each block
- mining_txn_id - Unique ID for each mining transaction
- events - Priority Queue to store the events
- global_transactions - To store the TXN objet indexed by the unique ID of the TXN