CS765 Spring 2023 Semester
Project Part-1
Simulation of a P2P Cryptocurrency Network

Harsh Shah (200050049)
Sahil Mahajan (200050124)
Shubham Bakare (200050133)

Contents

1 Introduction

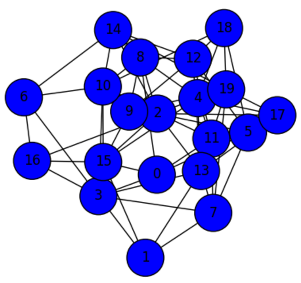
We have created a discrete-event simulation for a P2P cryptocurrency network in this assignment. This simulation represents various aspects of a decentralized network, such as block creation, validation, mining, and propagation, as well as the inter-arrival time between transactions. The simulation is visualized by changing various parameters and studying their effect on blockchain tree.

2 Transaction Generation

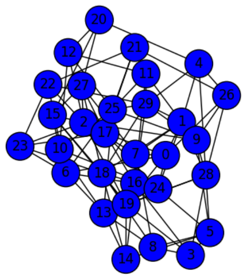
The transactions are generated by peers with an inter-arrival time chosen from an exponential distribution. The reason for choosing exponential distribution is that the transaction arrival is memory-less and the probability of next transaction to arrive is assumed to be independent of the previous transaction. The mean time parameter (T_{tx}) can be interpreted as the expected time between two consecutive transactions which controls the simulation overall transaction rate.

3 Peer Network

In our simulation, every peer is connected to 4 to 8 other peers chosen randomly such that they create a connected network and every peer has some path through the network to another peer. We have used the igraph python library to implement the graph. Details of the graph creation are in the graph.py code. A sample network graph of such connection is as below:



(a) n = 20



(b) n = 30

4 Network Delay

In the simulation, the queuing delay is chosen from an exponential distribution with mean inversely proportional to the link speed between the two peers. The queuing delay is time a transaction stays in queue until it is added to block. Link speed between two peers is the maximum speed with which data can move across the link. When a transaction is being sent through a link, another transaction gets added to queue and has to wait till this transaction has reached other peer. Thus for slower link speed of the network, transactions will have to wait longer and the queuing time for transactions will increase. For high link speed, the queuing time will be relatively lower. Hence queuing delay is inversely related to link speed.

5 Blockchain Implementation

5.1 Block Structure

The Block Class stores the hash of previous block, block size, transaction list, amount list of all nodes, time created and the node which created the block. Each block has a unique ID, generated by taking hash of concatenated string of previous block's hash, this block's transactions and the generation time of block

5.2 Block Creation and Validation

Every node has a genesis block at the start. A transaction is added to the block only if it is validated and satisfies the blockchain rules. After updating the amount list after transaction, no value in the amount list should get negative. The number of transactions contained in a block is chosen at random from an uniform distribution, capped on maximum block size. After block is created and validated, it is sent by the node to its neighbours.

5.3 Block Propagation

On receiving a block following steps are followed:

- First the transactions of the block are validated
- Check the parent of the block by checking with the hash values in the current tree of blockchain
- If the block is added to the main chain head, no fork resolution is required
- Similar is the case when the added block does not exceed height of main chain (when block is added to a stale branch)
- In case the height increases then fork resolution is required and the transactions of the main chain are added back to the transaction pool while the transactions of previously stale branch are used to modify the amount list- amount of money with each node
- Then this block is sent to neighbours while ensuring loop-less transfer

5.4 Loop-Less Transfer

Loop-less transfer of transactions and blocks through the network is ensured by: on receiving/creating a block, it is sent to all the neighbors from whom we have not received that block. We are list of all transactions/blocks that every nodes forwards to its peers and we make sure to check any incoming transaction/block with the list and in case it has already been forwarded to any peer we don't forward it to that perticular peer hence avoiding repeated transfer and ensuring loop-less transfer.

5.5 Hashing Power and T_k

Peers have a fraction hashing power (h_k) which indicates the fraction of total hashing power they have. The more the hashing power, the lesser the computation time and the faster the ability to make a block. Hence expected time is inversely proportional to the fraction of hashing power. The average inter-arrival time between blocks is parameter I. To model the time between blocks, the node generates a random variable T_k by drawing from an exponential distribution with a mean equal to I/h_k. This mean was chosen because it represents the expected time between blocks for that node taking into account its proportion of the total hashing power.

6 Parameters

6.1 Command Line Parameters

- n: Number of nodes in the network
- z_0: Fraction of total nodes which are slow (remaining are fast nodes)
- z_1: Fraction of total nodes which are low CPU (remaining are high CPU)
- simtime: Simulation time for which the simulator should run

6.2 Other Parameters

- ρ_{ij}: Positive minimum value corresponding to speed of light propagation delay
- c_{ij}: Link speed between two nodes, set to 5Mbps for slow and 100Mbps for fast
- I: Average interarrival time between blocks
- T_{tx}: Mean time of inter-arrival between transactions generated by a node

7 Design Document

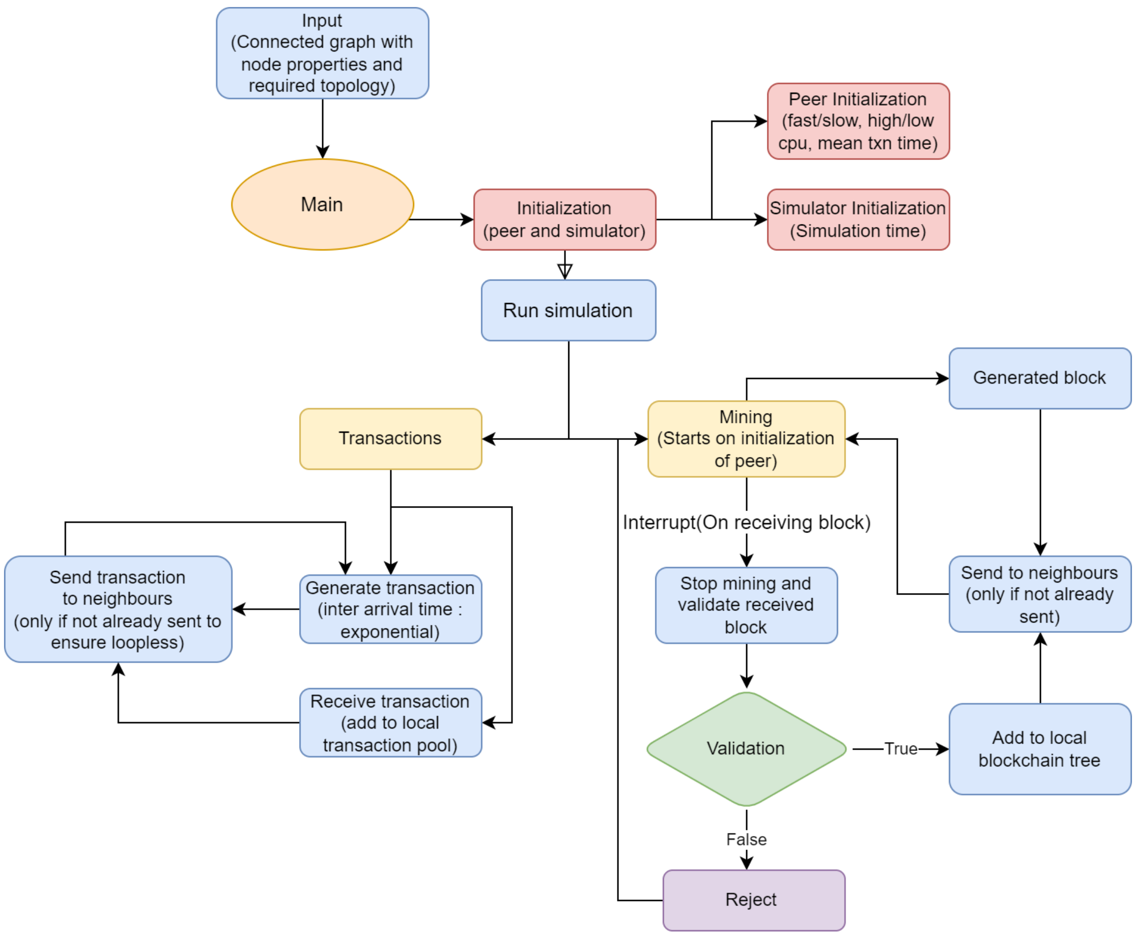

Input (Connected graph with node properties and required topology)
Main
Initialization (peer and simulator)
Peer Initialization (fast/slow, high/low cpu, mean txn time)
Simulator Initialization (Simulation time)
Run simulation
Transactions
Mining (Starts on initialization of peer)
Generated block
Send transaction to neighbours (only if not already sent to ensure loopless)
Generate transaction (inter arrival time : exponential)
Receive transaction (add to local transaction pool)
Interrupt(On receiving block)
Stop mining and validate received block
Send to neighbours (only if not already sent)
Validation
True
False
Add to local blockchain tree
Reject

Figure 2: Design Document

7.1 Code implementation

We have used the following libraries/packages for implementation of required simulation in code:

- Simpy : For discrete-event simulation
- igraph : For generating and visualizing graphs
- matplotlib : For visualization of graph topology
- xdot : For visualizing blockchain trees

7.2 Code walkthrough

Following is a brief overview of the code(a represented in the design document):

- main.py initializes the graph of required topology and inputs it to simulator which in turn initializes the peers.
- The peers repeatedly generate the transactions and block.
- Block mining is interrupted when another block is received.
- If the received block is valid, then the peer adds to its local chain(after fork resolution) and sends to neighbours, while ensuring loopless transfer.
- In case a peer mines a block, it updates its amount list and then sends to its neighbours.
- Any delay is simulated using timeout functionality in simpy, which then adds event to its priority heap based on the amount of timeout.
- Sending of any data(transaction or block) is simulated by considering each transfer as a separate event.

8 Visualization

We have used xdot and matplotlib for visualizing the blockchain tree created by each node.
The Trees are represented in the figures such that each block contains the Block ID (hash here is converted into an index) followed by the time of arrival (Ta) and followed by "By : Miner" representing who is the miner of that particular block. This will help us analyse and critque the various results in a systematic manner. The following is a sample output of the visualisation to give us an idea of how exactly the blockchain trees are created and visualised by our program:

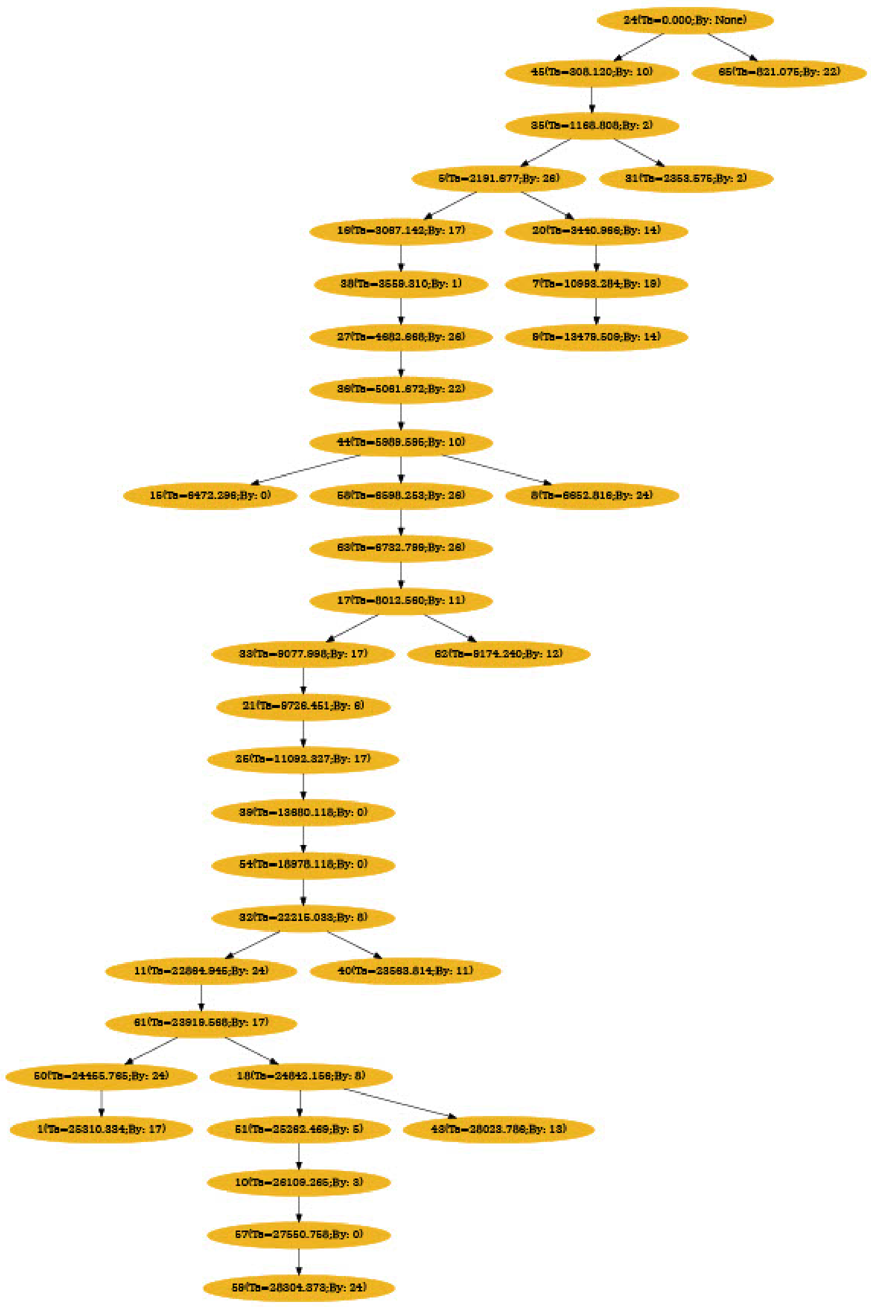


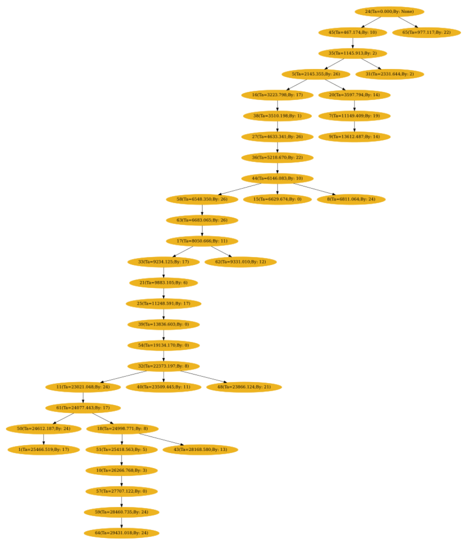
Figure 3: Sample Blockchain Tree

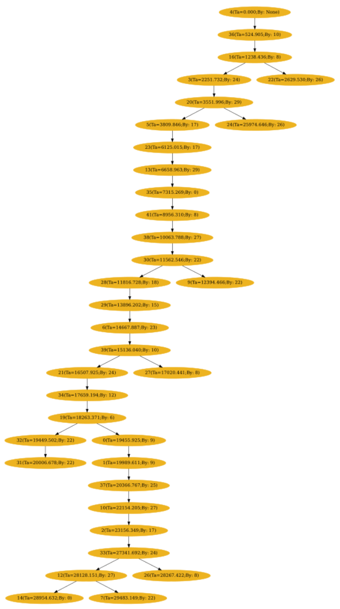
9 Observations and Insights

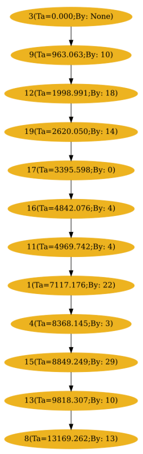
In this section we will provide our observations based on the graphical tree outputs that we obtained by varying different parameters of our simulation.

9.1 Average Block Inter-arrival Time (I)

Keeping other parameters constant, we ran the simulation on interarrival time of blocks' values 500, 1000 and 2000.
As we can see from the blockchain trees generated, when I is low, the blocks are generated at higher rate causing more branching whereas for higher I, the interarrival time is high so when one block is created it reaches everyone before the next block gets generated causing lesser branching.


(a) I=500

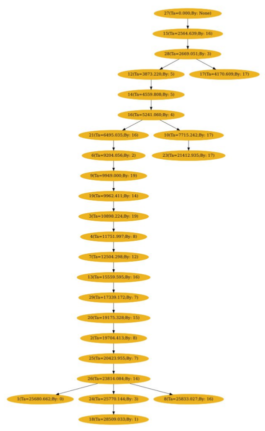

(b) I=1000

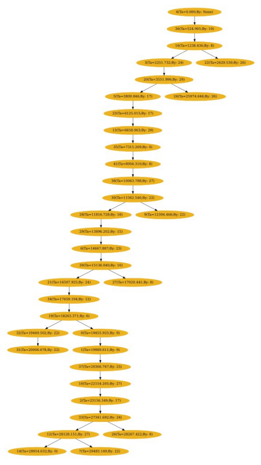


(c) I=2000

9.2 Number of nodes in the network (n)

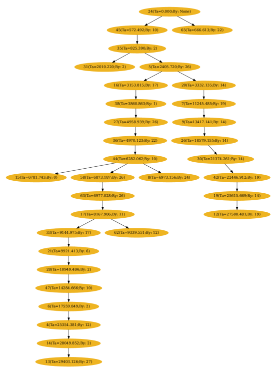
Keeping other parameters constant, we ran the simulation for 2 different values of the number of nodes in our network - 20 nodes and 30 nodes and here are the output blockchain trees of both of them side by side.
Here we can observe that the tree for the 20 node network can be seen to be containing lesser number of blocks as well lesser amount of branching and a relatively straighter blockchain while the one for the 30 node network contains more number of blocks with relatively more branching in the blockchain.
This can be explained by the fact that with lesser number of nodes, there are lesser number of blocks and transactions being generated which thus reduces the probability of a fork/branching happening as the blocks would get invalidated or not be generated in a pace fast enough.

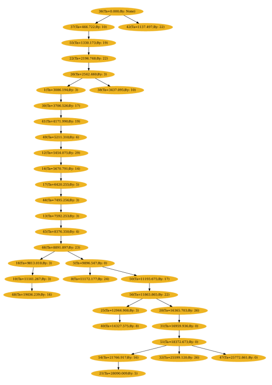

(a) n = 20


(b) n = 30

9.3 Mean Inter-arrival Time of Transactions

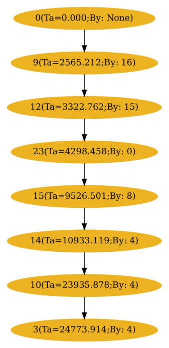
Keeping other parameters constant, we ran the simulation for 2 different values of the mean inter-arrival time of transactions - 500 and 1000. Here are the output blockchain trees of both of them. On low mean interarrival time more transactions are generated and initial blocks contain more transactions in them. On high mean, most blocks will contain only coinbase transactions in blocks they create.


(a) $T_{tx} = 500$

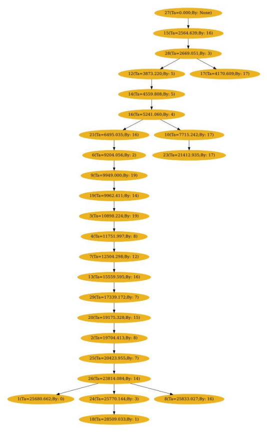

(b) $T_{tx} = 1000$

9.4 Fraction of slow nodes z_0

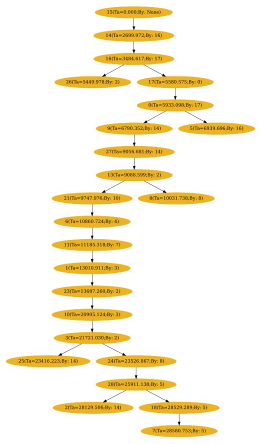
In this section, keeping other parameters constant, we will vary the number of slow nodes in our network z_0. We will try for values 0.25,0.5.0.75 i.e percentage of slow nodes in the network.
As we can clearly observe, with increasing number of slow nodes in the network, the blockchain tree begins to have an increased number of branching in its nodes as well as a greater number of nodes in the tree that is generated. The trend can be explained by the fact that with increasing number of slow nodes, the transaction would be forwarded at a slower pace and hence the probability of forks increases as the new block would reach later after a block has been mined already and the main chain has progressed.



(a) $z_0 = 0.25$



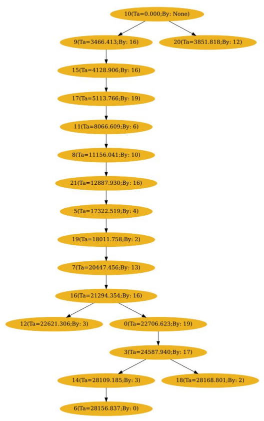
(b) $z_0 = 0.5$



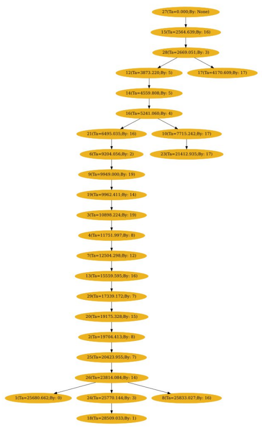
(c) $z_0 = 0.75$

9.5 Fraction of lowcpu nodes z_1

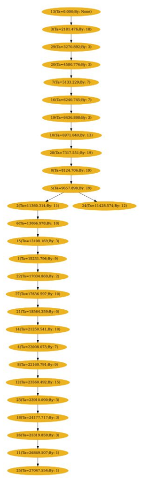
In this section, keeping other parameters constant, we will vary the number of lowcpu nodes in our network z_1. We will try for values 0.25,0.5.0.75 i.e percentage of lowcpu nodes in the network.
As we can clearly observe, with increasing number of lowcpu nodes in the network, the blockchain tree begins to have a reduced number of branching in its nodes and the tree begins to have a straighter structure with a majority of blocks being those of the few highcpu nodes that are there in the network. It is a clear trend as demonstrated below.
This can be explained by the fact that with increasing lowcpu nodes, the rate of block generation reduces and forks have reduced as there will be no clashes.



(a) $z_0 = 0.25$



(b) $z_0 = 0.5$



(c) $z_0 = 0.75$

9.6 Block generation and inclusion Ratio

In this section, we observe 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. After careful analysis of the data that we obtain after simulating various blockchain trees, we conclude to have a general order of variation of this ratio depending on node type as follows:

Fast,High cpu Node > Fast,Low cpu Node > Slow,High cpu Node > Slow,Low cpu Node

Here we provide an example tree and the values that we obtained for the ratio for various types of nodes:

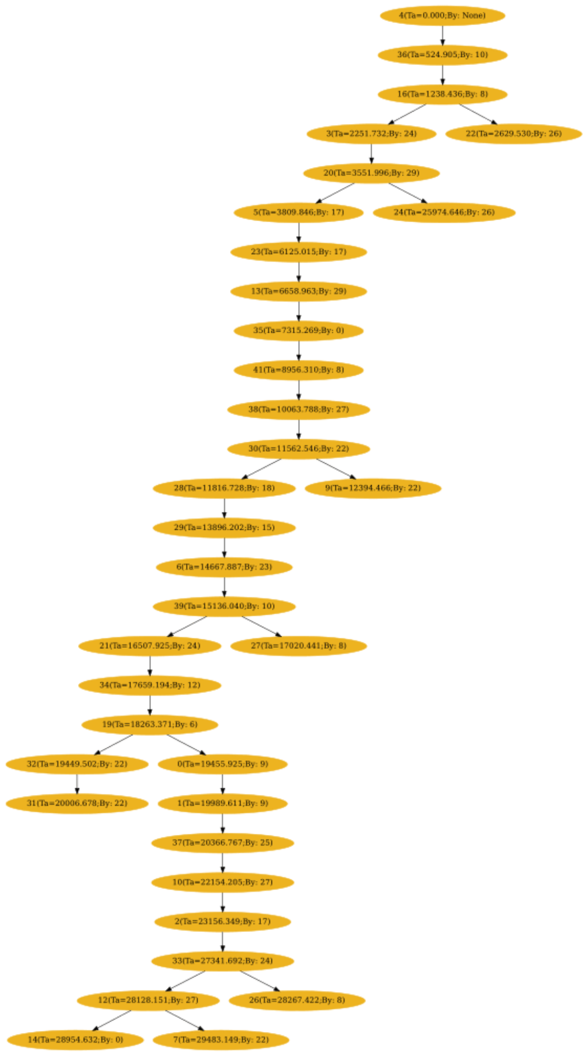


Figure 9: Sample tree for ratio

Now consider the node 17 which is a fast and highcpu node. The ratio that we obtain for node 17 is 3/26, i.e 3 of the 26 blocks that it generated were a part of the main chain which is an impressive number.
Now we consider the node 8 which is a Fast and lowcpu node. The ratio that we obtain for node 8 is 2/10 i.e i.e 2 of the 10 blocks that it generated were a part of the main chain which is lesser than 17 but still good considering it generated lesser blocks. Now we consider the node 12 which is a slow and highcpu node. The ratio that we obtain for node 12 is 1/26 i.e i.e 1 of the 26 blocks that it generated were a part of the main chain which is lesser than 8 but still good made one block a part of the chain. Now we consider the node 2 which is a slow and lowcpu node. The ratio that we obtain for node 2 is 0/9 i.e i.e 0 of the 9 blocks that it generated were a part of the main chain which means neither did it generate a good number of blocks but also none of them could make it to the main chain.
This trend can be explained simply by the fact that the probability of a block to be included in

the main chain increases with high cpu capacity and high link speed. High CPU capacity helps as it simply allows the node to generate more and more blocks which then may have a chance of being included in the main chain. High link speed ensures that block created gets transmitted to all peers quickly which would ensure that the block has a lesser chance of being invalidated by the arrival of some other block.

9.7 Branch Length

On setting parameter values to: $z_0 = 0.5,\ z_1 = 0.5,\ n = 30\ I = 500$ we get blockchain trees with lot of branching as seen in tree below. The main chain length is 20 whereas the longest branch length is 12. Of all the output graphs obtained this was a rare event where a branch was this long. One reason for this is too many simultaneous blocks being generated and transferred.

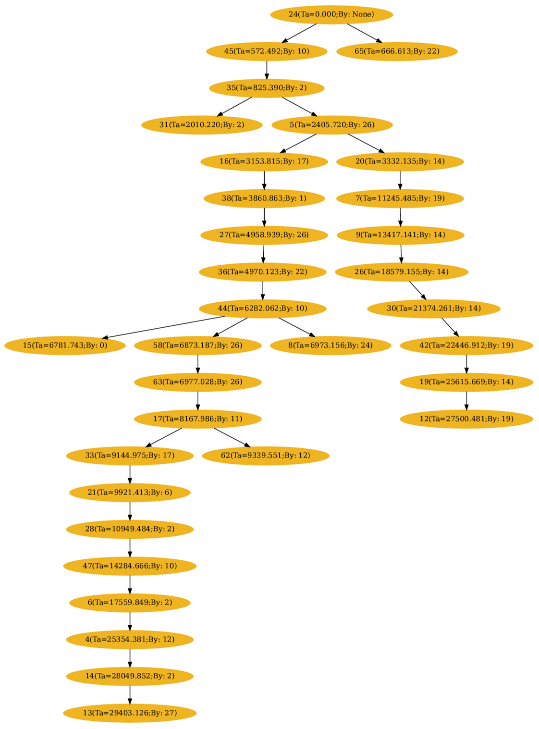


Figure 10: Sample tree for long branch

10 References

- Block arrivals in the Bitcoin Blockchain:

  https://arxiv.org/pdf/1801.07447.pdf
- SimPy Reference

  https://simpy.readthedocs.io/en/latest/simpy_intro/
- iGraph Library

  https://igraph.readthedocs.io/en/stable/