CS765 : PROJECT PART 1

SIMULATION OF A P2P CRYPTOCURRENCY NETWORK

Submitted On :

Saturday, February 17, 2024

Submitted By :

Swetha M (23M0756)

Chaitra Gurjar (23M0831)

Index

1 In Text Questions

(1) The interarrival 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. What are the theoretical reasons for choosing the exponential distribution?

Memory-less property of exponential distribution makes sure that time elapsed between last transaction and current time does not affect the generation probability of the next transaction. It is a perfect way to model the random arrival of transactions in the blockchain network. Also, it has a single parameter lambda which determines the mean of the distribution. This makes it computationally efficient.

(2) Latency = $\rho_{ij} + |m|/c_{ij} + d_{ij}$, where

ρ_{ij} : speed of light propagation delay $\in$ U[10ms, 500ms] at the start of the simulation
$|m|$: length of the message in bits
c_{ij} : link speed b/w *i* and *j* in Mbps, c_{ij}=100 if i and j are fast; c_{ij}=5 if *i* or *j* is slow
d_{ij} : queuing delay at node *i* to forward the message to node *j* $\in$ Exp (96 kbits / c_{ij})
Why is the mean of d_{ij} inversely related to c_{ij} ? Give justification for this choice.

The queuing delay (d_{ij}) measures the amount of time a message spends in a queue before being processed. It is less if the queue is empty i.e. the messages arrive at a regular pace. This gives enough time for processing the messages. However, if all messages arrive at the same time, the messages have a queuing delay. c_{ij} is the speed of link between nodes i and j. The queuing delay depends on the link speed. The queue is mostly empty for nodes with high link speed as messages are already processed. At low link speed, the queue becomes longer which increases the queuing delay. Hence, the inverse relation.

(3) Explanation for the choice of a particular mean for T_k.

If mean is a constant not varying based on the hashing power of a peer, all peers will get the same probability to create a block
By inversely varying the mean of the distribution with hashing power, higher CPU nodes can mine a block in a smaller time. This is the perfect simulation of PoW where higher CPU nodes try more nonce values per second.
Number of hashes/sec $\propto$ Hashing power $\propto$ Probability of mining a block
Also all hashing powers are normalized to sum to 1. This ensures that the same number of blocks are created irrespective of the number of peers.

2 Properties of P2P Network

(1) The network is connected.

Each peer has a degree between 3 and 6. However, while creating a random graph, there is a possibility of it not being connected. In such cases, we find all connected components in the graph (called clusters). We then randomly choose a peer from each cluster, and connect these to each other. This makes the graph connected.

(2) Messages (transactions and blocks) follow loopless forwarding.

To ensure loopless forwarding, we follow two rules :

a) A peer does not forward the message to its sender.

While finding the neighbors of a peer, we check if any of them is the sender of this peer. We then remove this neighbor from the forwarding list.

b) A peer does not forward the message to a peer to which it has already sent.

We maintain a list of peers which have already forwarded this transaction to their neighbors. This list is stored for each transaction and checked before forwarding.

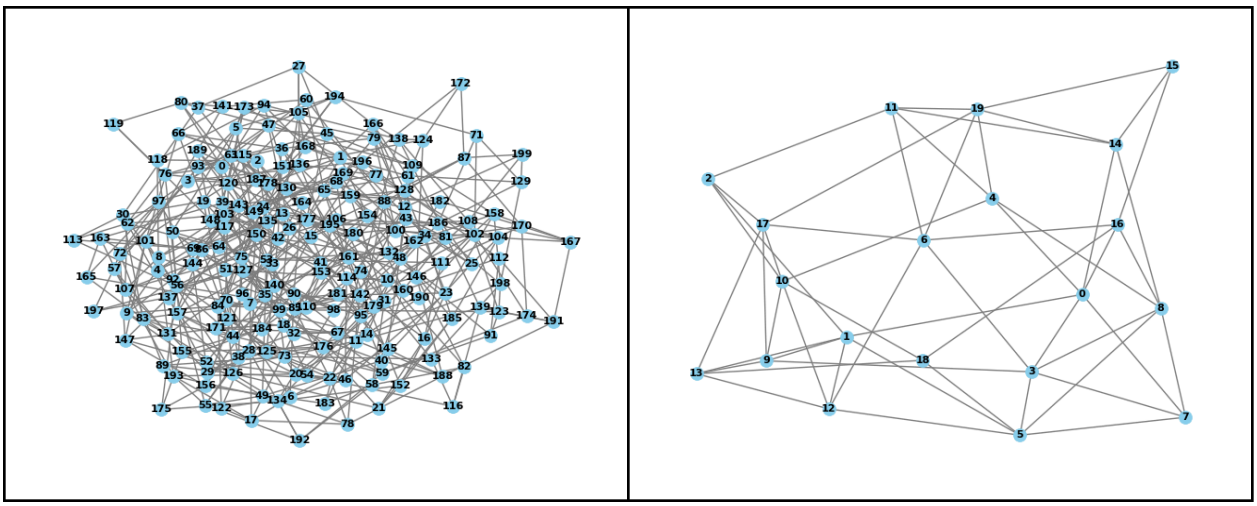

Figure 1 : Visualization of P2P network - Large vs Small

3 Changing Parameters : Results

There are two parameters for each peer, which affect the blockchain - hashing power (low or high) and link speed (slow or fast). Hence, we can tweak the parameters to obtain the fraction of blocks in each of the four combinations.

3.1 Change Number of Peers (n)

Other Parameters : I = 60 ; Ttx = 60 ; Slow = 0.5 ; LowCPU = 0.5

	Fraction of blocks created by				Fraction of blocks added in Longest Chain	Total blocks created in the blockchain
n	Slow Low	Fast Low	Slow High	Fast High		
10	0.000	0.083	0.500	0.417	1.000	12
100	0.105	0.000	0.579	0.316	1.000	19

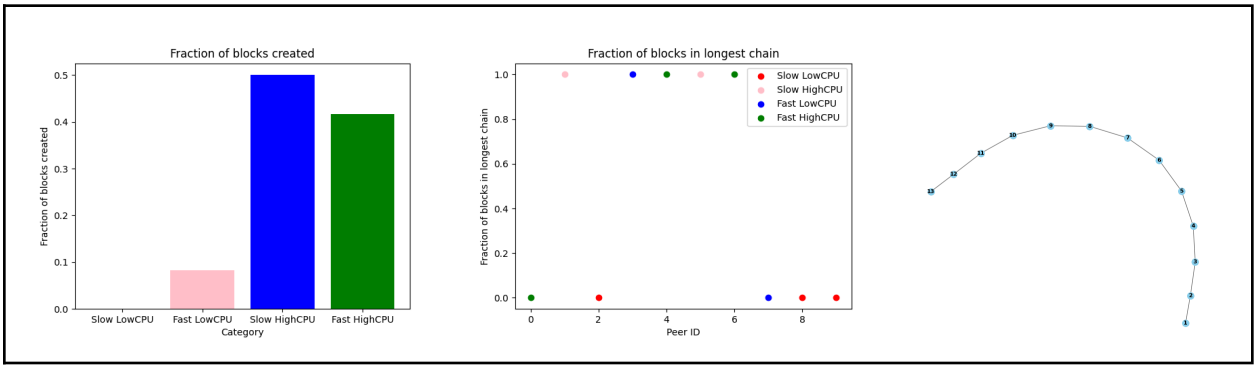

Figure 2 : Changing no. of peers - less peers

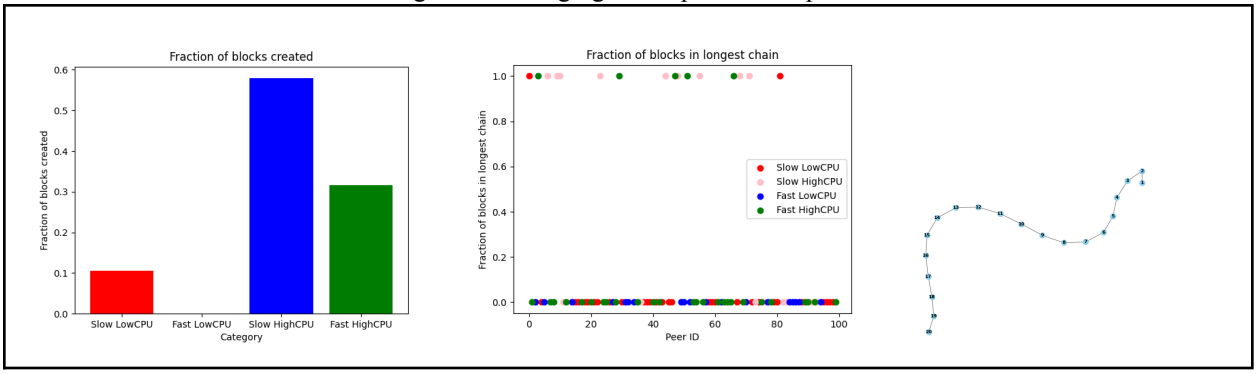

Figure 3 : Changing no. of peers - more peers

Observations :

- ➔ The number of blocks created remains constant. This is because the hashing power of all nodes sum up to 1 and it inversely affects the time of mining.
- ➔ Fraction of peers whose blocks made the longest chain is really less when there are a large number of peers.
- ➔ High CPU peers dominate Slow CPU peers in both cases.

3.2 Change Block Interval (I)

Other Parameters : n = 20 ; Ttx = 6 ; Slow = 0.5 ; LowCPU = 0.5

	Fraction of blocks created by				Fraction of blocks added in Longest Chain	Total blocks created in the blockchain
I	Slow Low	Fast Low	Slow High	Fast High		
0.1	0.029	0.059	0.676	0.236	0.676	34
1	0.000	0.027	0.459	0.514	0.973	37

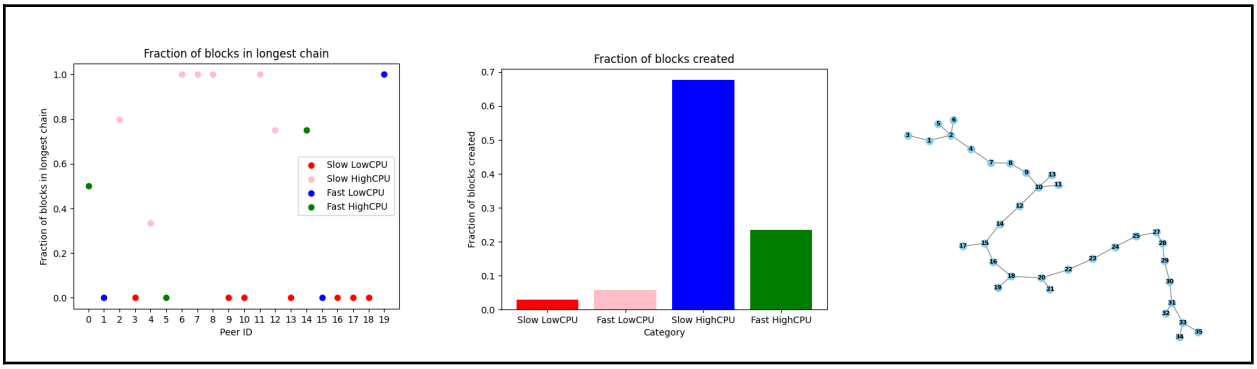

Figure 4 : Changing block interval - low interval

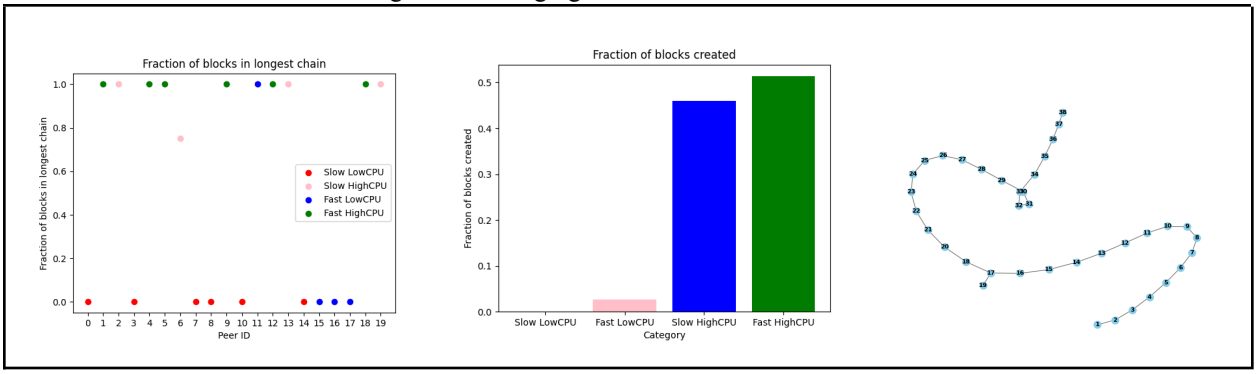

Figure 5 : Changing block interval - high interval

Observations :

- ➔ Low block interval creates forks/branches in the blockchain tree. This is because a peer mines a block before another block is received from its peers. The fork happens when the block interval is less than the time for forwarding the blocks to all the neighbors.
- ➔ Forked conditions favor low link speed fast CPU peers. They are able to mine blocks and add to their tree. Fast CPU peers continue to dominate block creation.
- ➔ Less forks are observed with high block intervals as the blocks reach all the peers within a small time. Fast CPU peers dominate block creation.
- ➔ Lower block interval has higher forks (32% forks) compared to higher block interval (3% forks). The ratio of generated blocks in the longest chain to the generated blocks is high when the block interval is higher.

3.3 Change Transaction Interval (Ttx)

Other Parameters : n = 50 ; I = 0.2 ; Slow = 0.7 ; LowCPU = 0.1

	Fraction of blocks created by				Fraction of blocks added in Longest Chain	Total blocks created in the blockchain
Ttx	Slow Low	Fast Low	Slow High	Fast High		
0.1	0.000	0.000	0.684	0.316	0.579	19
10	0.000	0.000	0.769	0.231	0.692	26

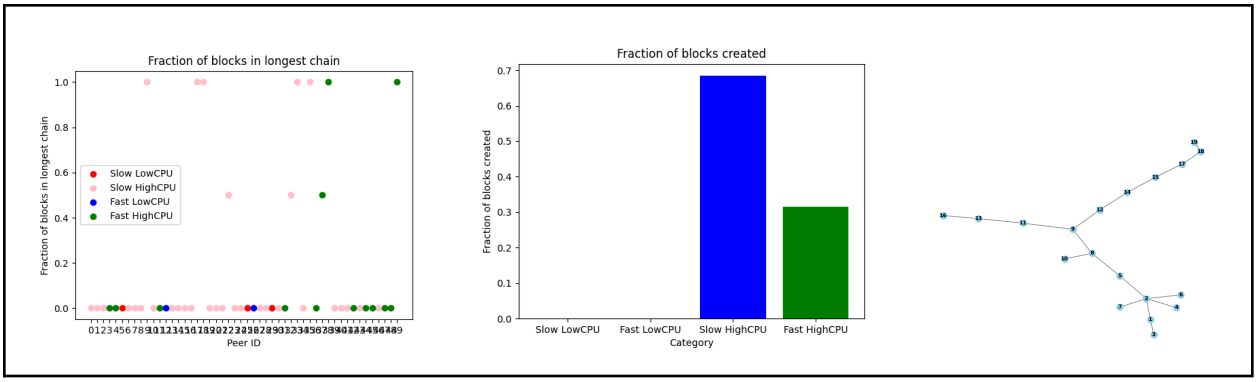

Figure 6 : Changing transaction interval - low interval

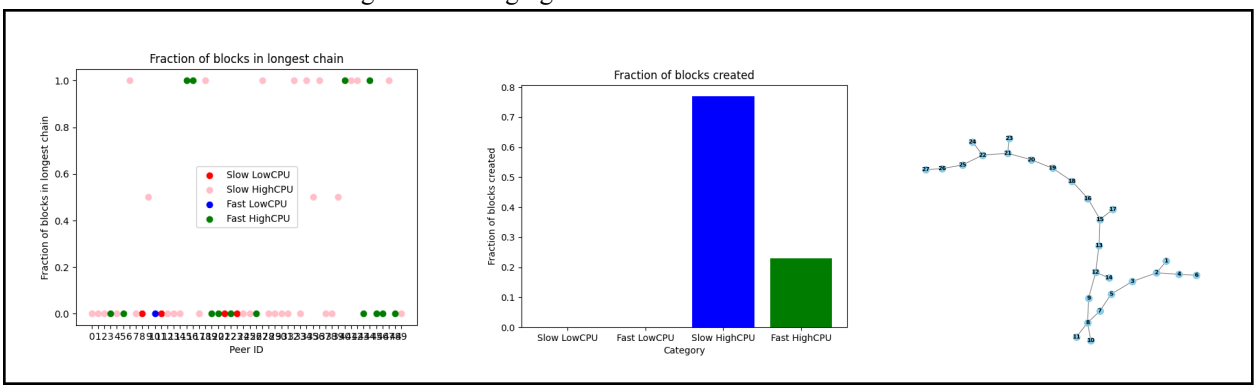

Figure 7 : Changing transaction interval - high interval

Observations :

- ➔ Forks are observed in both cases if the block interval is really less. Forks are slightly higher when Ttx is less.
- ➔ If the block interval is higher, a linear, non forked chain is formed for both low and high values of the Transaction interval.
- ➔ High CPU peers dominate as usual.

3.4 Change Fraction of Slow Nodes (z0)

Other Parameters : n = 50 ; I = 0.2 ; Ttx = 0.02 ; LowCPU = 0.5

	Fraction of blocks created by				Fraction of blocks added in Longest Chain	Total blocks created in the blockchain
z0	Slow Low	Fast Low	Slow High	Fast High		
0.1	0.000	0.154	0.077	0.769	0.846	13
0.9	0.071	0.000	0.786	0.143	0.643	14

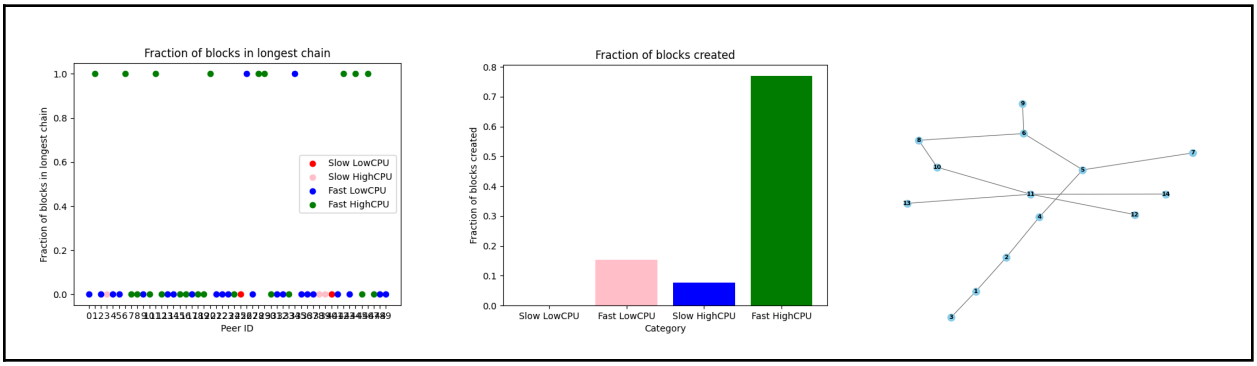

Figure 8 : Changing fraction of slow nodes - less slow nodes

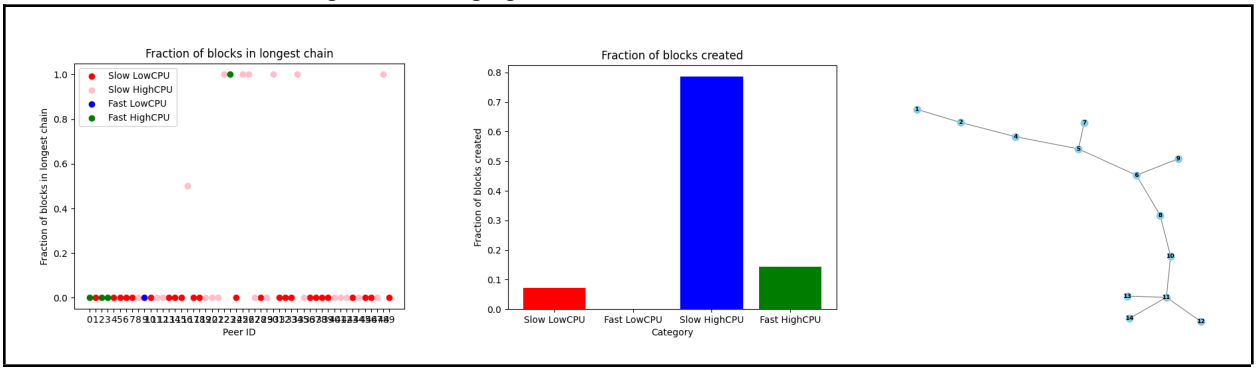

Figure 8 : Changing fraction of slow nodes - more slow nodes

Observations :

- ➔ Larger Slow nodes - Higher forks (36% forks) than when majority are fast nodes (16% forks). Ratio of the generated blocks in the largest chain is less when there are more slow nodes.
- ➔ This denotes that propagation of blocks to all peers takes more time and new branches can get created during that time.
- ➔ When majority are slow, Slow High CPU peers dominate the Fast High CPU in block creation and vice versa when majority are fast nodes.

3.5 Change fraction of low CPU nodes

Other Parameters : n = 20 ; I = 60 ; Ttx = 6 ; Slow = 0.5

	Fraction of blocks created by				Fraction of blocks added in Longest Chain	Total blocks created in the blockchain
z1	Slow Low	Fast Low	Slow High	Fast High		
0.1	0.000	0.000	0.448	0.552	1.000	29
0.9	0.275	0.241	0.275	0.209	1.000	29

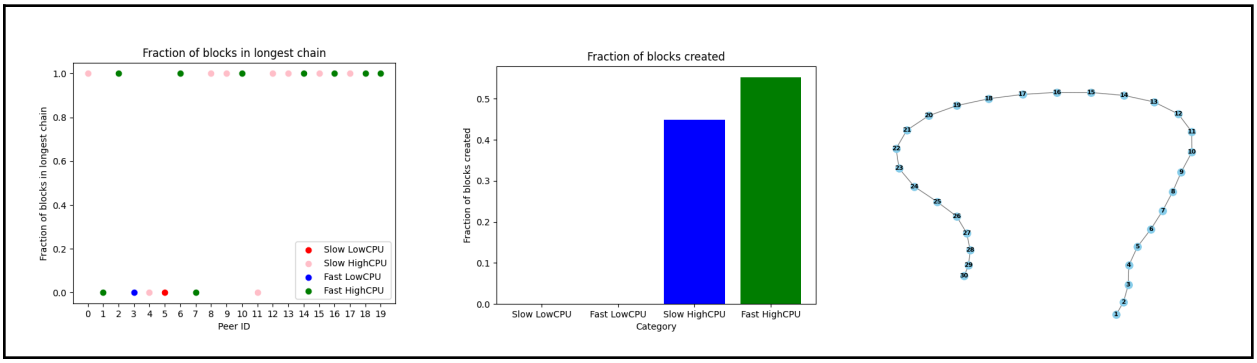

Figure 9 : Changing fraction of low CPU nodes - less low CPU nodes

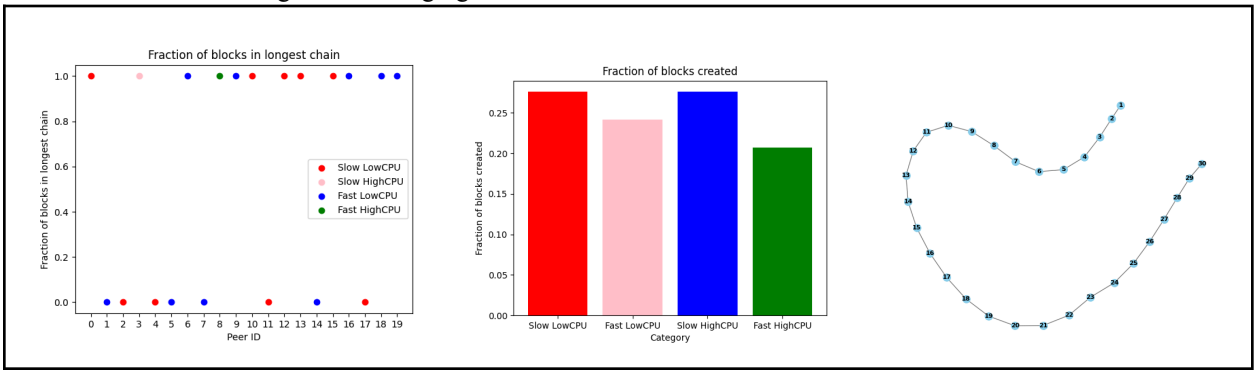

Figure 10 : Changing fraction of low CPU nodes - more low CPU nodes

Observations :

- ➔ Larger low CPU peers - all the peers have equal amount of blocks created.
- ➔ Larger number of high CPU peers - low CPU peers have zero blocks created.
- ➔ Forks are not created if the block interval is constant and large (greater than latency).

4 Some Insights on Forking

The above scatter plots for different parameters show the ratio of making to the longest chain vs generation of the block.

Fast nodes

High CPU - most generated blocks are able to make it to the longest chain. Some are orphaned as forking is high.
Low CPU - No forks, only few are able to create blocks and all created make the longest chain.

Slow nodes

High CPU - Most get orphaned in this particular run of the simulation.
Low CPU - Not able to create enough blocks.

When there are more forks, the ratio of generated blocks in the longest chain to the total number of generated blocks is very less (25% in this case).
Branches in terms of number of blocks generated are 29 out of 48 = ~60% (***higher)***
Longest chain length = 12 out of 48 = 25% ***(less)***

When there are no forks, Branch percentage is ***0%*** and longest chain length is 100%. This is observed when the delay in the network is less and the block interval is sufficiently high.

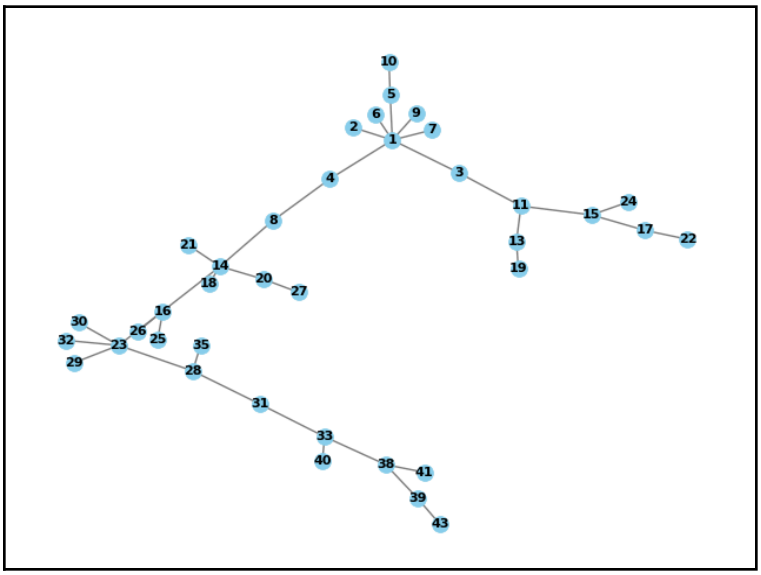

Figure 11 : High forking in a blockchain

5 Conclusion

The influence of varying parameters like scale, block and transaction interval and the proportion of high CPU peers on the blockchain network was studied in detail. The conditions which lead to forking were also examined with care. The modeling of PoW with probabilistic distributions which are ideally equivalent to real-life mining helped understand the internal working of the blockchain. The computation and space involved in the maintenance of the block trees and resolving the longest chain indicated how important it is to efficiently design the blockchain network.

6 Appendix

Some implementation details are as follows.

1) This part of the project is implemented in Python. It provides classes and objects to easily deal with entities such as transactions, peers, links and trees. It also provides some standard libraries like ‘random’ which is helpful to sample from various distributions.

2) SimPy : SimPy is a python library used to simulate events. The function call ‘env.process’ schedules events in the simulation's event queue, specifying when actions associated with the process should occur.

3) NetworkX Graph : In NetworkX, nodes can be any hashable object e.g. a text string, an image, an XML object, another Graph, a customized node object, etc. We noticed that sometimes, the creation of graphs using this library can lead to flooding of resources, which leads to a pause in the execution. In such cases, forceful termination and restart reloads the correct execution.

4) matplotlib : Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. We have used this to visualize the network and peer trees containing blocks.