CS765 Spring 2023 Semester, Project Part-1

Simulation of a P2P Cryptocurrency Network

C. Srikanth Yadav
23M0794

G. Siva Prasad Reddy
23M0747

Sourabh Kumar Kale
23M0783

The theoretical reasons for choosing the exponential distribution ?

Given the hashing power as β in a time frame of Δ, the probability of mining a new block is βΔ.

Let's denote the time interval I as $n\Delta$, where n represents the number of time frames.

The probability of not mining a block within the time interval I can be expressed as P[I > x].

By substituting I with x, we get the expression for the probability of not mining a block within time x as $(1 - \frac{\beta x}{n\Delta})$.

As the number of time frames *n* approaches infinity,

the expression $(1 - \frac{\beta x}{n\Delta})$ converges to the exponential function, $e^{(-\beta x)}$.

Therefore, we can conclude that the probability of not mining a block within a given time interval follows an exponential distribution.

Why is the mean of dij inversely related to cij ? Give justification for this choice ?

The mean queuing delay (d_{ij}) is inversely related to the link speed (c_{ij}) because higher link speeds reduce congestion and queuing delays. With increased link speed, the outgoing link's capacity rises, resulting in shorter queuing delays.

Explanation for the choice of a particular mean for Tk.

An optimal mean value for T ensures a balance between the frequency of block generation and the number of transactions per block. When there are more blocks generated, each block tends to contain fewer transactions, leading to underutilization of block capacity. On the other hand, if the rate of block generation is too low, it may result in a backlog of pending transactions awaiting inclusion in blocks. Therefore, it is important for selecting an appropriate mean value for T is essential to maintain an equilibrium between block frequency and transaction density within each block.

Program Outline :

1. Class - Transaction:
    - TransactionID (Unique transaction id)
    - Source (Unique Node id of the payer)
    - Destination (Unique Node id of payee)
    - Transfer Amount (Value to be paid)
    - Message (Transaction message)

2. Class - Block
    - Block ID (Unique block id)
    - ParentID (ID of block on which it is mined)
    - Length of Chains (length of the chain in which it is till this block)
    - Transactions (Transactions inserted in this block)
    - Pointed_by (Blocks which are mined on this block i.e child)
    - Time_Created (Time the block has created)
    - Time_Added (Time it is added by the node to it's blockchain)

3. Class - Node
    - NodeID (Unique node id)
    - Type_of_CPU
    - Type_of_node
    - Amount (Balance of this node)
    - Hashing Power (Hashing power the node has based on type of CPU)
    - Transactions (Transactions seen by this node)
    - Peers (Peers this node is connected to)
    - Genesis_Block

4. Class - Event
    - Operation (name of the operation to be performed)
    - ID (id of the transaction/block this event is about)
    - Source (ID of the node to which this operation is targeted to)
    - Time (time at which this operation to be performed)

Proposed Approach :

We are using an Event priority queue to synchronize the events i.e transaction generation, transaction receive, block generation, block receive etc. Every event will have time associated with it at which this event should be performed. Every time we will pick the event with least time and we will perform it, if it creates some other event then we will push that new event into the queue again with the time associated to each event.

If every node starts mining a block at time 0, then every time the first block added to the block chain will be an empty block. So we have created a first_block_gen for every node at start. Here every node will start mining at some random time chosen from exponential distribution with mean 100.

So the node which has the least time will add the first block into the blockchain and broadcast this block into the network. If any other node has a first_block_gen event performed before it receives this new block then we can have a fork at first level. If a node has not performed the first_block_gen event before receiving this block, then the node will shift to mine on this new block.

When this node sees a first_block_gen event created by this, it will check whether the node is working on the same block this first_block_gen event is pointing to or it have shifted to another block, as we have shifted the node mining point when it has received a block from other node, it will come to know that a new longest chain has created before I have cracked the POW and it will ignore this block.

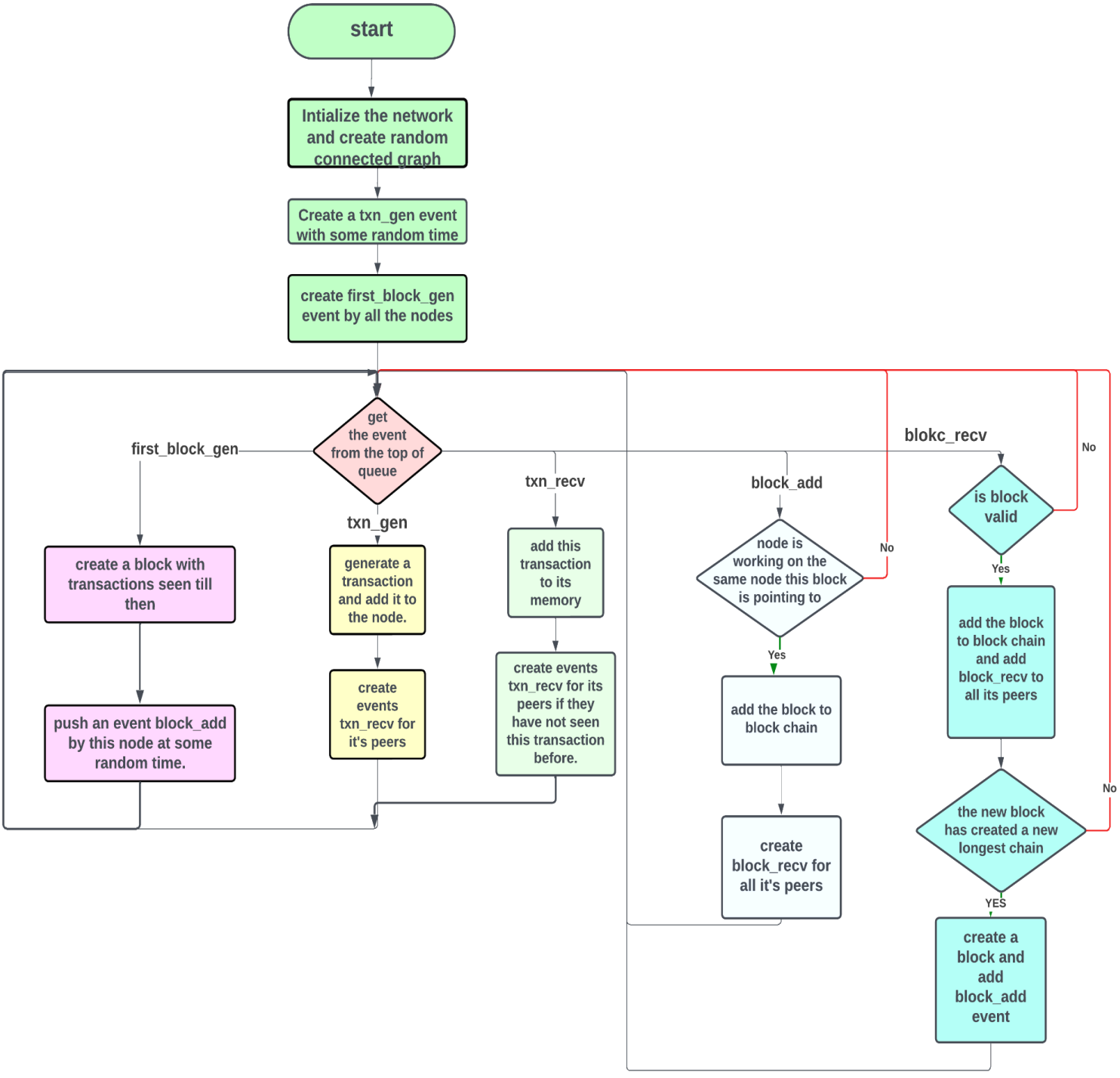
How are forks resolved ?

Every time when we receive a block and the block is valid, after inserting it into the chain if this block has created a longer chain(by checking chain length field in prev. block) than the chain on which the node is currently working on then the node will shift to the new chain and start mining there.

When we receive a block_gen event created on the previous shortest chain, by seeing the node working point the node comes to know that we have got a new longest chain and ignores this event.

When there is a fork, the node will mine on the block which it has seen first, then based on the length of the chain it shifts to the longest chain.

FLOW CHART:


start
Intialize the network and create random connected graph
Create a txn_gen event with some random time
create first_block_gen event by all the nodes
get the event from the top of queue
first_block_gen
txn_gen
txn_recv
block_add
blokc_recv
create a block with transactions seen till then
push an event block_add by this node at some random time.
generate a transaction and add it to the node.
create events txn_recv for it's peers
add this transaction to its memory
create events txn_recv for its peers if they have not seen this transaction before.
node is working on the same node this block is pointing to
No
Yes
add the block to block chain
create block_recv for all it's peers
is block valid
No
Yes
add the block to block chain and add block_recv to all its peers
the new block has created a new longest chain
No
YES
create a block and add block_add event

Observations and Analysis based on Parameters :

The following are the parameters used for Analysis :

- Number of Nodes (n)
- Percentage of Slow Nodes (z_0)
- Percentage of Low_CPU_Nodes (z_1)
- Mean Time for Exp. Dist. for inter arraival Transactions (T_{tx})
- Mean Time for Exp. Dist. for Block Creation ($T_k = I/h_k$)

H_k Is the hashing power of the node. It depends on the type of CPU the node has. The hashing power of the High CPU node is 10 times larger than the hashing power of the Low CPU node. And the sum of hashing power of all nodes should be 1.

Let the no.of nodes is n

no.of Low CPU nodes =m

Hashing power of Low CPU nodes =H_k

then no.of High CPU nodes = n-m

We know the sum of hashing power is 1.

So $m H_k + (n-m)\ 10 H_k = 1$

$(m+10n-10m) H_k = 1$

$H_k = 1 / (10n-9m)$

So the hashing power of each Low CPU node is 1/(10n-9m)

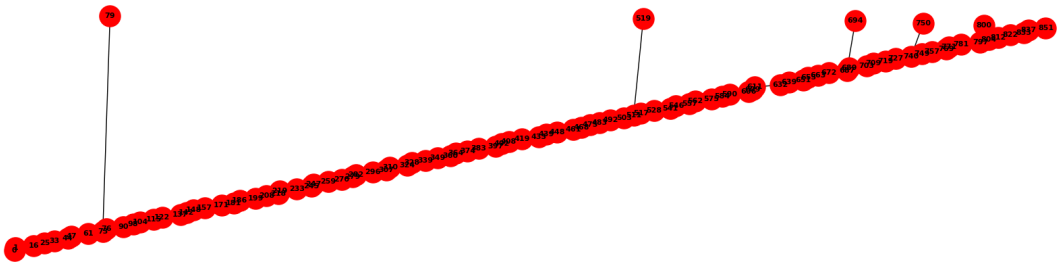
Hashing power of each High CPU node is 10/(10n-9m)

Note : We are running our program until we have *100 Blocks* in the block chain of each node.

Observation-1 : (n=10, z_0=60, z_1=20, T_{tx}=1 sec , I=10 sec)

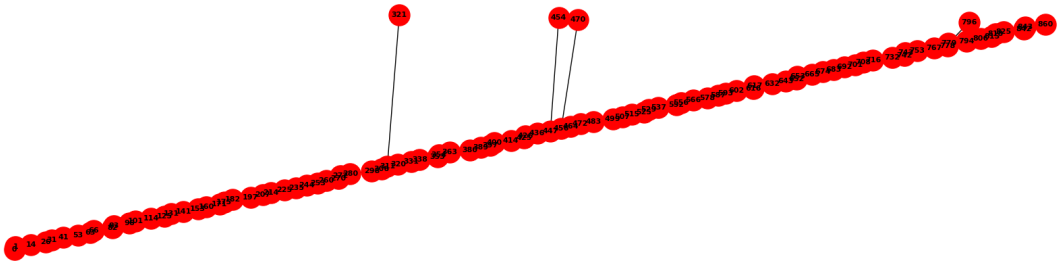
Total Time : 945 sec

Forks : 5



Node	Type of Node	Type of CPU	Txns generate	Blocks generate	Blocks Added (including forks)	Blocks in Longest chain	Contribution in Longest chain(%)
0	fast	high CPU	165	84	14	12	12.63
1	fast	high CPU	154	84	13	12	12.63
2	fast	high CPU	157	82	15	13	13.68
3	slow	low CPU	147	95	0	0	0
4	slow	high CPU	163	85	11	11	11.57
5	slow	high CPU	161	87	8	8	8.42
6	slow	high CPU	178	84	12	12	12.63
7	slow	low CPU	163	93	2	2	2.10
8	slow	high CPU	167	82	13	13	13.68
9	fast	high CPU	150	85	12	12	12.63

Observation-2 : (n=10, z_0=60, z_1=40, T_{tx}=1 sec , I=10 sec)
Total Time : 1272 sec
Forks : 4



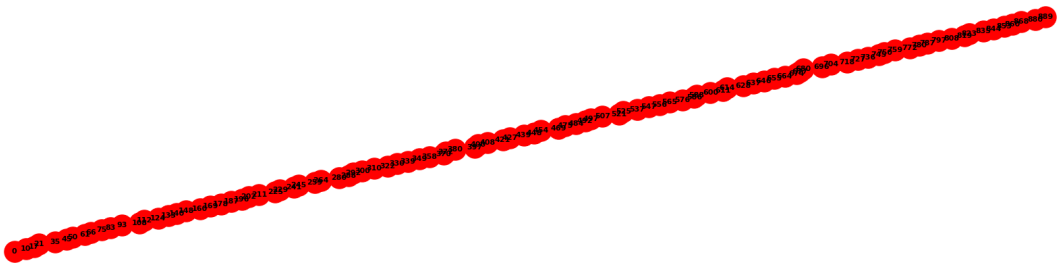
Node	Type of Node	Type of CPU	Txns generate	Blocks generate	Blocks Added (including forks)	Blocks in Longest chain	Contribution in Longest chain(%)
0	slow	low CPU	189	96	0	0	0
1	slow	high CPU	229	82	16	15	15.62
2	fast	high CPU	214	82	15	15	15.62
3	fast	high CPU	212	84	13	13	13.54
4	fast	high CPU	234	83	15	13	13.54
5	slow	low CPU	238	96	0	0	0
6	slow	high CPU	231	77	21	21	21.87
7	slow	high CPU	211	78	18	18	18.75
8	fast	low CPU	188	95	1	1	1.04
9	slow	low CPU	212	96	1	0	0

We can observe that no.of forks decreased and time taken increased with the decrease in no.of High CPU nodes with same block inter arrival time.

Observation-3 : (n=10, z_0=50, z_1=80, T_{tx}=1 sec , I=60 sec)

Total Time : 8530 sec

Forks : 0

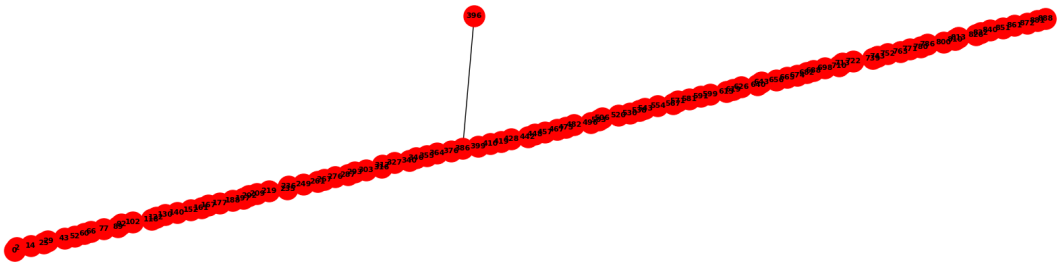


Node	Type of Node	Type of CPU	Txns generate	Blocks generate	Blocks Added (including forks)	Blocks in Longest chain	Contribution in Longest chain(%)
0	slow	low CPU	1512	97	3	3	3
1	fast	high CPU	1574	71	29	29	29
2	slow	low CPU	1450	93	7	7	7
3	fast	low CPU	1582	94	6	6	6
4	fast	low CPU	1513	95	6	6	6
5	fast	low CPU	1516	96	4	4	4
6	fast	low CPU	1572	93	7	7	7
7	slow	low CPU	1468	95	5	5	5
8	slow	high CPU	1501	69	31	31	31
9	slow	low CPU	1505	98	2	2	2

Observation-4 : (n=10, z_0=50, z_1=20, T_{tx}=1 sec, I=60 sec)

Total Time : 6017 sec

Forks : 1



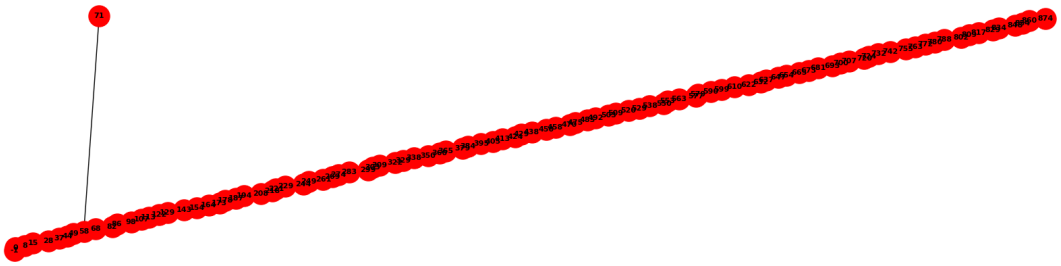
Node	Type of Node	Type of CPU	Txns generate	Blocks generate	Blocks Added (including forks)	Blocks in Longest chain	Contribution in Longest chain(%)
0	fast	high CPU	1072	85	15	15	15.15
1	slow	high CPU	1088	89	10	10	10.10
2	slow	low CPU	1093	98	1	1	1.01
3	fast	high CPU	1053	89	10	10	10.10
4	slow	high CPU	1057	88	11	11	11.11
5	slow	low CPU	1058	97	2	2	2.02
6	slow	high CPU	1029	91	9	9	9.09
7	fast	high CPU	1043	88	11	11	11.11
8	fast	high CPU	1048	80	19	19	19.19
9	fast	high CPU	1033	88	12	11	11.11

We can observe the increase in time taken and decrease in no.of forks with the increase in block arrival time from 10 to 60 sec. And in 60 sec too there is increase in time taken and decrease in no.of forks with decrease in no.of high cpu nodes.

Observation-5 : (n=10, z_0=50, z_1=20, T_{tx}=1 sec , I=30 sec)

Total Time : 3059 sec

Forks : 1

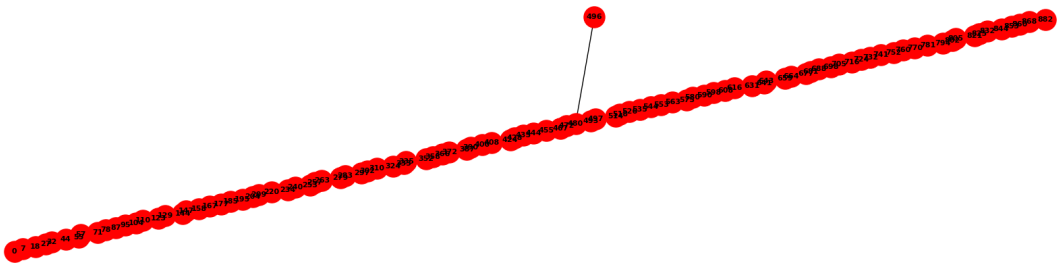


Node	Type of Node	Type of CPU	Txns generate	Blocks generate	Blocks Added (including forks)	Blocks in Longest chain	Contribution in Longest chain(%)
0	fast	low CPU	569	98	0	0	0
1	fast	high CPU	552	89	10	10	10.10
2	slow	high CPU	551	90	8	8	8.08
3	fast	high CPU	559	94	5	5	5.05
4	slow	high CPU	508	85	13	13	13.13
5	slow	low CPU	517	94	4	4	4.04
6	fast	high CPU	543	81	18	18	18.18
7	slow	high CPU	570	83	15	15	15.15
8	fast	high CPU	550	88	12	12	12.12
9	slow	high CPU	537	84	15	14	14.14

Observation-6 : (n=10, z_0=50, z_1=80, T_{tx}=1 sec , I=30 sec)

Total Time : 4221 sec

Forks : 1

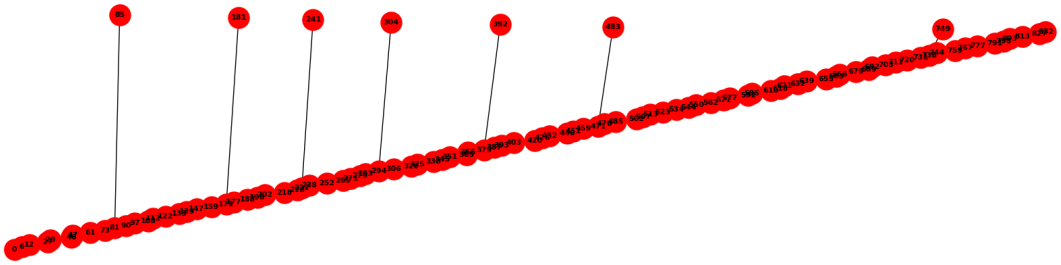


Node	Type of Node	Type of CPU	Txns generate	Blocks generate	Blocks Added (including forks)	Blocks in Longest chain	Contribution in Longest chain(%)
0	slow	low CPU	737	92	7	7	7.07
1	fast	low CPU	708	94	5	5	5.05
2	fast	high CPU	734	72	28	28	28.28
3	slow	low CPU	686	94	5	5	5.05
4	slow	low CPU	691	97	2	2	2.02
5	slow	high CPU	680	71	28	28	28.28
6	fast	low CPU	731	92	7	7	7.07
7	fast	low CPU	739	91	8	7	7.07
8	slow	low CPU	728	98	2	2	2.02
9	fast	low CPU	730	92	8	8	8.08

Observation-7 : (n=10, z_0=50, z_1=20, T_{tx}=1 sec , I=5 sec)

Total Time : 569 sec

Forks : 7

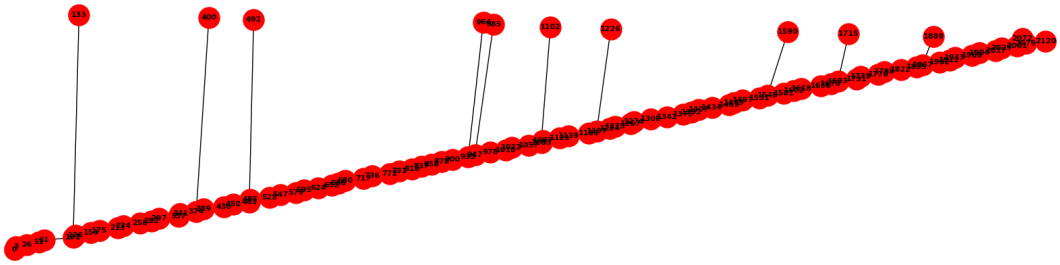


Node	Type of Node	Type of CPU	Txns generated	Blocks generated	Blocks Added	Blocks in Longest chain	Contribution in Longest chain(%)
0	fast	high CPU	105	85	9	9	9.67
1	fast	high CPU	99	75	20	20	21.50
2	slow	high CPU	96	84	12	12	12.90
3	fast	high CPU	97	86	8	7	7.52
4	slow	low CPU	113	89	4	4	4.30
5	slow	high CPU	106	83	11	11	11.82
6	fast	high CPU	75	86	11	8	8.60
7	slow	low CPU	110	93	0	0	0
8	slow	high CPU	113	84	9	9	9.67
9	fast	high CPU	103	80	16	13	13.97

Observation-8 : (n=25, z_0=50, z_1=20, T_{tx}=1 sec , I=5 sec)

Total Time : 484 sec

Forks : 11



Node	Type of Node	Type of CPU	Txns generated	Blocks generated	Blocks Added	Blocks in Longest chain	Contribution in Longest chain(%)
0	fast	low CPU	33	89	0	0	0
1	slow	high CPU	38	86	3	3	3.37
2	fast	high CPU	34	88	1	1-	1.12
3	slow	high CPU	43	85	6	4	4.49
4	slow	low CPU	36	89	0	0	0
5	fast	high CPU	37	84	5	5	5.61
6	fast	high CPU	30	85	6	5	5.61
7	slow	high CPU	29	84	6	6	6.74
8	slow	high CPU	28	86	5	5	5.61
9	slow	low CPU	27	89	0	0	0
10	fast	high CPU	26	86	3	3	3.37
11	slow	high CPU	24	86	4	3	3.37
12	fast	high CPU	32	85	6	4	4.49
13	fast	low CPU	26	88	1	1	1.12
14	fast	high CPU	31	84	7	6	6.74

15	slow	high CPU	37	85	8	7	7.86
16	slow	high CPU	33	85	6	5	5.61
17	slow	high CPU	41	83	8	8	8.98
18	fast	high CPU	39	87	2	2	2.24
19	slow	high CPU	29	87	4	3	3.37
20	fast	high CPU	36	84	5	5	5.61
21	fast	high CPU	20	87	2	2	2.24
22	fast	high CPU	39	86	3	3	3.37
23	fast	low CPU	30	88	1	1	1.12
24	slow	high CPU	27	82	8	7	7.86

As the No. of nodes increased from 10 to 25 without changing other parameters, forks increased from 7 to 11.
We can conclude that, Increase in the nodes also leads to increase in forks.

Analysis :

Observations	Nodes	z_0	z_1	I	Time Taken	Forks
1	10	60	20	10 sec	945 sec	5
2	10	60	40	10 sec	1272 sec	4
3	10	50	80	60 sec	8530 sec	0
4	10	50	20	60 sec	6017 sec	1
5	10	50	20	30 sec	3059 sec	1
6	10	50	80	30 sec	4221 sec	1
7	10	50	20	5 sec	569 sec	7
8	25	50	20	5 sec	484 sec	11

- The forks and the Time taken to create a block chain of **100 blocks** are entirely dependent on the *block inter arrival time* , *no. of nodes* and *percentage of high CPU* nodes.
    - If we increase the block inter arrival time, forks will be reduced and the time taken will increase.
    - If we increase the no.of nodes or percentage of high CPU nodes, the forks get increased and the time taken decreases.

- Most of the blocks in the longest chain are created by the high CPU nodes.As they have higher hashing power than the Low CPU nodes they will crack the POW at lesser time compared to low cpu nodes. So every low cpu node will shift to the new longest chain before it cracks the POW.

When Interarrival block time : 10 sec

No.of high CPU nodes are 80% then Time Taken : 945 sec with 5 forks.
No.of high CPU nodes are 60% then Time Taken : 1272 sec with 4 forks
This shows a significant change in time with just 20% of more high CPU nodes.

When Interarrival block time : 60 sec

No.of high CPU nodes are 20% then Time Taken : 8530 sec with 0 forks
No.of high CPU nodes are 80% then Time Taken : 6017 sec with 1 fork.
This shows an increase in Forks and decrease in time taken with increase in no.of high CPU nodes.

When Interarrival block time : 30 sec .

No.of high CPU nodes are 20% then Time Taken : 4221 sec
No.of high CPU nodes are 80% then Time Taken : 3059 sec.
This shows that Time Taken has increased by a factor of n with an increase in Interarrival block time by n.

Low CPU nodes are not able to generate blocks more than 4% in the chain if most of the nodes are high CPU.

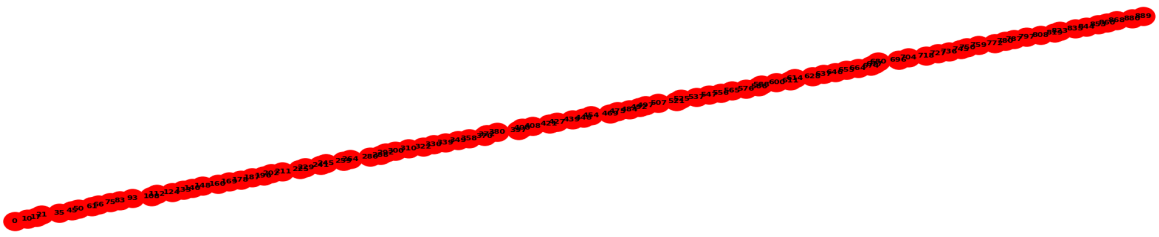
If 80% of nodes in the network are low CPU nodes, still 60% of blocks are generated by high CPU nodes(20% in Network size).

Therefore, Hashing Power influenced a lot in the generation of Blocks in Blockchain.

Visualization of Blockchains :

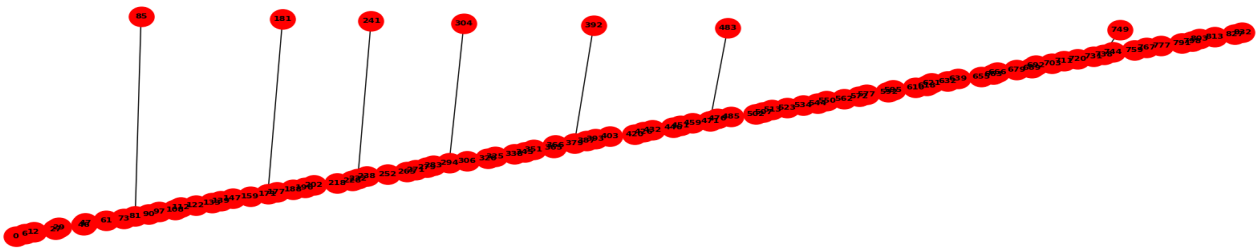
The Experimental values used for Visualization :

- Number of Nodes : 10
- Percentage of Slow Nodes (z_0) : 50%
- Percentage of Low_CPU_Nodes (z_1) : 80%
- Interarrival Block time (I) : 60 sec



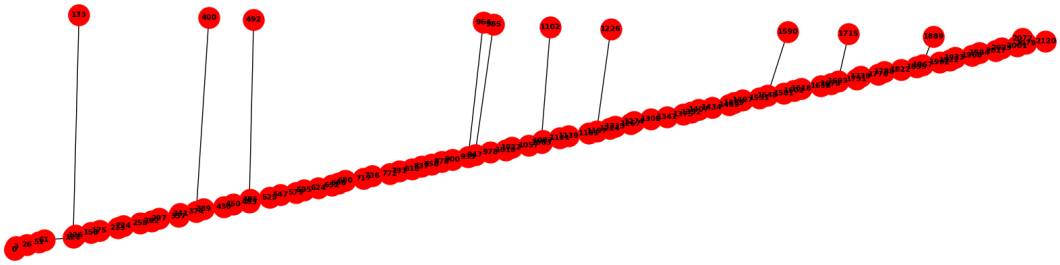
The Experimental values used for Visualization :

- Number of Nodes : 10
- Percentage of Slow Nodes (z_0) : 50%
- Percentage of Low_CPU_Nodes (z_1) : 20%
- Interarrival Block time (I) : 5 sec



The Experimental values used for Visualization :

- Number of Nodes : 25
- Percentage of Slow Nodes (z_0) : 50%
- Percentage of Low_CPU_Nodes (z_1) : 20%
- Interarrival Block time (I) : 5 sec



The blocks which are not aligned in the chain are Forks.
From above Visualizations, we can conclude that Number of nodes, Percentage of high CPU nodes and Interarrival block time plays a vital role in the occurrence of Forks.