CS 765 Assignment 1

Adish Shah, 200020012
Akshay Kiran, 200050093
Pinkesh Raghuvanshi, 200050106

February, 2023

Contents

1 Part 2. Reasons for choosing exp distribution

The reason for choosing the exponential distribution for the inter-arrival time between transactions generated by any node is the fact that it is memory-less, meaning that the probability of an event occurring in a given time interval only depends on the length of that interval, and not on when the previous event occurred. This property is a reasonable assumption for blockchain networks, where transactions are generated randomly and independently by nodes.

Another reason for using the exponential distribution in blockchain simulations is that it is a continuous distribution with a well-known mathematical form, which makes it easy to work with.

2 Part 4. Network Topology

This has been implemented in `topology.py`. We generate a random undirected graph, and check if it's connected. If not, we continue till we get a connected graph. The function returns a dictionary, which is then used to add peers to the nodes. Orange nodes represent fast nodes, and gray nodes represent slow nodes. We used networkx library for visualisation.

3 Part 5 Latencies

d_{ij} represents the queuing delay at node i to forward the message to node j, and c_{ij} is the link speed between i and j. The mean of d_{ij} is inversely related to c_{ij} because, greater the link speed, lesser will be the queuing delay.

Inverse proportionality between queuing delay and link speed means that as the link speed increases, the queuing delay decreases. This is because a higher link speed increases the capacity of the network to transmit data, so there is less likelihood of congestion and less likelihood of packets having to wait in a queue to be transmitted.

The plots have been genereted by the script `plot.py`, which I have included in my submission. It requires that a `plots` directory be made before running this, as all the relevant statefiles, mdps and random policies (`rand_pol.txt` extrapolated to the set of required states) that I generate are placed inside this directory. The following are the graphs plotted, which include 2 lines - one for the optimal policy and another for the arbitrary policy provided.

4 Part 6 Loop less forwarding

To ensure this we have a dictionary `p2p_history`, which adds the transactions and blocks received by other blocks and the current node forwards it to them only if the object isn't present in the `p2p_history`.

5 Part 7 Simulating PoW

The nodes have a field called `frac_cpu_power`, which is set to 1 by default. While initialising the nodes, z1 (frac of High CPU) is used, to set these nodes' `frac_cpu_power` to 10 (10 times slow cpu). Then we calculate the total hashing power by adding these values (10 for high cpu node and 1 for low cpu node) and then the final hashing fraction is set by dividing `frac_cpu_power/total_hashing_power`. The T_k is sampled from a mean which is set by dividing the $T_{overall}(= b)$ by the node's `frac_cpu_power`.

6 Part 8 Visualisation and analysis

We will use the following parameters to analyse different cases:

- -n or –nodes: Number of nodes in the peer to peer network. Set to 10 by default
- -e or –endtime: End time of simulation in terms of simulation time (which is equivalent to seconds). Default value 1000
- -t or –interarrival: Mean interarrival time (in s) of transactions. Default value is 1
- -z0 or –fracslow: Fraction of slow nodes in the network. By default set to 0.3
- -z1 or –frac-powerful : Fraction of nodes with high cpu power. Default value is 0.5.
- -b or –block-interarrival: Interarrival time between the blocks. Default value is 5.

Note on visualisation

Visualisation was done using networkx library in Python. We started out by making png files for graphs, but as we went ahead and number of blocks increased the png files became useless (due to large amount of cluttering). So we switched to creating dot files which provided clean viewing for large graphs. The graph attached below is an example of a typical blockchain generated when the following paramters are set :

```
python main.py -n 5 -e 50 -t 0.5 -z0 0.1 -b 3.5 -z1 0.6
```

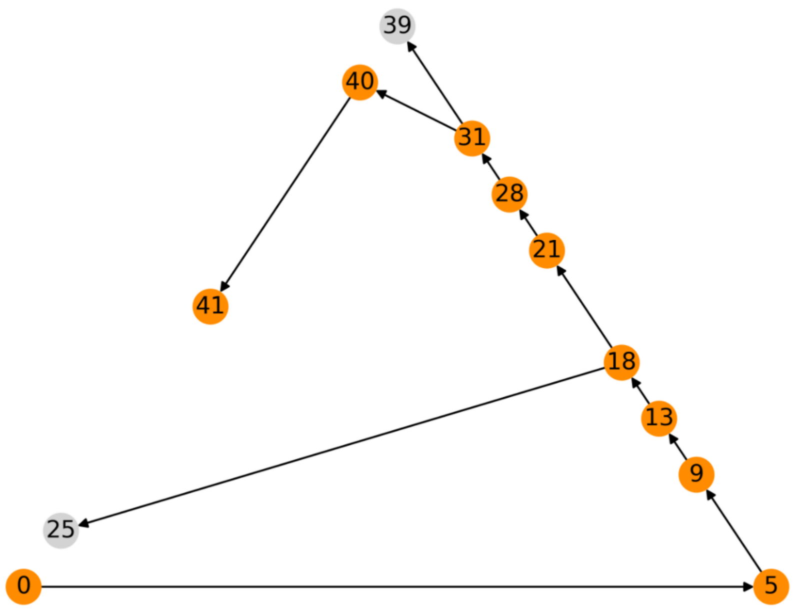


Figure 1: A Typical blockchain generated when the number of blocks is small. When the number of blocks is large the image cannot be fit into a png file and dot files need to be used. Orange colored nodes represent longest chain and greyed out nodes represent orphaned blocks

Verifying correctness of generated blockchain

As we have seen above the given blockchain visualisation did not give sufficient context, we switched to dot files.

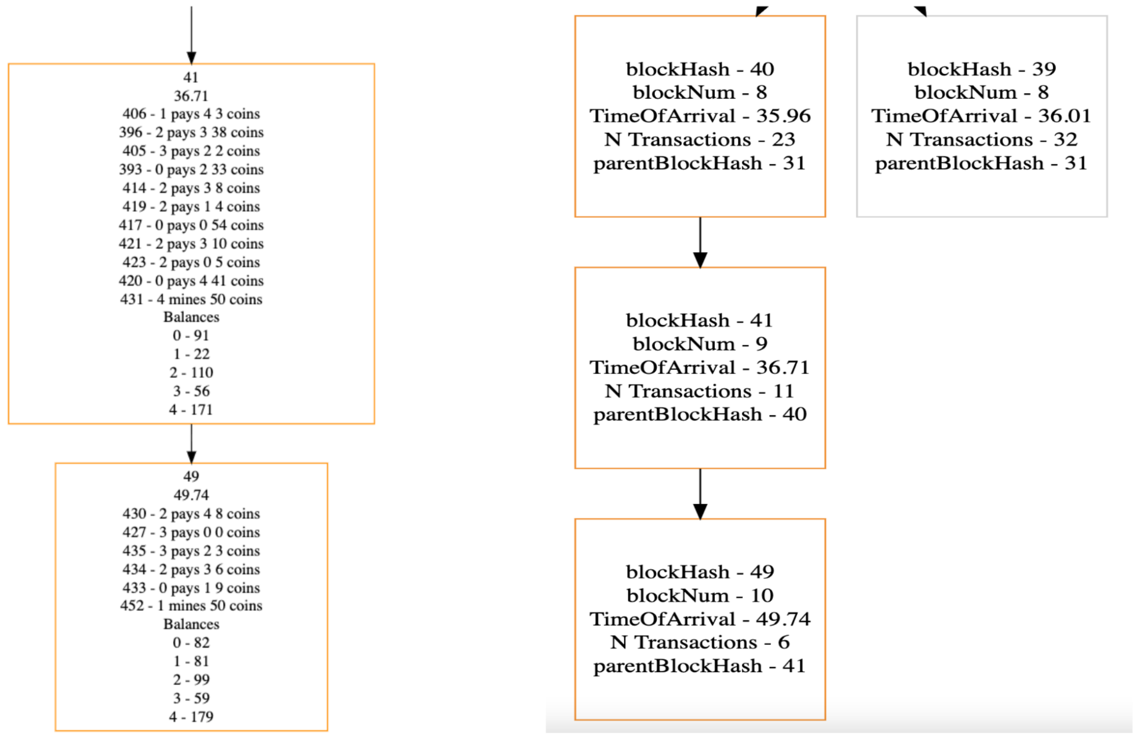


Figure 2: The above figures represent a detailed view of block that can be used to verify correctness. The balances in the left block represent the balances left after making the transactions of this block

Using the above visualisations, we ensured that our code is working correctly e.g.. there are **No Invalid Transactions**, in any block. Also, we gauged the performance of our blockchain wrt hyperparemeters (like Tk ,Ttk etc) using these metrics. This also allowed us to visualise forks and see how the ties are being broken. We observed that there is no concrete rule of breaking forks when the time of arrivals are similar, but earlier block is preferred if difference is large.

The above graphs were generated for each node and they were found to be similar (except for the case where due to propagation delay a block could not reach the a particular node) Additionally, an analysis file was generated for each node containing the following data -

- Fast/Slow characteristic of the node
- The mean value for Tk and the fraction of hashing power possessed by the node
- Ratio of number of blocks generated by the node that are in longest chain to the total number of blocks in longest chain
- Branch lengths of the tree maintained by the node

6.1 Analysis of ratio, while altering the slow nodes ratio z0

Values of parameters: n = 20, e = 400, t = 0.5, z1 = 0.6

z0 = 0.1,b = 5

Only one fork was observed and the branch lengths were 60 and 63.

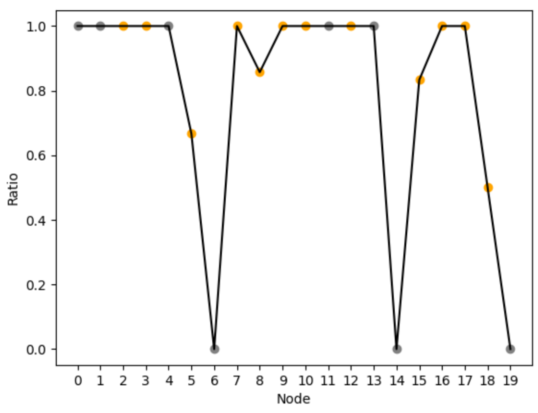


Figure 3: Ratio of successful mines by each node

z0 = 0.4,b = 5

In this case 5 branches were observed with branch lengths [15, 27, 28, 37, 82]

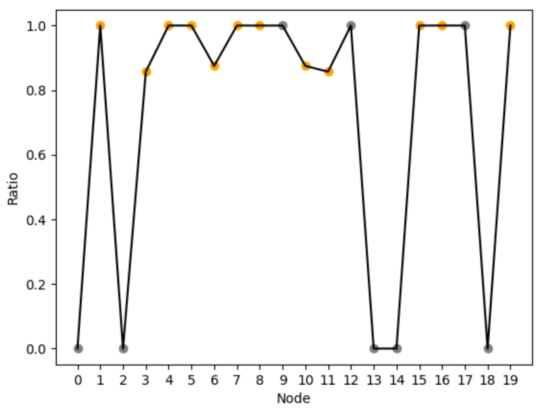


Figure 4: Ratio of successful mines by each node

z0 = 0.6,b = 5

In this case 3 branches were observed with branch lengths 61, 71, 77

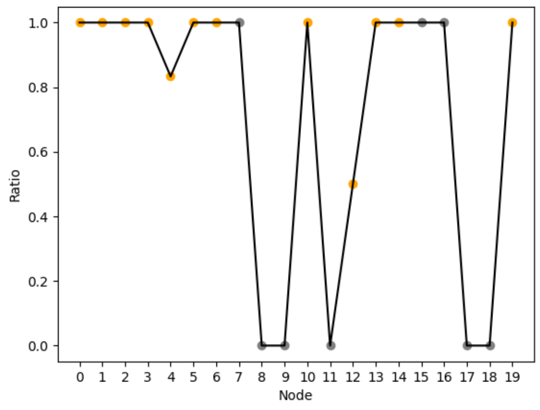


Figure 5: Ratio of successful mines by each node

In the above figures, we can observe that the number of slow nodes in the network will definitely increase the latencies and the time of propagation of a block or a transaction, it doesn't significantly affect the number of nodes with ratios ¡ 1 (essentially the number of forks), as long as the interarrival time between blocks in the network ($T_{overall}$) is at a large enough safe value. If only $T_{overall}$ were very small and comparable to the propagation times, an increase in z0 would have increased the number of forks and may have even led to a chaotic situation because of the increased propagation times. Note : Whether a particular node is fast or slow won't really affect the fraction of the block it created which got into the longest chain, because the delay at that particular node will have only a small role to play in the total propagation delay of a block in the vast P2P network. Also, it won't matter because the interarrival time is large enough to be affected by small fluctuations in latencies during block propagation.

6.2 Analysis of ratio, while altering the block interarrival time b(T overall)

1. Now, keeping z0 = 0.4, we decrease the value of block interarrival time to see the changes in ratio. We set b = 4, and run the simulation: In this case 4 branches were observed with lengths [12, 42, 42, 76]

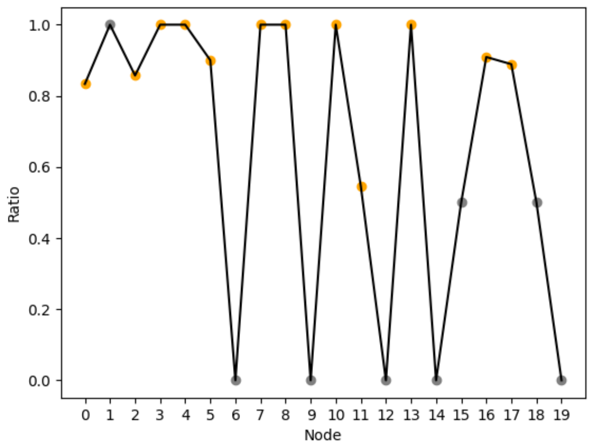


Figure 6: Ratio of successful mines by each node

2. Now, we set b=2.5, keeping other parameters same. In this case 24 branches were observed with lengths list = [15, 20, 24, 29, 31, 47, 49, 61, 64, 68, 77, 77, 92, 104, 109, 110, 114, 126, 134, 136, 145, 145, 149, 149]

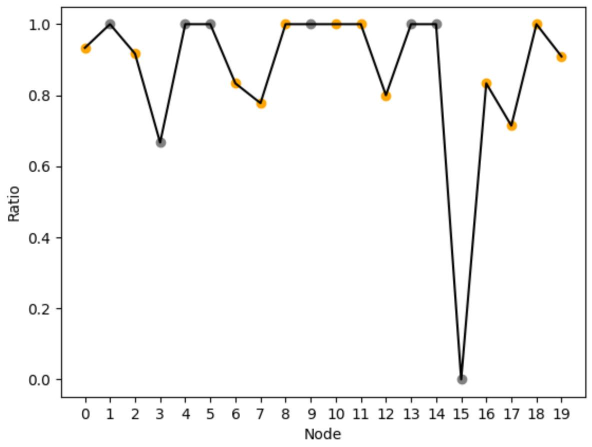


Figure 7: Ratio of successful mines by each node

3. b = 1, keeping other parameters same. In this case many branches were observed with lengths list = [5, 15, 16, 20, 22, 33, 35, 35, 36, 38, 44, 51, 59, 61, 61, 62, 63, 72, 75, 77, 81, 81, 82, 85, 87, 90, 95, 95, 97, 100, 101, 102, 107, 114, 118, 122, 124, 125, 125, 126, 127, 130, 132, 134, 137, 139, 142, 146, 150, 153, 159, 160, 161, 163, 165, 179, 182, 182, 190, 197, 205, 208, 211, 218, 219, 220, 223, 224, 228, 234, 237, 244, 245, 251, 251, 252, 257, 259, 259, 260, 262, 264, 267, 277, 282, 289, 291, 299, 304, 310, 310, 323, 325]

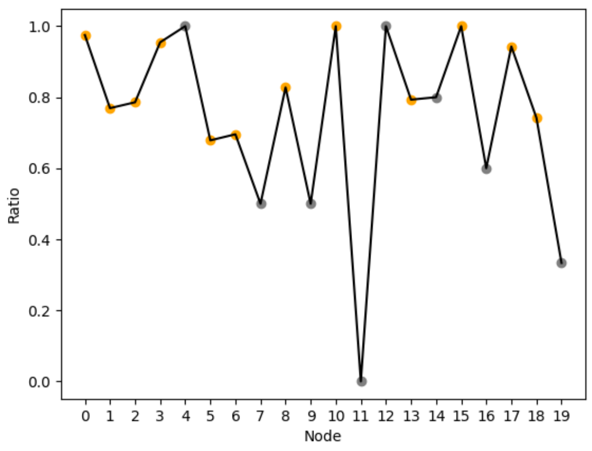


Figure 8: Ratio of successful mines by each node

As we can see in the above results (by varying expected interarrival times), when the inter- arrival time was large enough (b = 4) as compared to the propagation delays for the blocks, almost all the blocks generated by the nodes got into the longest chain.

When the interarrival time was significantly reduced (b = 2.5) and somewhat comparable to many propagation delays of the blocks, there are significant number of forks, and half of the nodes had a number of their blocks mined outside the main longest chain.

But, when the interarrival time became significantly small (b = 1), and very much comparable to most of the propagation delays, it lead to a chaotic situation where we observed a lot of forks and branches and that most of the nodes got a lot of blocks outside the final longest chain.

6.3 Plot: Branches vs Block Interarrival time(b)

Alongside, the above cases, effect of changing block interarrival time was also analysed for small number of nodes(5) and simulation time, (with params values as n=5, e=50, z0 = 0.1, z1 = 0.6 and t=0.5), for different values of b and a plot was generated. The experiment was repeated 5 times for each value of b and the value on y axis was taken to be the average of those 5 values

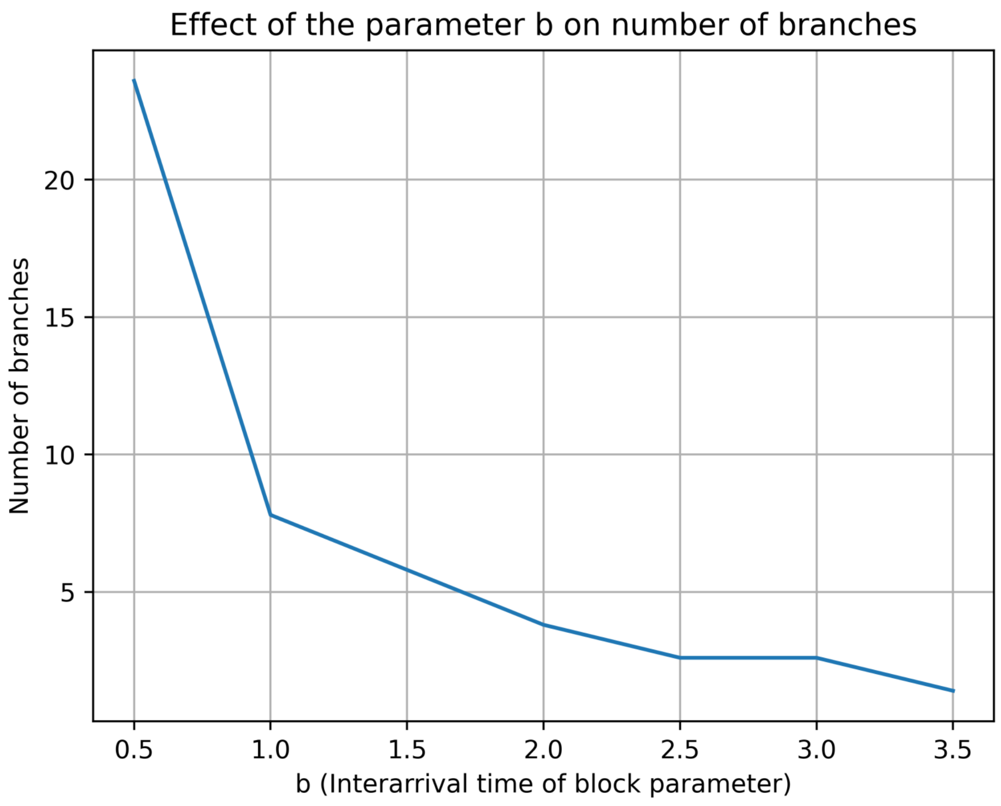


Figure 9: Plot: Branches in longest chain vs b

The result we obtain is pretty much expected because as the interarrival time of blocks in- creases, the probability of two blocks coming at the same time will reduce, thereby reducing forks and number of branches.

6.4 Effect of CPU power

Now, keeping z= 0.4, b= 4, we make z1 = 0.1, meaning 1 node is high CPU and has majority of the hashing power.

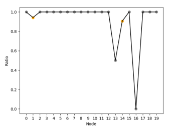


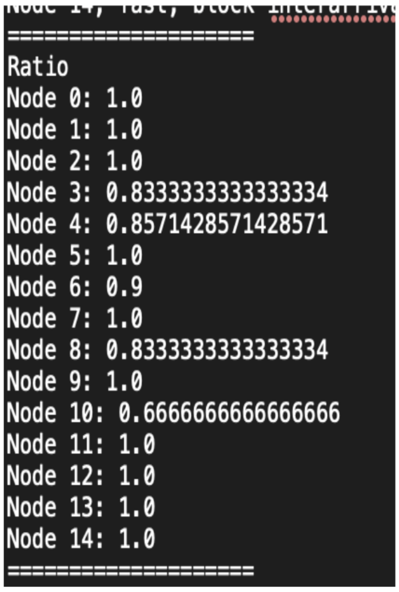
Figure 10: Setting z1=0.1

While we must concede that this is not a drastic manifestation of CPU power, it is true that blocks will be generated by the powerful miners at a much higher rate than the others, which means a higher chance of the other nodes of not being able to create blocks in the longest chain, and of them creating forks (which may be instantly suppressed by the powerful nodes). The simulation generated total of 88 blocks out of which the following was the distribution.

```
0 : 3
1 : 35
2 : 3
3 : 2
4 : 4
5 : 3
6 : 5
7 : 2
8 : 4
9 : 4
10 : 1
11 : 3
12 : 1
13 : 2
14 : 21
15 : 2
16 : 0
17 : 2
18 : 1
19 : 1
```

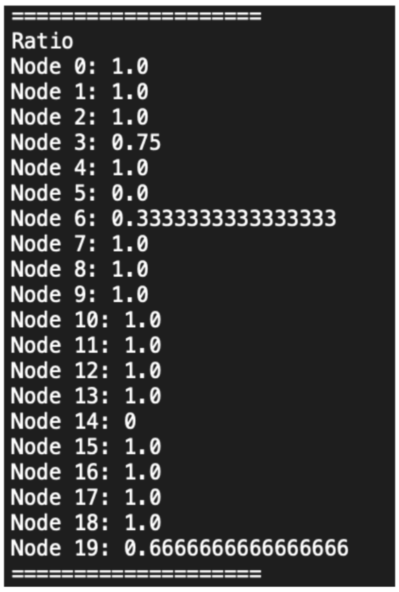
6.5 Effect of number of nodes

We vary the number of nodes keeping other parameters fixed to the following values e = 400 , z0=0.1, t = 0.5, b=5, z1 = 0.6



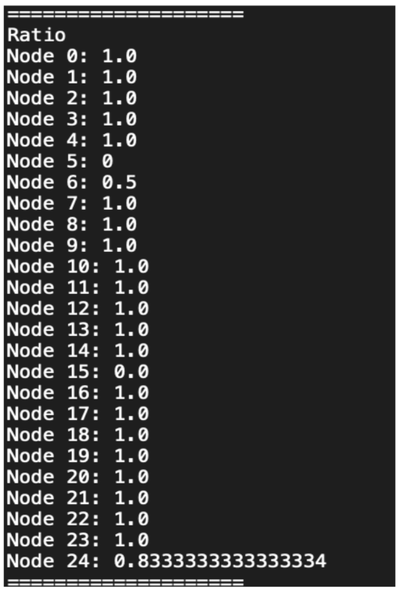
```
Node 14, fast, block interarriv
====================
Ratio
Node 0: 1.0
Node 1: 1.0
Node 2: 1.0
Node 3: 0.8333333333333334
Node 4: 0.8571428571428571
Node 5: 1.0
Node 6: 0.9
Node 7: 1.0
Node 8: 0.8333333333333334
Node 9: 1.0
Node 10: 0.6666666666666666
Node 11: 1.0
Node 12: 1.0
Node 13: 1.0
Node 14: 1.0
====================
```

(a) **n = 15**. Branching: [8, 13, 30, 52, 73, 82]



```
====================
Ratio
Node 0: 1.0
Node 1: 1.0
Node 2: 1.0
Node 3: 0.75
Node 4: 1.0
Node 5: 0.0
Node 6: 0.3333333333333333
Node 7: 1.0
Node 8: 1.0
Node 9: 1.0
Node 10: 1.0
Node 11: 1.0
Node 12: 1.0
Node 13: 1.0
Node 14: 0
Node 15: 1.0
Node 16: 1.0
Node 17: 1.0
Node 18: 1.0
Node 19: 0.6666666666666666
====================
```

(b) **n=20**, Branching: [6, 14, 52, 61, 66, 68]



```
====================
Ratio
Node 0: 1.0
Node 1: 1.0
Node 2: 1.0
Node 3: 1.0
Node 4: 1.0
Node 5: 0
Node 6: 0.5
Node 7: 1.0
Node 8: 1.0
Node 9: 1.0
Node 10: 1.0
Node 11: 1.0
Node 12: 1.0
Node 13: 1.0
Node 14: 1.0
Node 15: 0.0
Node 16: 1.0
Node 17: 1.0
Node 18: 1.0
Node 19: 1.0
Node 20: 1.0
Node 21: 1.0
Node 22: 1.0
Node 23: 1.0
Node 24: 0.8333333333333334
====================
```

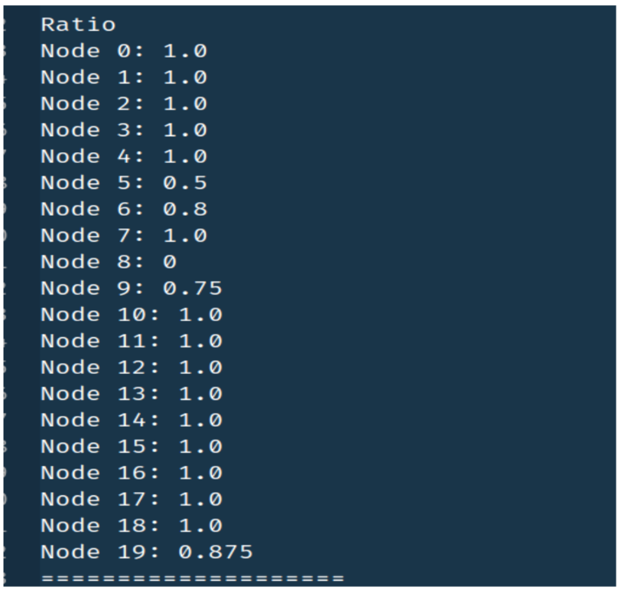
(c) **n=25**, Branching: 46, 49, 52, 87

Figure 11: Figure representing successful block ratio for each node and fork with varying number of nodes and other parameters fixed

It is easy to observe that having more nodes (a larger network) may somewhat insignificantly increase the chances of a node mining on the wrong block, but it doesn't matter overall, as long as the interarrival time has been suitably set. Mathematically, for a fixed $T_{overall}(= b)$, Tk depends on the fraction of CPU power with the k-th node, which will decrease as more and more nodes join the network. Since the increase in propagation delay will happen at a much slower rate, and we have chosen $T_{overall}$ to be large enough to give us enough cushion space, we can safely expect and observe that only a few forks are created in the blockchain.

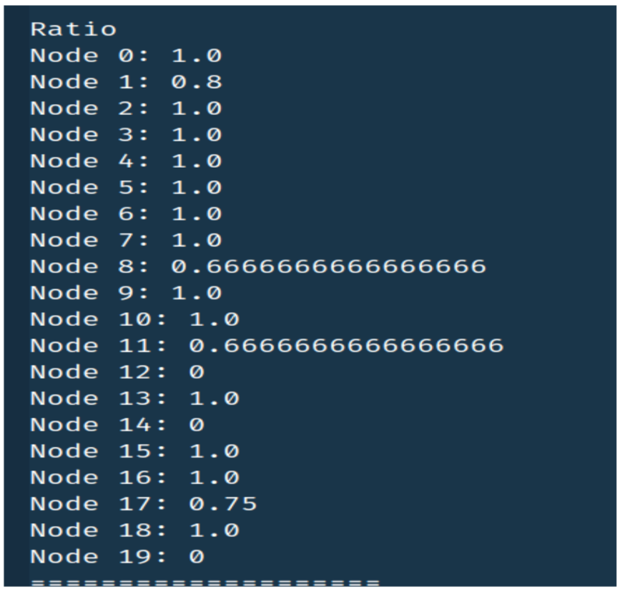
6.6 Analysing behaviour based on Mean transaction time(t)

We vary the interarrival time of transactions keeping other parameters fixed to the following values n = 20, e = 400 , z0 = 0.1, b=5, z1 = 0.6



```
Ratio
Node 0: 1.0
Node 1: 1.0
Node 2: 1.0
Node 3: 1.0
Node 4: 1.0
Node 5: 0.5
Node 6: 0.8
Node 7: 1.0
Node 8: 0
Node 9: 0.75
Node 10: 1.0
Node 11: 1.0
Node 12: 1.0
Node 13: 1.0
Node 14: 1.0
Node 15: 1.0
Node 16: 1.0
Node 17: 1.0
Node 18: 1.0
Node 19: 0.875
====================
```

(a) **t=0.5**. Branching: [39, 58, 70, 79, 80]



```
Ratio
Node 0: 1.0
Node 1: 0.8
Node 2: 1.0
Node 3: 1.0
Node 4: 1.0
Node 5: 1.0
Node 6: 1.0
Node 7: 1.0
Node 8: 0.6666666666666666
Node 9: 1.0
Node 10: 1.0
Node 11: 0.6666666666666666
Node 12: 0
Node 13: 1.0
Node 14: 0
Node 15: 1.0
Node 16: 1.0
Node 17: 0.75
Node 18: 1.0
Node 19: 0
====================
```

(b) **t=2**, Branching: [3, 23, 37, 42, 61]

Figure 12: Varying t

For parameters values, n=5, e=50, z0=0.1, z1=0.6 and b=5, plot of number of transactions in longest chain vs t was generated, which shows the trend that, increasing t, decreases the number of transactions in longest chain. The experiment was done 5 times with the above values and the average values obtained are plotted on y axis

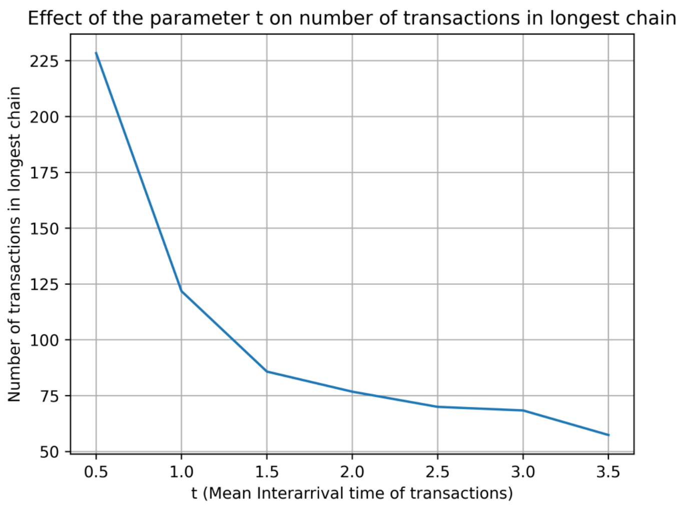


Figure 13: Plot: Transactions in longest chain vs t