CS-765 Assignment-1 Report

Simulation of P2P Cryptocurrency Network

200050019 - Bagathi Shiv Kiran
200050020 - Bale Teja Rama Chandra Murthy
200050075 - Moganti Harshadeep

Theoretical Explanations :

1. What are the theoretical reasons for choosing exponential distributions for transaction interarrival time?

Reason: Let us consider a time interval Δ. Since we know that the transactions have an equal likelihood to occur at any moment in time, the probability for a transaction to occur in a time interval Δ is directly proportional to it. Let's call it $\beta\Delta$.

$$\Rightarrow P(txn\ arriving\ at\ time\ \Delta) = \beta\Delta$$

$$\Rightarrow P(I = \Delta) = \beta\Delta$$

Let us have a time interval of T = nΔ. Then the probability of having a transaction arrive at time nΔ interval is given by,

$$P(I = n\Delta) = (1 - P(I = \Delta))^{n-1} * P(I = \Delta)$$

[Since, the prob. for txn not arriving in each interval Δ can be treated independently]

$$\Rightarrow P(I = n\Delta) = (1 - \beta\Delta)^{n-1} * \beta\Delta$$

$$\Rightarrow P(I > n\Delta) = (1 - \beta\Delta)^{n}$$

Let us have x = nΔ,

$$\Rightarrow P(I > x) = (1 - \beta\frac{x}{n})^{n}$$

As we have larger intervals T i.e., as n $\rightarrow\infty$,

$\lim_{n \to \infty} P(I > x) = \lim_{n \to \infty} (1 - \beta\frac{x}{n})^n = e^{-\beta x}$ which is an exponential distribution.

Exponential distribution has memory less property. For simulation in a decentralized peer-to-peer network we need to generate transactions independent of other peers. Exponential distribution being memory less helps us to generate future inter arrival time independent of past times.

2. Why is the mean of d_{ij} inversely related to c_{ij}?

Reason: Here d_{ij} denotes the queuing delay at node i to forward the message to node j in seconds, c_{ij} denotes the link speed between i and j in bits per second. We can observe that queuing delay for a packet will be higher if there are more packets waiting in the queue at node i, which will be higher if the packets are sent between nodes at a slower rate which means having lower link speed. Thus, we can see that d_{ij} is inversely related to c_{ij}.

To show it mathematically, let us assume there are n packets waiting in queue before our packet at node i to be sent to node j, size of each packet is m bits,

$$d_{ij}(sec) = \frac{Number\ of\ packets\ in\ the\ queue\ before\ our\ packet \times Size\ of\ each\ packet(bits)}{c_{ij}(bits/sec)}$$

$$\Rightarrow d_{ij}(sec) = \frac{nm}{c_{ij}(bits/sec)}$$

So, we prove that d_{ij} is inversely related to c_{ij}.

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

Reason: Given that we have to sample the T_k from the exponential distribution having mean mean I/h_k. Where h_k is the inherent property of

the Peer which is determined by initially random allocation of low_CPU Peers. Our only control is with hyperparameter I.
Ideally we want to minimize the forking of the blockchain, for that we need to have use suitable I. From the given constraints we get
$h_k = 1/(n + 9 * highCpu_{count})$ for low_CPU and
$h_k = 10/(n + 9 * highCpu_{count})$ for high_CPU

In our given Latency conditions, we get the values of latencies in the range of around 10, 500 milliseconds. From the reference of Satoshi's Bitcoin paper, where he considered for Network Delays of ~ 10 sec, the average block mining time to be around 10 minutes (600 seconds).

This is because, if block mining time has order sufficiently larger than the network latencies, a single successfully mined block is likely to be propagated to all the nodes before multiple nodes succeed in mining. This reduces the likelihood of forks in the blockchain.

For our case we can hence assume the average block mining time I/h_k to be in the order of 6000 milliseconds to 10,000 milliseconds. Assuming h_k to be in the order of 1/20. We can have I around 1000 - 5000 milliseconds for optimal Blockchain.

Experiments and Insights:

Parameters we are varying in our experiments :

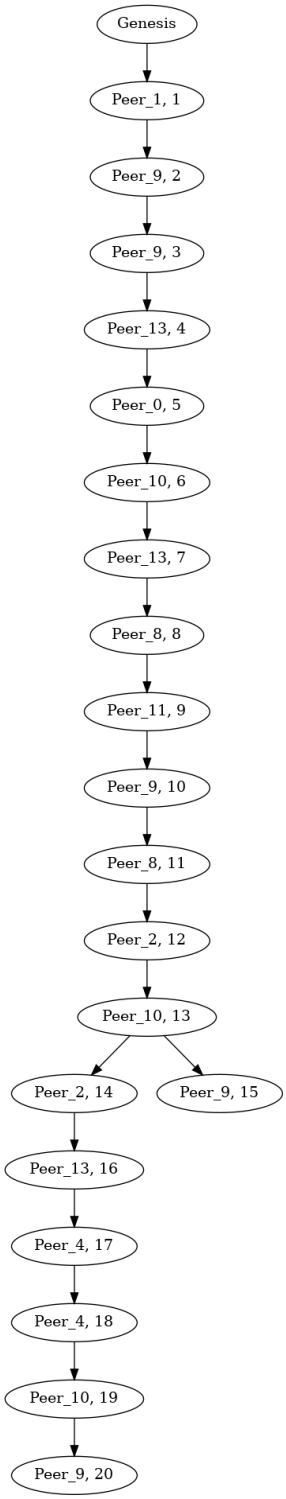
n : number of peers in the network
z0 : percentage of slow peers
z1 : percentage of peers having low CPU power
ttx : mean inter-arrival time between transactions
I : average time taken to mine a block
save : save the events in the file

Example 1 :

n=15, z0=50, z1=50, ttx=10, I=3000

```
Ratio of blocks mined by Peer_0 of type low_slow that made it to the longest chain: 1.0
Ratio of blocks mined by Peer_1 of type high_fast that made it to the longest chain: 1.0
Ratio of blocks mined by Peer_2 of type high_slow that made it to the longest chain: 1.0
Ratio of blocks mined by Peer_3 of type low_fast that made it to the longest chain: None
Ratio of blocks mined by Peer_4 of type high_slow that made it to the longest chain: 1.0
Ratio of blocks mined by Peer_5 of type low_slow that made it to the longest chain: None
Ratio of blocks mined by Peer_6 of type low_fast that made it to the longest chain: None
Ratio of blocks mined by Peer_7 of type low_slow that made it to the longest chain: None
Ratio of blocks mined by Peer_8 of type high_fast that made it to the longest chain: 1.0
Ratio of blocks mined by Peer_9 of type high_fast that made it to the longest chain: 0.8
Ratio of blocks mined by Peer_10 of type high_slow that made it to the longest chain: 1.0
Ratio of blocks mined by Peer_11 of type low_fast that made it to the longest chain: 1.0
Ratio of blocks mined by Peer_12 of type high_fast that made it to the longest chain: None
Ratio of blocks mined by Peer_13 of type high_fast that made it to the longest chain: 1.0
Ratio of blocks mined by Peer_14 of type low_slow that made it to the longest chain: None
Average ratio of blocks in longest chain mined by low_slow node: 1.0
Average ratio of blocks in longest chain mined by high_slow node: 1.0
Average ratio of blocks in longest chain mined by low_fast node: 1.0
Average ratio of blocks in longest chain mined by high_fast node: 0.95
```

We are choosing I in range of 1000-5000, as we've discussed in the previous question, choosing I in this range keeps the blockChain very efficient with less forks. The BlockMining Time is large enough to exceed Network Latencies, this ensures to keep Peers informed about the new MinedBlocks before they are done with their hence updating the LongChainBlockHead. Therefore, we are getting a BlockChain with minimal branches.



BlockChain of Peer_0

```
Peer_0 is of type low_slow
Peer Block Details:{5}dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 20])dict_keys([])dict_keys([0, 1, 2, 3,
4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
Length of longest chain (including genesis block):20
Longest chain:[20, 19, 18, 17, 16, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5,
4, 3, 2, 1, 0]
Total number of blocks at by Peer_0 : 20
Fraction of longChain to Total Blocks 0.9523809523809523
Ratio of blocks mined by Peer_0 that made it to the longest chain: 1.0
```

Almost all blocks of each type are mined into the longest Chain. Reason for high_slow is similar to the above discussion where high_CPU node is generating the block before the blocks from neighboring Peers are reached due to slow_speed.
The length of branches in this blockchain(in blocks) = 1 (out of 20).

Example 2:

n=15, z0=50.0, z1=50.0, ttx=10, I=100

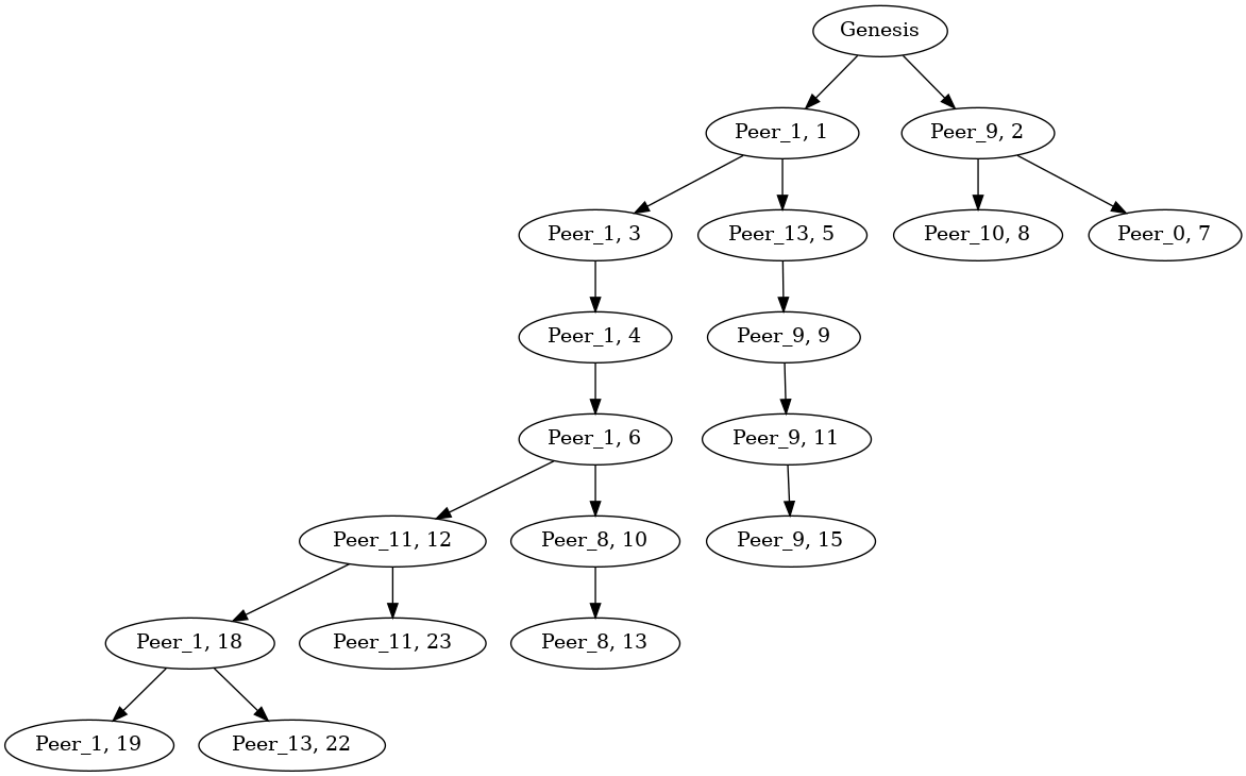
Forks at almost all the blocks in the longest chain. Low block interarrival time, so the blocks from other nodes are not reached before the current Nodes is mining the block.

```
Peer_1 is of type high_fast
Peer Block Details:{1, 3, 4, 6, 18, 19}dict_keys([0, 1, 3, 4, 5, 2, 6, 8, 7, 12, 10, 18, 19,
9, 11, 23, 13, 15, 22])dict_keys([17])dict_keys([0, 1, 3, 4, 5, 2, 6, 8, 7, 12, 10, 18, 19, 9,
11, 23, 13, 15, 22, 17])
Length of longest chain (including genesis block):8
Longest chain:[19, 18, 12, 6, 4, 3, 1, 0]
Total number of blocks at by Peer_1 : 24
Fraction of longChain to Total Blocks 0.32
Ratio of blocks mined by Peer_1 that made it to the longest chain: 1.0
```

The Blocks from High_CPU are generating more blocks and as the network connection is faster, blocks are always arriving to it before it can

generate a new Block. Even if it generates it loses in fork resolution.

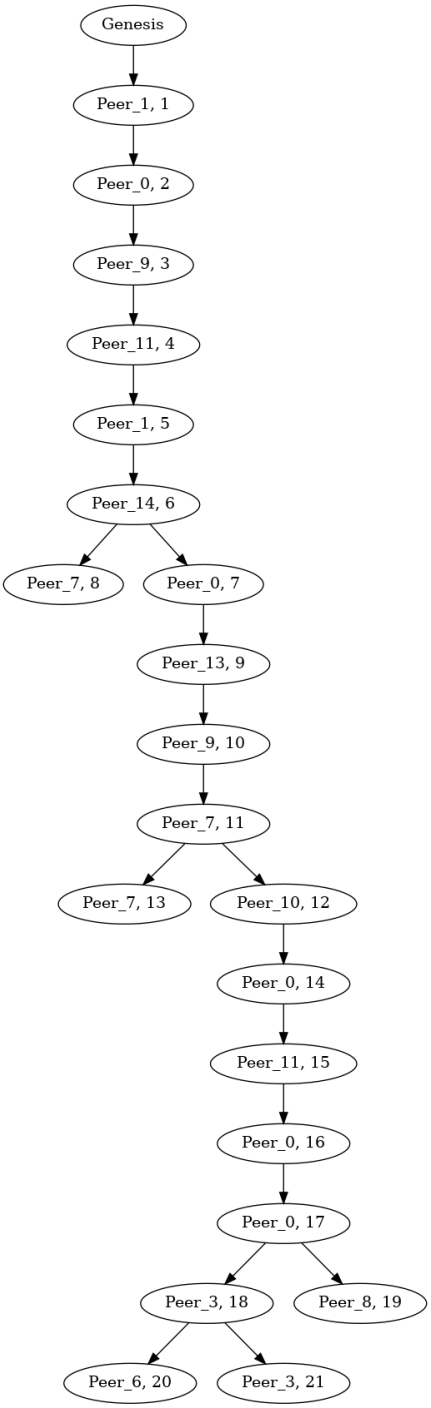
```
Ratio of blocks by Peer_0 of low_slow in the longest chain: 0.5
Ratio of blocks by Peer_1 of high_fast in the longest chain: 0.857
Ratio of blocks by Peer_2 of high_slow in the longest chain: 0.333
Ratio of blocks by Peer_3 of low_fast in the longest chain: None
Ratio of blocks by Peer_4 of high_slow in the longest chain: 1.0
Ratio of blocks by Peer_5 of low_slow in the longest chain: None
Ratio of blocks by Peer_6 of low_fast in the longest chain: None
Ratio of blocks by Peer_7 of low_slow in the longest chain: None
Ratio of blocks by Peer_8 of high_fast in the longest chain: 0.0
Ratio of blocks by Peer_9 of high_fast in the longest chain: 1.0
Ratio of blocks by Peer_10 of high_slow in the longest chain: 0.5
Ratio of blocks by Peer_11 of low_fast in the longest chain: 0.0
Ratio of blocks by Peer_12 of high_fast in the longest chain: 1.0
Ratio of blocks by Peer_13 of high_fast in the longest chain: 0.6
Ratio of blocks by Peer_14 of low_slow in the longest chain: None
Average ratio of blocks in longest chain mined by low_slow node: 0.5
Average ratio of blocks in longest chain mined by high_slow node: 0.611
Average ratio of blocks in longest chain mined by low_fast node: 0.0
Average ratio of blocks in longest chain mined by high_fast node: 0.691
```



BlockChain at Peer 1

Example 3:

n=15, z0=0.0, z1=0.0, ttx=10.0, I=1000

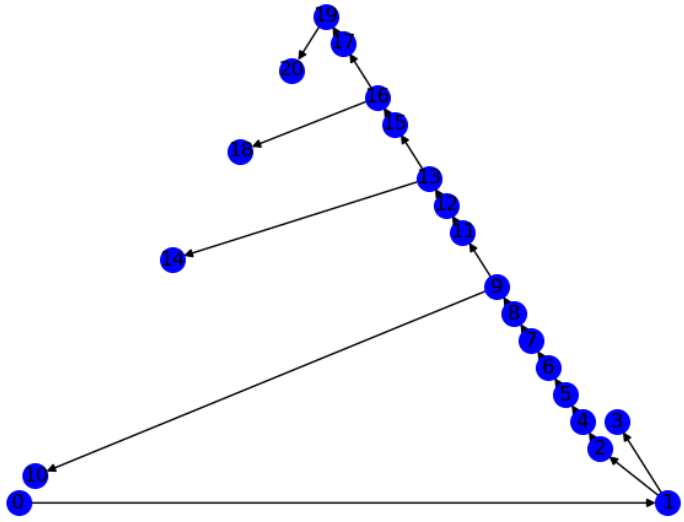


Block chain at Peer 7

Side branches and their lengths are relatively higher. Might depend on z0,z1

```
Peer_7 is of type high_fast
Peer Block Details:{8, 11, 13}dict_keys([0, 1, 2, 3, 4, 5, 6, 8, 7, 9, 10, 11, 13, 12, 14, 15
16, 17, 18, 19, 20, 21])dict_keys([])dict_keys([0, 1, 2, 3, 4, 5, 6, 8, 7, 9, 10, 11, 13, 12,
14, 15, 17, 16, 18, 19, 20, 21])
Length of longest chain (including genesis block):18
Longest chain:[20, 18, 17, 16, 15, 14, 12, 11, 10, 9, 7, 6, 5, 4, 3, 2, 1, 0]
Total number of blocks at by Peer_7 : 21
Fraction of longChain to Total Blocks 0.8181818181818182
Ratio of blocks mined by Peer_7 that made it to the longest chain: 0.333
```

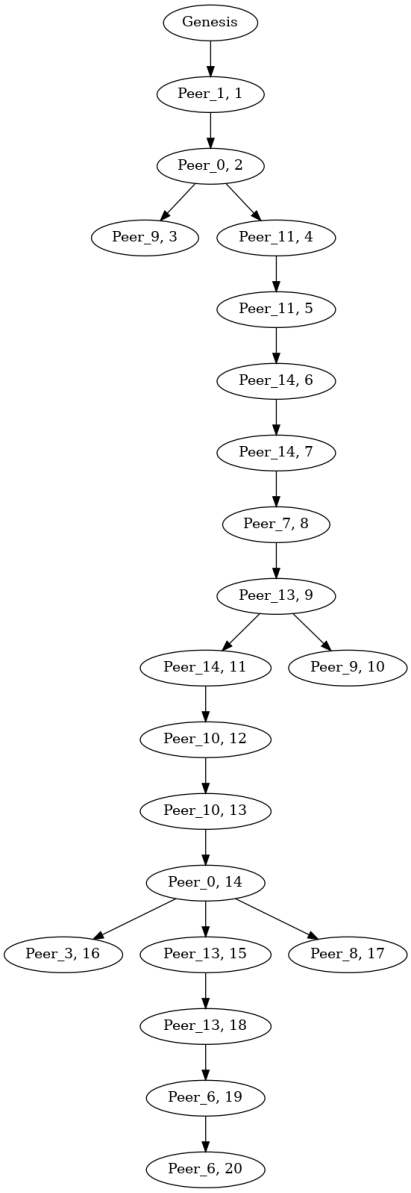
In this case we are choosing all the Peers to be high_CPU and fast Network speeds. Here we are getting higher branching than the first where we had higher Block Interarrival time. In this scenario, as we are taking both as high_CPU and fast Networks we are getting relatively lower than when we consider a percentage of low_CPU and fast Network.



BlockChain formed when n=15, z0=25.0, z1=0.0, ttx=100.0, I=1000.0

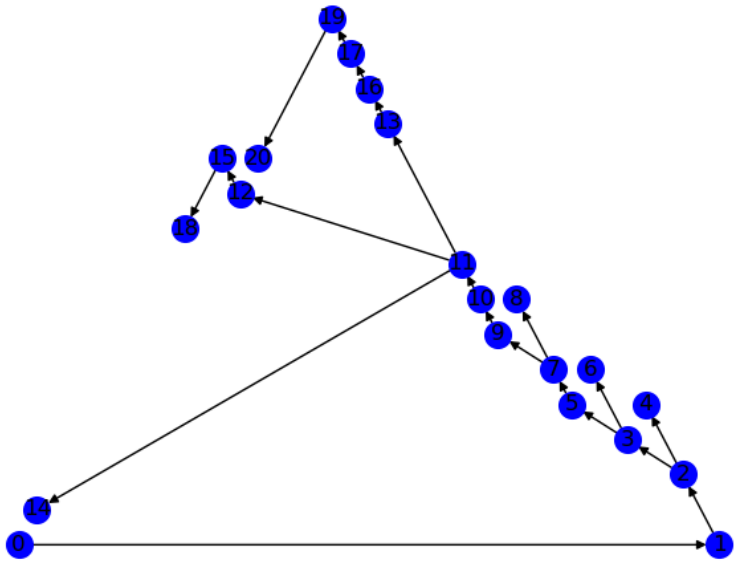
Example 4:

n=15, z0=100.0, z1=100.0, ttx=10.0, I=1000



Blockchain at Peer 10

```
Peer_10 is of type low_slow
Peer Block Details:{12, 13}dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 10, 12, 13, 14, 16,
15, 17, 18, 19, 20])dict_keys([])dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 10, 12, 13, 14,
16, 15, 17, 18, 19, 20])
Length of longest chain (including genesis block):17
Longest chain:[20, 19, 18, 15, 14, 13, 12, 11, 9, 8, 7, 6, 5, 4, 2, 1, 0]
Total number of blocks at by Peer_10 : 20
Fraction of longChain to Total Blocks 0.8095238095238095
Ratio of blocks mined by Peer_10 that made it to the longest chain: 1.0
```

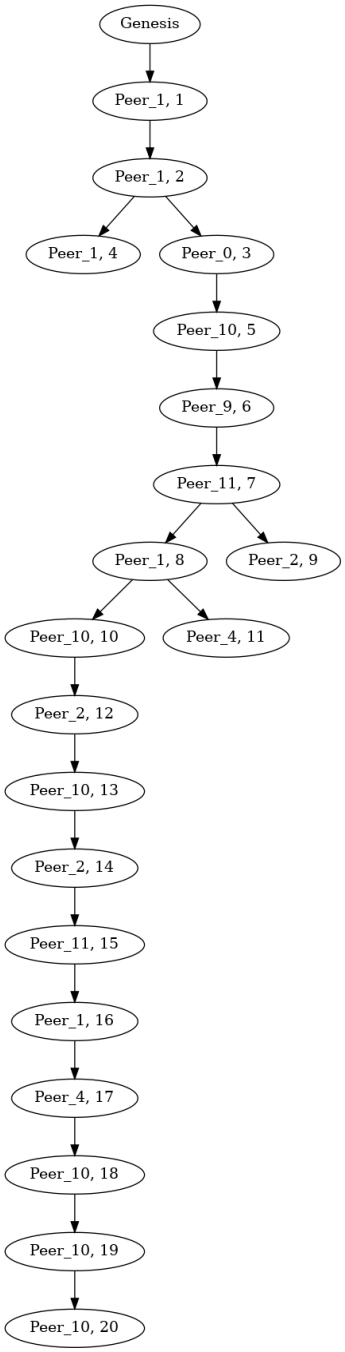


Blockchain for n=15, z0=100.0, z1=100.0, ttx=100.0, I=1000.0
Here comparing with ttx = 100, we can observe that keeping low ttx is almost changing behavior into low_CPU and fast Network speeds for transactions, that the former have less forks when compared to the latter.

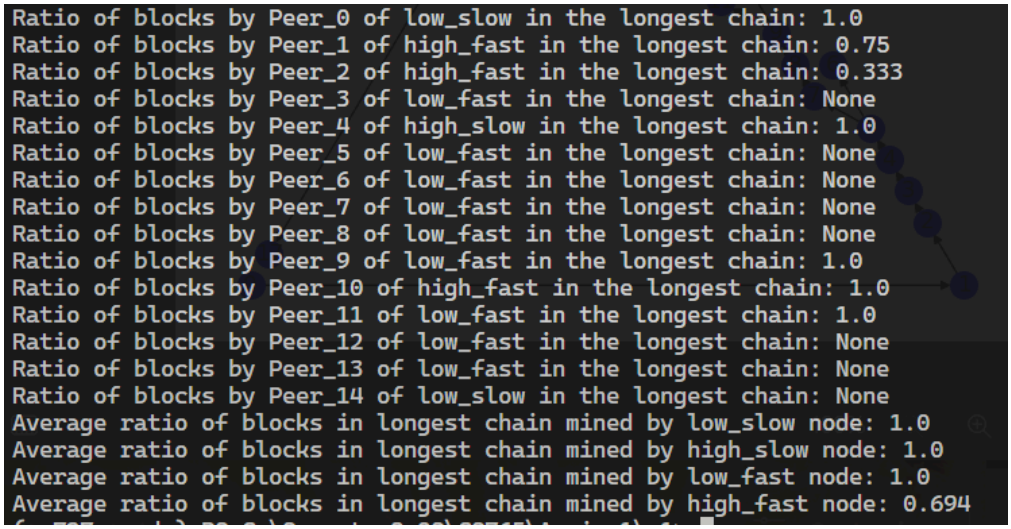
```
Ratio of blocks by Peer_0 of low_slow in the longest chain: 1.0
Ratio of blocks by Peer_1 of low_slow in the longest chain: 1.0
Ratio of blocks by Peer_2 of low_slow in the longest chain: None
Ratio of blocks by Peer_3 of low_slow in the longest chain: 0.0
Ratio of blocks by Peer_4 of low_slow in the longest chain: None
Ratio of blocks by Peer_5 of low_slow in the longest chain: None
Ratio of blocks by Peer_6 of low_slow in the longest chain: None
Ratio of blocks by Peer_7 of low_slow in the longest chain: 1.0
Ratio of blocks by Peer_8 of low_slow in the longest chain: 1.0
Ratio of blocks by Peer_9 of low_slow in the longest chain: 0.8
Ratio of blocks by Peer_10 of low_slow in the longest chain: 0.5
Ratio of blocks by Peer_11 of low_slow in the longest chain: 0.0
Ratio of blocks by Peer_12 of low_slow in the longest chain: None
Ratio of blocks by Peer_13 of low_slow in the longest chain: 0.667
Ratio of blocks by Peer_14 of low_slow in the longest chain: 1.0
Average ratio of blocks in longest chain mined by low_slow node: 0.697
Average ratio of blocks in longest chain mined by high_slow node: None
Average ratio of blocks in longest chain mined by low_fast node: None
Average ratio of blocks in longest chain mined by high_fast node: None
```

Example 5:

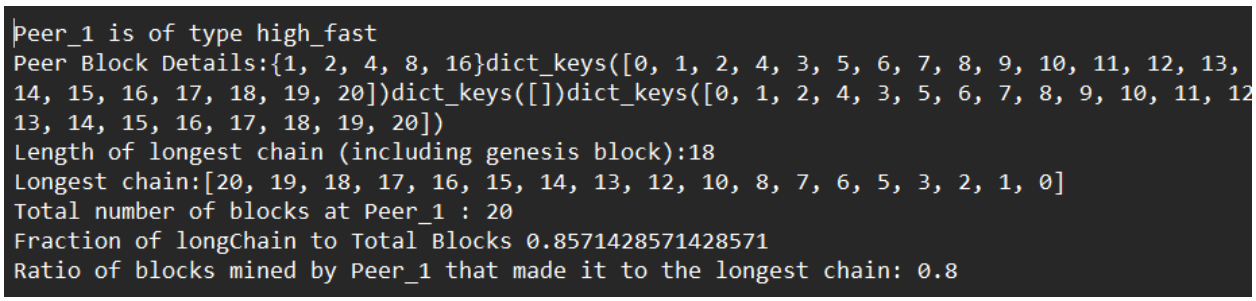
n=15, z0=25.0, z1=75.0, ttx=100.0, I=1000.0



Blockchain at Peer 1



As we have more fast Nodes than slower Nodes and less high_CPU nodes than low_CPU, the blocks generated by high_fast are generating



Conclusion Remarks :

- Increasing the value of peers increases the number of branches in the blockchain as more blocks are possible to generate at every instant of time.
- Increasing percentage of slow speed peers increase the number of forks as the now we are dependent on the link speed which will be slow throught the network, and there is more possibility to generate blocks in between Block Receive time
- Increasing the percentage of slow CPU peers, here depending upon the Latency we get two kinds of behavior : at lower latencies, we get less branches as every block which is generated gets moved to head of Blockchain of every Peer. Similarly if we have higher latency and higher high_CPU peers we get more branching. For the cases of high slow CPU and higher Latency, and high high_CPU and lower_latency we can experience anykind of behavior.
- I : Increasing inter arrival time reduces the amount of branching as blocks can utilize time between block mining to send to its Peers in the network