Discrete Event Simulator for P2P Cryptocurrency Network

Shalabh Gupta 180050095
Vrinda Jindal 180050120
Pratyush Agarwal 180050078

Task 2: Reasons for choosing Exponential distribution

For individual peers, time can be broken down into discrete intervals, and the event that a transaction will be received in a particular time interval -> Bernoulli trials. When the time intervals tend to 0, and the number of such steps tend to infinity, the interarrival time (x, where x = t*n, t is time interval, t-> 0 and n is number of intervals, n -> infinity) can be modelled as belonging to the exponential distribution (as derived in class for interarrival time for blocks) Thus in real blockchain systems, interarrival times are from exponential distribution, which is why we simulated interarrival transaction times to come from those.

$$P(I = nt) = (1 - \beta t)^{n-1}\beta t$$
$$P(I > nt) = (1 - \beta t)^{n}$$
$$Setting\ x = nt,\ P(I > x) = (1 - \beta x/n)^{n} \Rightarrow^{n->\infty} e^{-\beta x}$$

Thus the random variable I can be modelled as coming from exponential distribution.
Also, using the exponential distribution provides the memoryless property that the interarrival times follow.

Task4: Random Connected Graph Sampling

The node network forms a graph which follows a power law distribution with coefficient 3. All real networks generally follow the power law degree distribution. Not only this, almost all some-what real bigger networks are scale free.

We have used the Barabahsi-Albert model to sample our graphs (scale-free). We have used the networkx python library to implement the graph sampling.

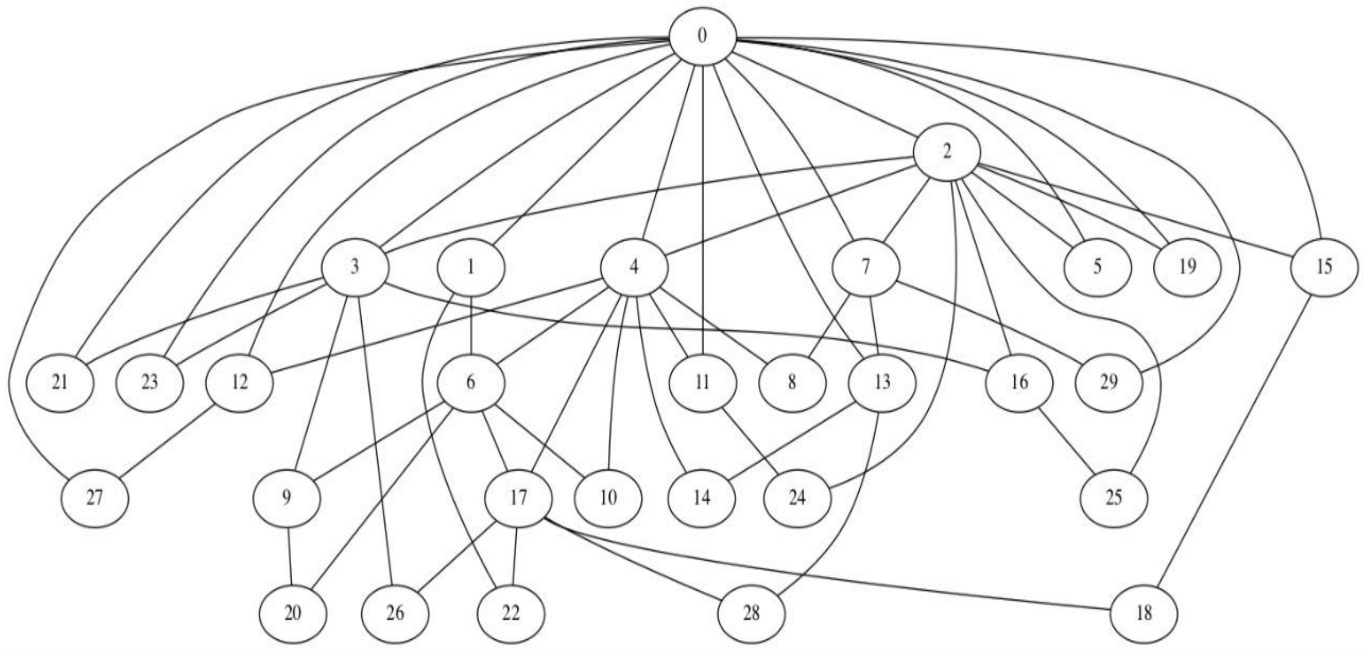
The reference mentioned below (Baumann et. al.) also concludes that the real bitcoin network converges to a scale-free graph.

In such graphs, the fraction P(k) of nodes in the network having k connections to other nodes goes for large values of k as

$$P(k) \sim k^{-3}$$

Also, when a new node is added to the network it first attaches itself to ‘m’ existing nodes.
We have this parameter as 2.

Sample Graph Generated by this method (30 nodes):



Reference:
https://www.researchgate.net/publication/262562539_Exploring_the_Bitcoin_Network

Task5: Relation between mean of d_{ij} and c_{ij}

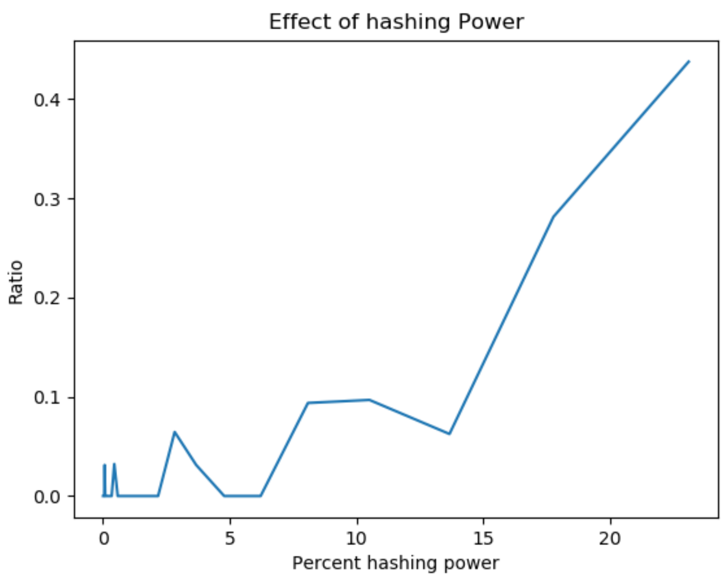
c_{ij} is the link speed between nodes i and j while d_{ij} is the queuing delay (delay to put the packet on the link) incurred at node i, to send the message to node j. If links are able to carry the messages faster (link speed c_{ij} is higher), then waiting queues will naturally be smaller, and message will not have to wait much to be scheduled on the link, thus reducing d_{ij}. On the other hand, if link speed c_{ij} is less, queues at node i will be longer, and waiting times will be higher, increasing d_{ij}. Thus, mean of d_{ij} is inversely proportional to c_{ij}.

Task7: Suitable mean for Block mining time T_k

With the interarrival time for transactions in mind, we tried to vary the mining time T_k, and it was observed that keeping T_k roughly same as T_tx helps balance the system. If T_k is kept large, less number of blocks will be generated and the contention between transactions to enter the blocks will increase and more transactions will go ignored for a long time. Also, if T_k is kept small, a lot of blocks will be generated and at a much faster rate, taking up most of the events in the queue.

Analysis

Comparison of ratio with % hashing power



Parameters

- Number of nodes (n) = 30
- T_tx = 2
- T_k = 0.5

- Ratio : Number of blocks generated by a node included in longest chain / total blocks generated by node
- Fraction of slow nodes : 0
- DIstribution of Hashing power : c*(1.3)^i for ith node

Inference : As expected, the more hashing power a node has, more are the blocks it can mine and hence has a higher ratio of blocks entering the longest chain.

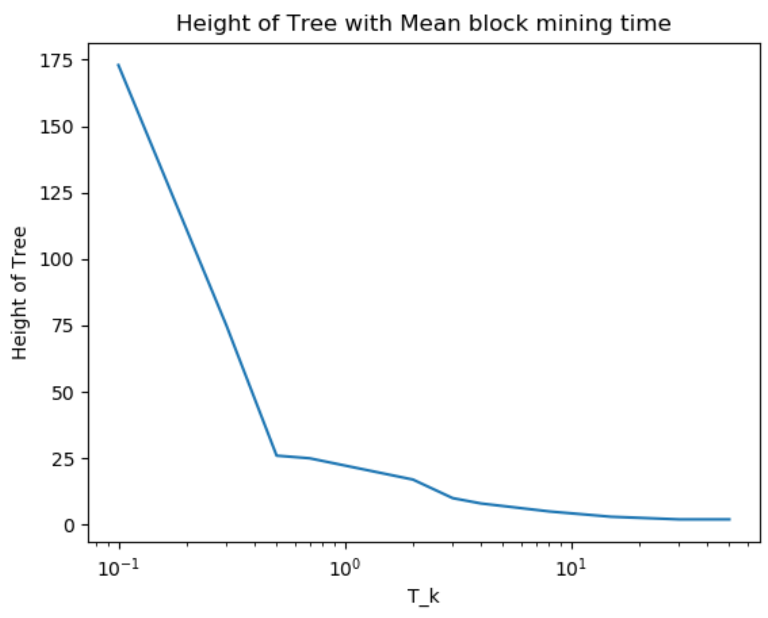
Slow vs Fast nodes

Parameters

- Number of nodes (n) = 30
- T_tx = 2
- T_k = 0.5
- Ratio : Number of blocks generated by a node included in longest chain / total blocks generated by node
- Fraction of slow nodes = 0.5
- Average ratio for slow nodes = 0.0202
- Average ratio for fast nodes = 0.0467
- Distribution of Hashing power : Random

Inference : Slow nodes have low ratio, ie lower fraction of their mined blocks are included in the longest chain due to the high latency incurred to their blocks in reaching other nodes.

Height of Tree with mean block mining time (How long are the branches)

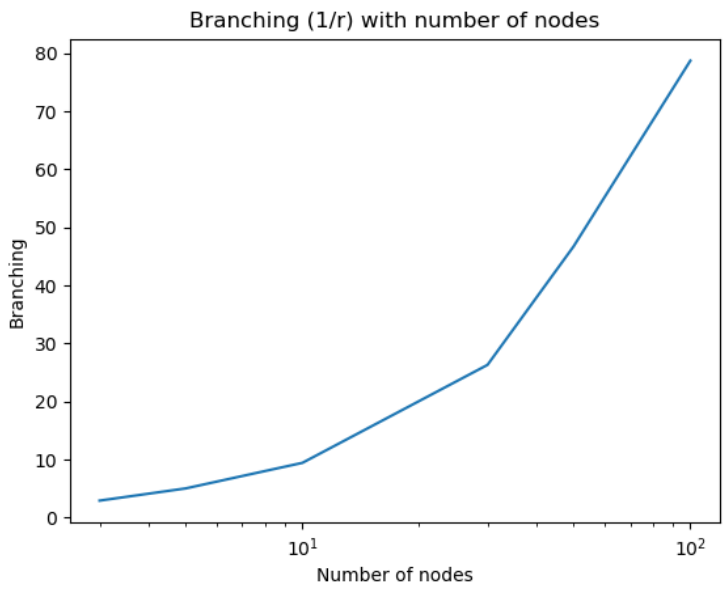


Height of the tree here refers to the number of blocks in the longest chain of the tree.
Parameters

- Number of nodes (n) = 30
- T_tx = 2
- Fraction of slow nodes : 0
- Distribution of Hashing power : Random

Inference : As mean block mining time increases, hashing power reduces, hence less blocks are mined and smaller trees are obtained.

Branching vs number of nodes



The Hashing Power of all nodes is kept the same
Branching is interpreted as 1/r where r is the ratio of blocks generated by a node included in the longest chain to the total number of blocks the node generates. This makes sense since less the value of r is, it means that more number of blocks of a node got wasted into creating forks (branches)

Parameters

- T_tx = 2
- T_k = 0.5
- Fraction of slow nodes : 0

Inference : As the number of nodes increases, competition between the nodes increases, and the value of r for nodes decreases since all the nodes compete for mining a block for the largest chain. Hence chances of forks being formed increase, thus leading to an increase in branching

Effect of Hashing Power distribution among Nodes

Each node has a relative hashing power. We study three different ways of assigning percentages of hashing powers to the nodes :

- Equal percent of hashing power for each node
- Percentage of Hashing power increases exponentially as node index increases ie node i has hashing power c*(1.3)^i, where c is normalising factor
- Random percentage of hashing power to each node

Parameters

- T_tx = 2
- T_k = 0.5
- Fraction of slow nodes : 0

Average value of 'r' : r is the ratio of blocks generated by a node included in the longest chain to the total number of blocks the node generates. We average this value over all the nodes.

n	Hashing Power	Height of Tree	Average Value of 'r'
10	Same for all (1/n)	40	0.105
	c* (1.3)^i	47	0.0998
	Random	55	0.102
30	Same for all (1/n)	44	0.034
	c* (1.3)^i	33	0.0399
	Random	56	0.0339
50	Same for all (1/n)	27	0.0221

	c* (1.3)^i	38	0.02
	Random	52	0.02

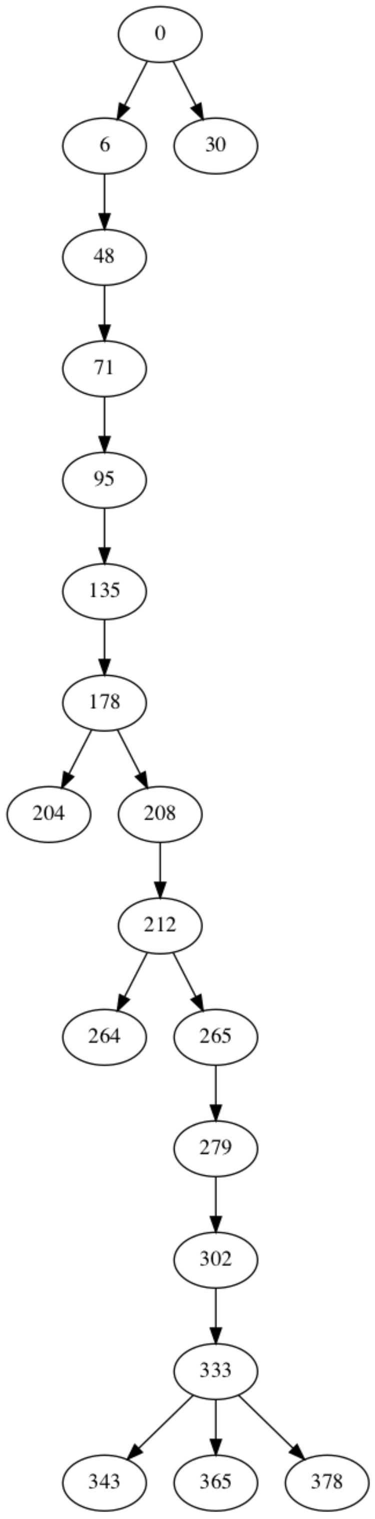
Observations :

- The average value of r remains the same even if we change the distribution of hashing power. We observe that in the GP distribution of hashing power, there are only a few nodes(9 out of 30) that have r>0 (the nodes with greater hashing power), whereas when all the nodes have equal hashing power, the r values across nodes don't vary that much, ie each node contributes almost equally
- As seen previously, the average value of r decreases as the number of nodes increases in the network, i.e. less number of blocks generated by a node get included in the longest chain, leading to more branching.

Task 8: Visualization

We use .dot files to visualise graphs. The .dot files are generated by the C++ code.

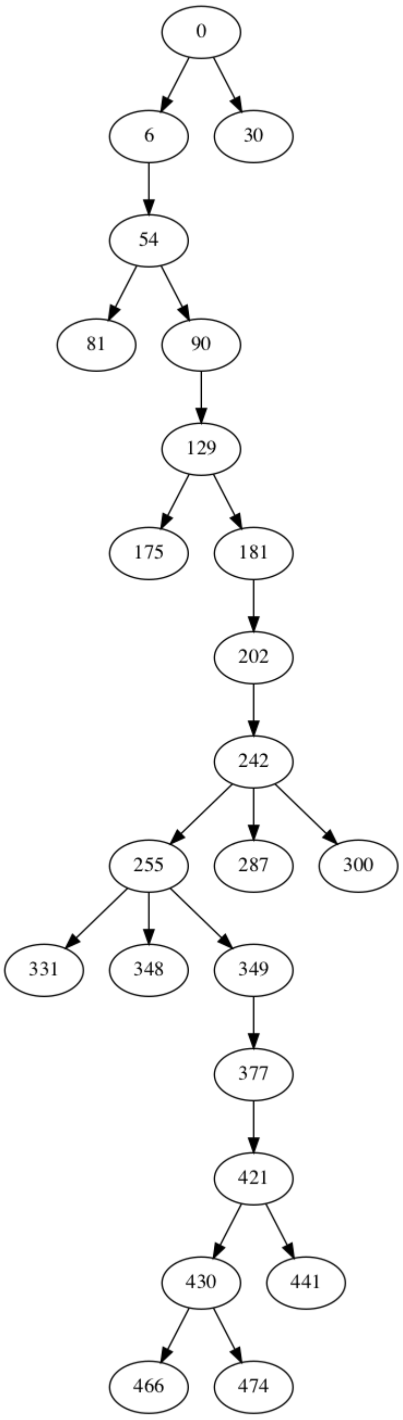
Tree Visualisation



To keep the diagram clear, we use low hashing power so that tree is not much big

Parameters

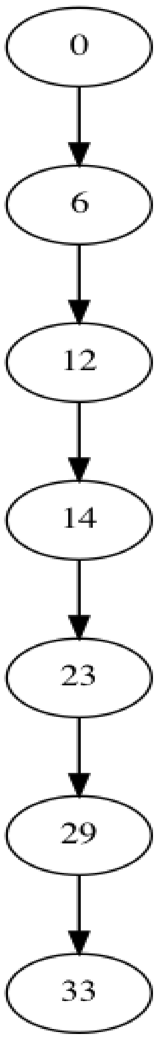
- Number of nodes (n) = 30
- T_tx = 2
- T_k = 2
- Fraction of slow nodes = 0
- Distribution of Hashing power : Equal to all



A more branched tree is obtained when the underlying graph (network) is dense and the number of nodes in the P2P network are high.

Parameters:

- Nodes = 40
- Simulation time = 20 seconds
- T_tx = 4
- T_k = 1
- Fraction of slow nodes : 0.5
- Distribution of Hashing power: Random



A linear and smaller tree is obtained when there are less number of nodes in the network, and when interarrival time of blocks is set higher.

Parameters:

- Nodes : 6
- Simulation time = 20 seconds
- T_tx = 5
- T_k = 5
- Fraction of slow nodes : 0
- Distribution of Hashing power: Random

Final Submission Details

- Source code : big.cpp, gen_graph.py
- Details to run the code: README.txt
- Design doc: Modular Code Flow in DesignDoc.jpg, detailed code-flow of Block lifecycle in DesignDoc2.jpg
- Report.pdf