CS765: Introduction of Blockchains, Cryptocurrencies & Smart Contracts

Project Title: Simulation of a Peer-to-Peer Cryptocurrency Network

Instructor: Prof. Vinay Ribeiro

Group Members:

1. Darshit Gandhi (22M0824)
2. Niteen Pawar (22M0800)
3. Abhijeet Singh (22M0749)

Overview:

This project consists of building a **Discrete Event Simulator** for a P2P cryptocurrency network, mimicking blockchain mining activities like **transaction generation, transaction forwarding, PoW mining, and blockchain maintenance**. We also simulated the **selfish mining and stubborn mining attacks** on top of discrete-event simulator. Then we quantitatively assessed and visualized the resultant blockchain tree for different parameters.

Features:

1. This discrete-event simulator maintains an "event-queue" from which the earliest event is executed. This event may create further future events which get added to the queue. For example, an event in which one node "sends a block" to connected peers will create future events of "receive block" at its peers.
2. There are n peers, each with a unique ID, where n is set at the time of initiation of the network. Some of these nodes (say z 0 percent, where z 0 is a command line simulation parameter) are labeled "slow" and the others "fast". In addition, some of these nodes (say z 1 percent, where z 1 is a command line simulation parameter) are labeled "low CPU" and the others "high CPU".
3. Each transaction has the format: "TxnID: ID x pays ID y C coins".
4. This simulator contains functions to **create a new transaction, forward the transaction, PoW mining, receive a block, send a new block, and blockchain maintenance**. A node forwards any transaction heard from one peer to another connected peer, provided it has not already sent the same transaction to that peer, or provided it did not hear (receive) the transaction from that peer.
5. **Proof of Work (PoW)** is simulated as follows: All nodes have the genesis block at the start of the simulation. Each block must have a unique ID, say BlkID (You can choose any method to ensure this). Any peer, say peer k, maintains a tree of blocks as in bitcoin. When it receives a block from another peer, it validates all its transactions (no balance of any peer should go negative), and if the block is valid it adds the block to its tree. When a received block creates a new longest chain at peer k (longer than the previous longest chain), say at time tk , then we simulate PoW mining of a new block as follows: Peer k forms a block at tk by selecting a subset of the transactions received so far and not included in any blocks in the longest chain. After forming a block, node k generates a random variable Tk , at time tk , as follows. Suppose the interarrival time between blocks on average is I (for example, I is 600 sec in Bitcoin), and node

k has fraction h k (where 0 < h k < 1) of the total hashing power. Then T k is drawn from an exponential distribution with mean equal to I/hk . If peer k is a high CPU P node, then it is assumed to have 10 times higher hashing power than a low CPU node.

6. Each node **maintains a tree of all blockchains** heard since the start of the simulation. The node stores the time of arrival of every block in its tree. This information is written to a file at the end of the simulation.
7. Experimented by choosing different values for different parameters (like n, z, T tx , Tdij , mean of Tk , etc.) and **visualized corresponding blockchain trees**.
8. **Selfish mining and Stubborn mining attacks** are simulated on top of this P2P discrete simulator. Tried different values of the parameters and visualized how these attacks can affect the respective blockchains.

Compilation and Source Code Structure:

1. **Location: /Code/P2P Cryptocurrency Network Code/**

- **Compiling and running the code:**

    1. python3 main.py <Number of nodes> <Ttx (TXN Interarrival mean time)> <percent of slow nodes> <percent of low CPU nodes>
    2. The program will run for 30 sec. After the code runs successfully, the blockchain tree for each node will be displayed and will be saved in a png file. Also, the longest chain for each node will be displayed and will be saved in a png file.
    3. Also, a txt file will be created for each node which contains the block ids and its corresponding arrival time of each block present in that node's blockchain tree.

- **Source Code Structure:**

1. main.py: This file will take the required command line inputs. Then it will create a randomly connected graph between 4 and 8 other peers from n nodes by taking help from generate_graph.py. Then, it will initiate the initial TXN and initial block generation event. It will then pop events one by one from the events queue, and based on the event type, it will do the processing. There will be 4 types of events: TXN_GEN, TXN_REC, BLK_GEN, and BLK_REC. Then, after all the processing, it will display the blockchain tree and longest chain for each node.

    2. node.py: This file contains various properties and data structures required by the node. This also includes functions for **TXN Generation, TXN Receive, BLK Generation and BLK Receive.**

    3. generate_graph.py: It will create a randomly connected graph to between 4 and 8 other peers from n nodes

    4. transaction_object.py: This file contains the parameters required for a transaction object.

    5. block.py: This file contains the parameters required for a block object.

    6. event.py: This file contains the parameters required for an Event object.

    7. global_data.py: This file contains the global parameters that can be required by many other files.

2. Location: /Code/ Selfish and Stubborn mining/

- **Compiling and running the code:**
    1. python3 main.py n Ttx z0 z1 hp zeta

       n = Number of nodes, Ttx = TXN Interarrival mean time, z0 = percent of slow nodes, z1 = percent of low CPU nodes, hp = Hashing Power of the attacker, zeta = The fraction of honest nodes, an adversary is connected to
    2. The program will run until a particular amount of blocks are created. After the code runs successfully, the blockchain tree for each node will be saved in a png file. Also, the longest chain for each node will be saved in a png file.
    3. Also, a txt file will be created for each node which contains the block ids and its corresponding arrival time of each block present in that node's blockchain tree.

- **Source Code Structure:**

  1. main.py: This file will take the required command line inputs. Then it will create a randomly connected graph between 4 and 8 other peers from n nodes by taking help from generate_graph.py. Then, it will initiate the initial TXN and initial block generation event. It will then pop events one by one from the events queue, and based on the event type, it will do the processing. There will be 4 types of events: TXN_GEN, TXN_REC, BLK_GEN, and BLK_REC. Now, the selfish mining attack and the stubborn mining attack will be executed based on the hashing power and zeta. Then, after all the processing, it will display the blockchain tree and longest chain for each node.

  2. node.py: This file contains various properties and data structures required by the node. This also includes functions for **TXN Generation, TXN Receive, BLK Generation and BLK Receive**. Also, Node.py contains the code to perform the **Selfish mining attack and the Stubborn Mining attack.**
  3. generate_graph.py: It will create a randomly connected graph to between 4 and 8 other peers from n nodes
  4. transaction_object.py: This file contains the parameters required for a transaction object.
  5. block.py: This file contains the parameters required for a block object.
  6. event.py: This file contains the parameters required for an Event object.
  7. global_data.py: This file contains the global parameters that can be required by many other files.

Questions:

1. What are the theoretical reasons for choosing the exponential distribution?
   - The exponential distribution is "memoryless", which means that the probability of discovering a block at any given moment is unaffected by the amount of time since the discovery of the previous block. Thus, it makes sense to simulate the process of block discovery.
   - The average time between blocks is the only parameter needed to explain the exponential distribution, which has a straightforward shape. This makes it practical to use and offers a straightforward approach to mimic the block discovery procedure.
2. Why is the mean of dij inversely related to cij ? Give justification for this choice.
   - The message queueing delay at node I before forwarding the message to node j is represented by the variables dij and cij, respectively Therefore, the speed of cij will determine how quickly a node can process its queue, giving rise to an inverse relationship between cij and dij.
3. Justification for the chosen mean value for Tk from an exponential distribution.
   - Tk is the average time spent waiting for blocks to generate. Consequently, a larger value of Tk would result in fewer blocks being created, and vice versa. The number of transactions per block de- creases with an excess of blocks, whereas the node transaction queue fills up with an excess of blocks. Therefore, these two parameters must be balanced.

Observations of the Part 1 of the project:

Let,

$$r = \frac{\textit{Number of blocks generated in the longest chain}}{\textit{Number of blocks generated at the end of simulation}}$$

N=No. of nodes

Ttx =Mean transaction inter-arrival time Z0= number of slow nodes

Z1= number of low cpu nodes

CASE 1 :

N= 10

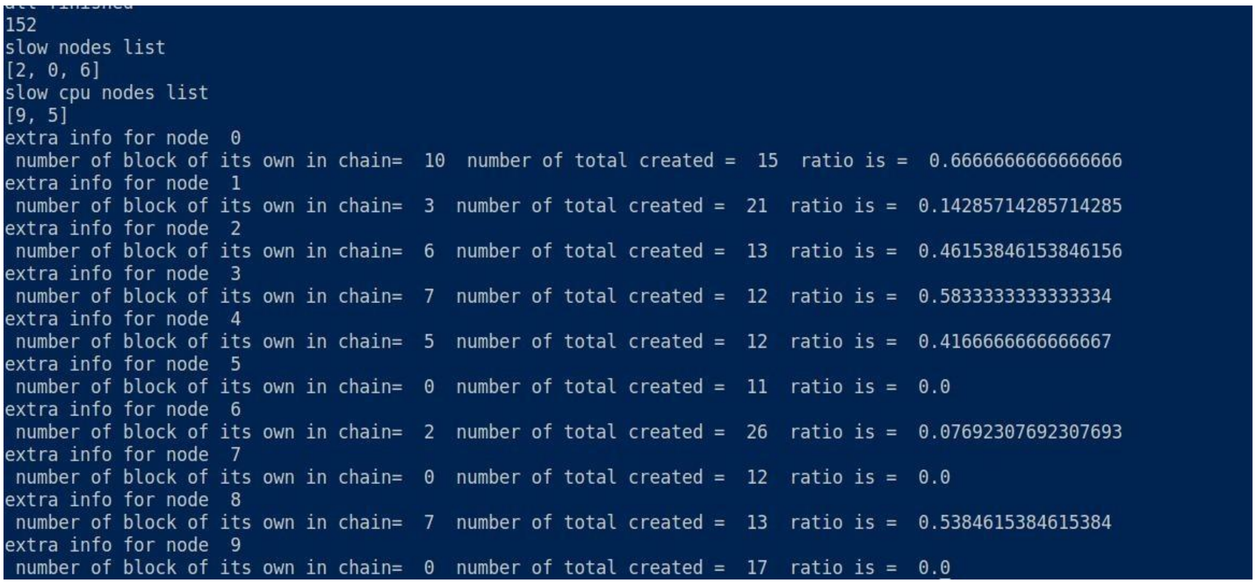
Ttx =60

Z0= 30

Z1= 20

r on slow node= 0.67 r on fast node= 0.58 r on low cpu= 0.00

r on high cpu=0.67



```
152
slow nodes list
[2, 0, 6]
slow cpu nodes list
[9, 5]
extra info for node  0
 number of block of its own in chain=  10  number of total created =  15  ratio is =  0.6666666666666666
extra info for node  1
 number of block of its own in chain=  3  number of total created =  21  ratio is =  0.14285714285714285
extra info for node  2
 number of block of its own in chain=  6  number of total created =  13  ratio is =  0.46153846153846156
extra info for node  3
 number of block of its own in chain=  7  number of total created =  12  ratio is =  0.5833333333333334
extra info for node  4
 number of block of its own in chain=  5  number of total created =  12  ratio is =  0.4166666666666667
extra info for node  5
 number of block of its own in chain=  0  number of total created =  11  ratio is =  0.0
extra info for node  6
 number of block of its own in chain=  2  number of total created =  26  ratio is =  0.07692307692307693
extra info for node  7
 number of block of its own in chain=  0  number of total created =  12  ratio is =  0.0
extra info for node  8
 number of block of its own in chain=  7  number of total created =  13  ratio is =  0.5384615384615384
extra info for node  9
 number of block of its own in chain=  0  number of total created =  17  ratio is =  0.0
```

CASE 2 :

N= 10

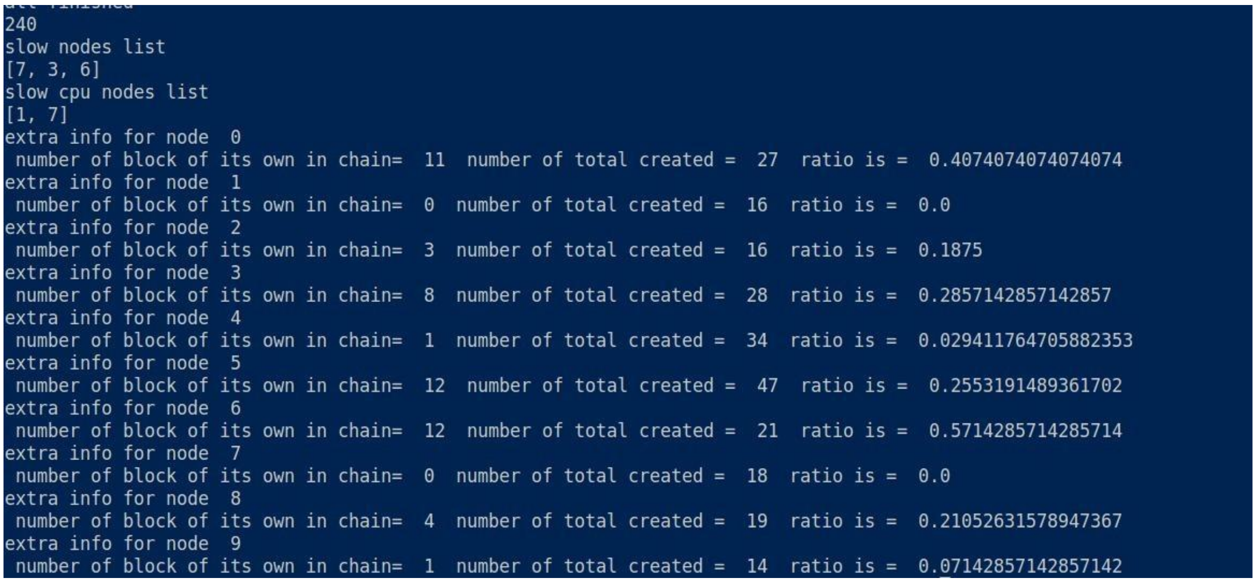
Ttx = 120

Z0= 30

Z1= 20

r on slow node= 0.57 r on fast node= 0.40 r on low cpu=0.0

r on high cpu=0.57



```
240
slow nodes list
[7, 3, 6]
slow cpu nodes list
[1, 7]
extra info for node  0
 number of block of its own in chain=  11  number of total created =  27  ratio is =  0.4074074074074074
extra info for node  1
 number of block of its own in chain=  0  number of total created =  16  ratio is =  0.0
extra info for node  2
 number of block of its own in chain=  3  number of total created =  16  ratio is =  0.1875
extra info for node  3
 number of block of its own in chain=  8  number of total created =  28  ratio is =  0.2857142857142857
extra info for node  4
 number of block of its own in chain=  1  number of total created =  34  ratio is =  0.029411764705882353
extra info for node  5
 number of block of its own in chain=  12  number of total created =  47  ratio is =  0.2553191489361702
extra info for node  6
 number of block of its own in chain=  12  number of total created =  21  ratio is =  0.5714285714285714
extra info for node  7
 number of block of its own in chain=  0  number of total created =  18  ratio is =  0.0
extra info for node  8
 number of block of its own in chain=  4  number of total created =  19  ratio is =  0.21052631578947367
extra info for node  9
 number of block of its own in chain=  1  number of total created =  14  ratio is =  0.07142857142857142
```

CASE 3 :

N= 10

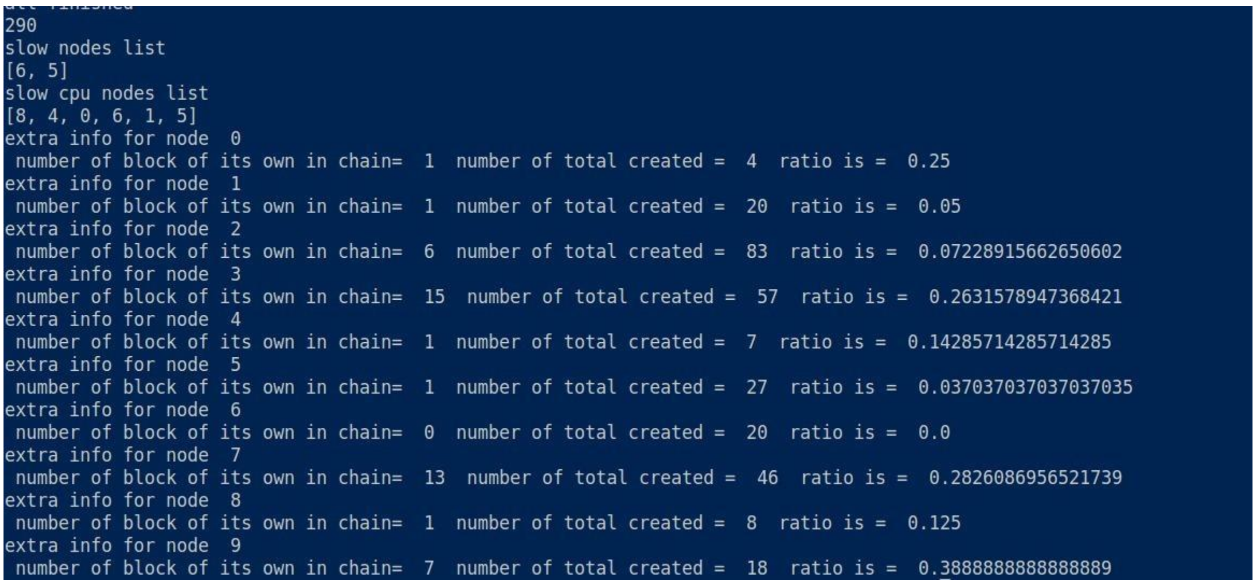
Ttx =250

Z0= 20

Z1= 60

r on slow node=0.03 r on fast node=0.39 r on low cpu= 0.25

r on high cpu=0.38



```
290
slow nodes list
[6, 5]
slow cpu nodes list
[8, 4, 0, 6, 1, 5]
extra info for node  0
 number of block of its own in chain=  1  number of total created =  4  ratio is =  0.25
extra info for node  1
 number of block of its own in chain=  1  number of total created =  20  ratio is =  0.05
extra info for node  2
 number of block of its own in chain=  6  number of total created =  83  ratio is =  0.07228915662650602
extra info for node  3
 number of block of its own in chain=  15  number of total created =  57  ratio is =  0.2631578947368421
extra info for node  4
 number of block of its own in chain=  1  number of total created =  7  ratio is =  0.14285714285714285
extra info for node  5
 number of block of its own in chain=  1  number of total created =  27  ratio is =  0.037037037037037035
extra info for node  6
 number of block of its own in chain=  0  number of total created =  20  ratio is =  0.0
extra info for node  7
 number of block of its own in chain=  13  number of total created =  46  ratio is =  0.2826086956521739
extra info for node  8
 number of block of its own in chain=  1  number of total created =  8  ratio is =  0.125
extra info for node  9
 number of block of its own in chain=  7  number of total created =  18  ratio is =  0.3888888888888889
```

CASE 4 :

N= 10

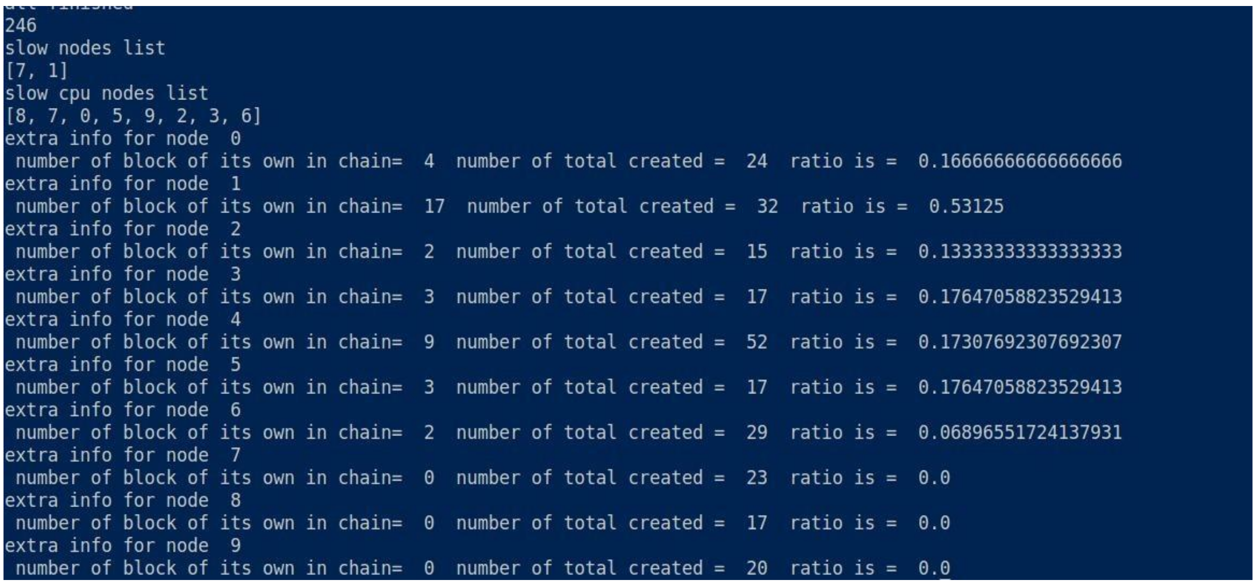
Ttx = 300

Z0= 20

Z1= 80

r on slow node=0.53 r on fast node=0.17 r on low cpu=0.17

r on high cpu=0.53



```
246
slow nodes list
[7, 1]
slow cpu nodes list
[8, 7, 0, 5, 9, 2, 3, 6]
extra info for node  0
 number of block of its own in chain=  4  number of total created =  24  ratio is =  0.16666666666666666
extra info for node  1
 number of block of its own in chain=  17  number of total created =  32  ratio is =  0.53125
extra info for node  2
 number of block of its own in chain=  2  number of total created =  15  ratio is =  0.13333333333333333
extra info for node  3
 number of block of its own in chain=  3  number of total created =  17  ratio is =  0.17647058823529413
extra info for node  4
 number of block of its own in chain=  9  number of total created =  52  ratio is =  0.17307692307692307
extra info for node  5
 number of block of its own in chain=  3  number of total created =  17  ratio is =  0.17647058823529413
extra info for node  6
 number of block of its own in chain=  2  number of total created =  29  ratio is =  0.06896551724137931
extra info for node  7
 number of block of its own in chain=  0  number of total created =  23  ratio is =  0.0
extra info for node  8
 number of block of its own in chain=  0  number of total created =  17  ratio is =  0.0
extra info for node  9
 number of block of its own in chain=  0  number of total created =  20  ratio is =  0.0
```

CASE 5 :

N= 10

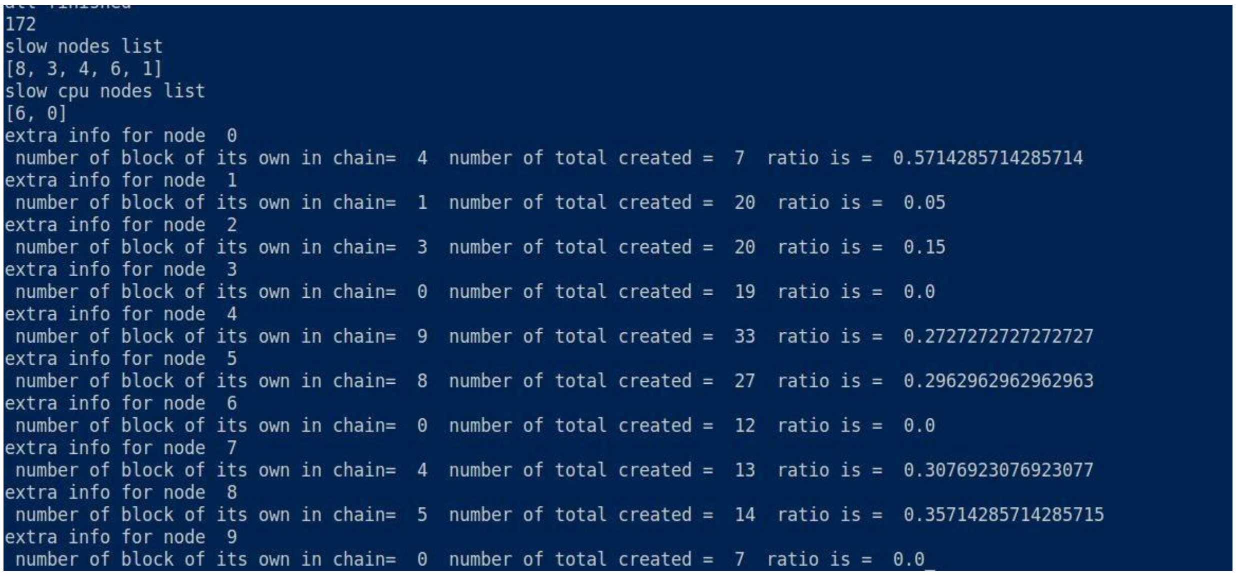
Ttx = 60

Z0= 50

Z1= 20

r on slow node=0.35 r on fast node=0.57 r on low cpu=0.57

r on high cpu=0.35



```
172
slow nodes list
[8, 3, 4, 6, 1]
slow cpu nodes list
[6, 0]
extra info for node  0
 number of block of its own in chain=  4  number of total created =  7  ratio is =  0.5714285714285714
extra info for node  1
 number of block of its own in chain=  1  number of total created =  20  ratio is =  0.05
extra info for node  2
 number of block of its own in chain=  3  number of total created =  20  ratio is =  0.15
extra info for node  3
 number of block of its own in chain=  0  number of total created =  19  ratio is =  0.0
extra info for node  4
 number of block of its own in chain=  9  number of total created =  33  ratio is =  0.2727272727272727
extra info for node  5
 number of block of its own in chain=  8  number of total created =  27  ratio is =  0.2962962962962963
extra info for node  6
 number of block of its own in chain=  0  number of total created =  12  ratio is =  0.0
extra info for node  7
 number of block of its own in chain=  4  number of total created =  13  ratio is =  0.3076923076923077
extra info for node  8
 number of block of its own in chain=  5  number of total created =  14  ratio is =  0.35714285714285715
extra info for node  9
 number of block of its own in chain=  0  number of total created =  7  ratio is =  0.0
```

Conclusion of Observations

- r of high CPU nodes is always greater than that of low CPU nodes in most cases because the low CPU nodes has lesser mining power so due to this it is able to mine lesser blocks as compared to high CPU node so its blocks are lesser in the longest chain.

- r of slow nodes is higher than r of faster nodes in most cases because the slow node will broadcast the blocks at a slower rate to its neighbors than the fast nodes.

- r value decreases with an increase in the value of Ttx.

Pictures of Blockchains

For,

number of nodes= 10

Ttx (TXN Interarrival mean time)= 60 percent of slow nodes = 30

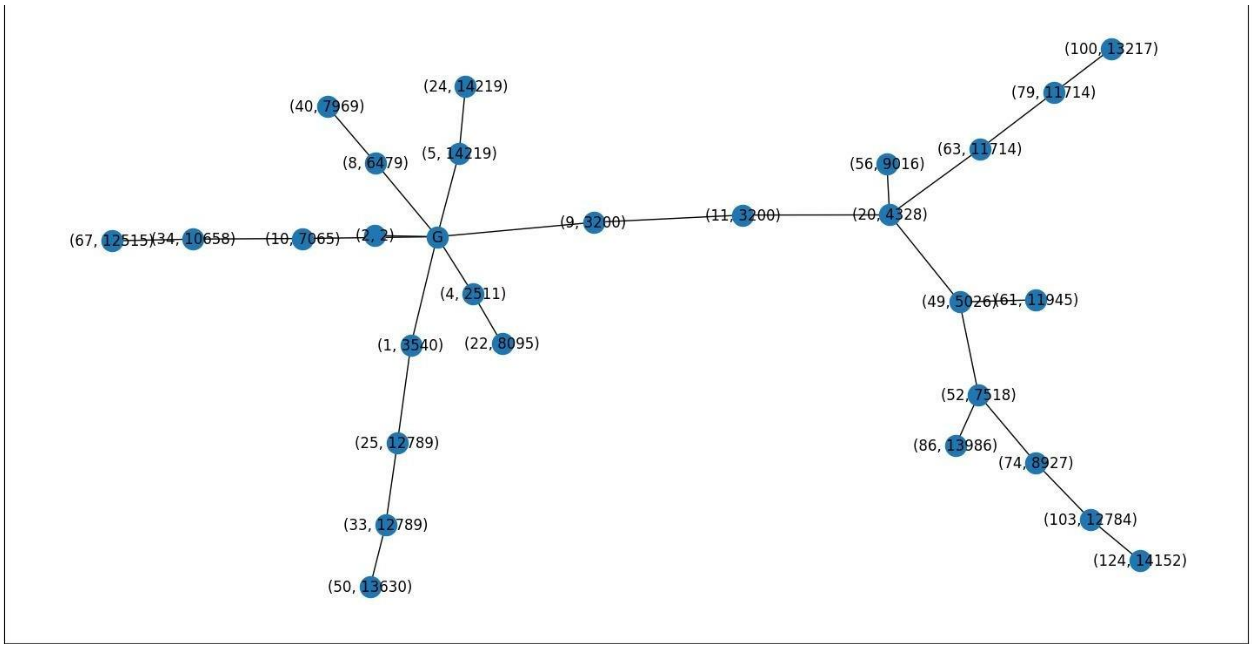
percent of low CPU nodes=20 slow nodes={2,6,8}

fast node={0,1,3,4,5,7,9} low cpu={5,6}

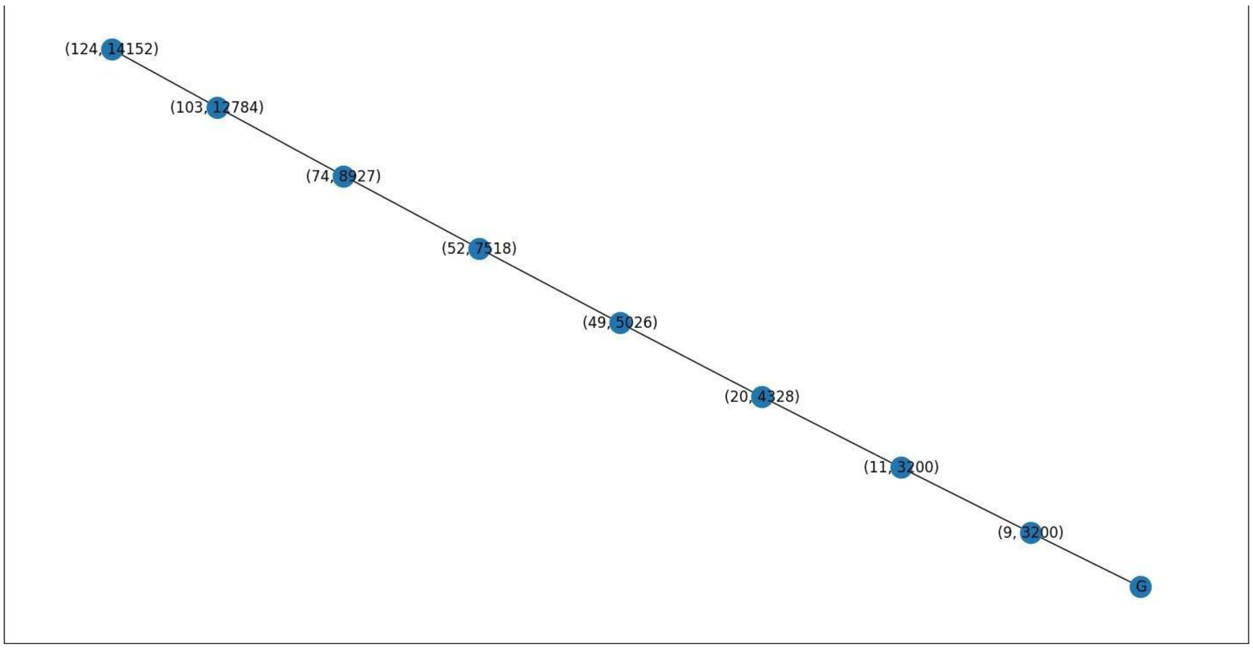
fast cpu={0,1,2,3,4,7,8,9}

Slow nodes + Fast CPU

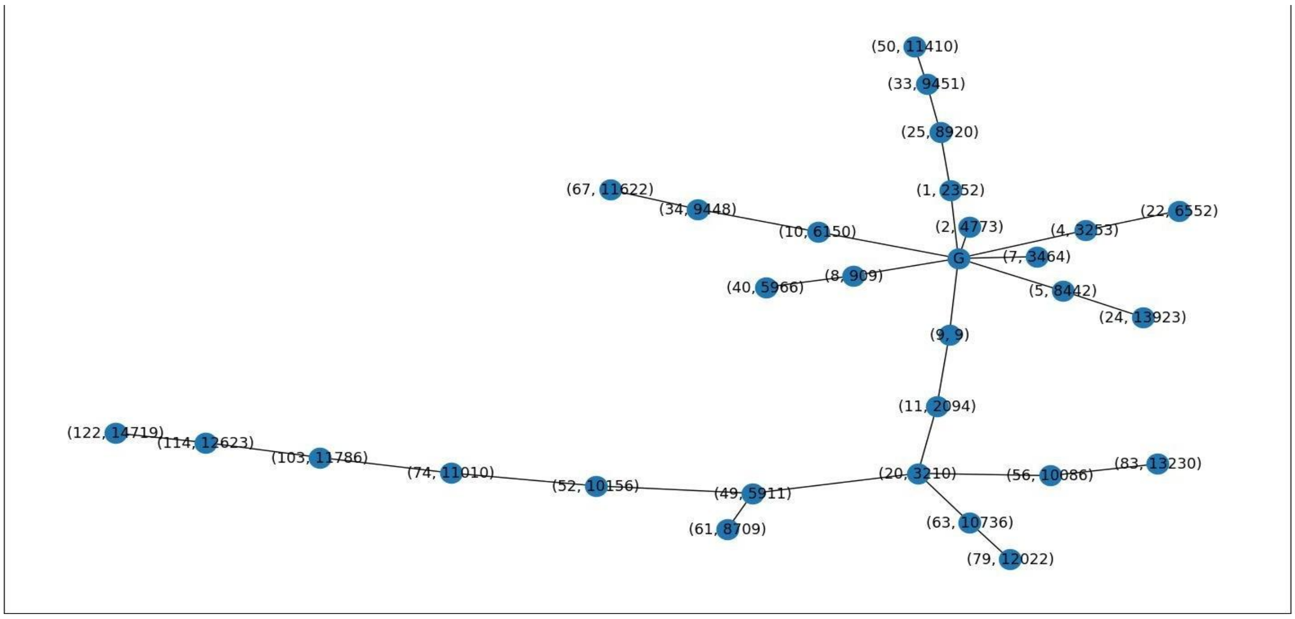
1.1 Tree of node 2



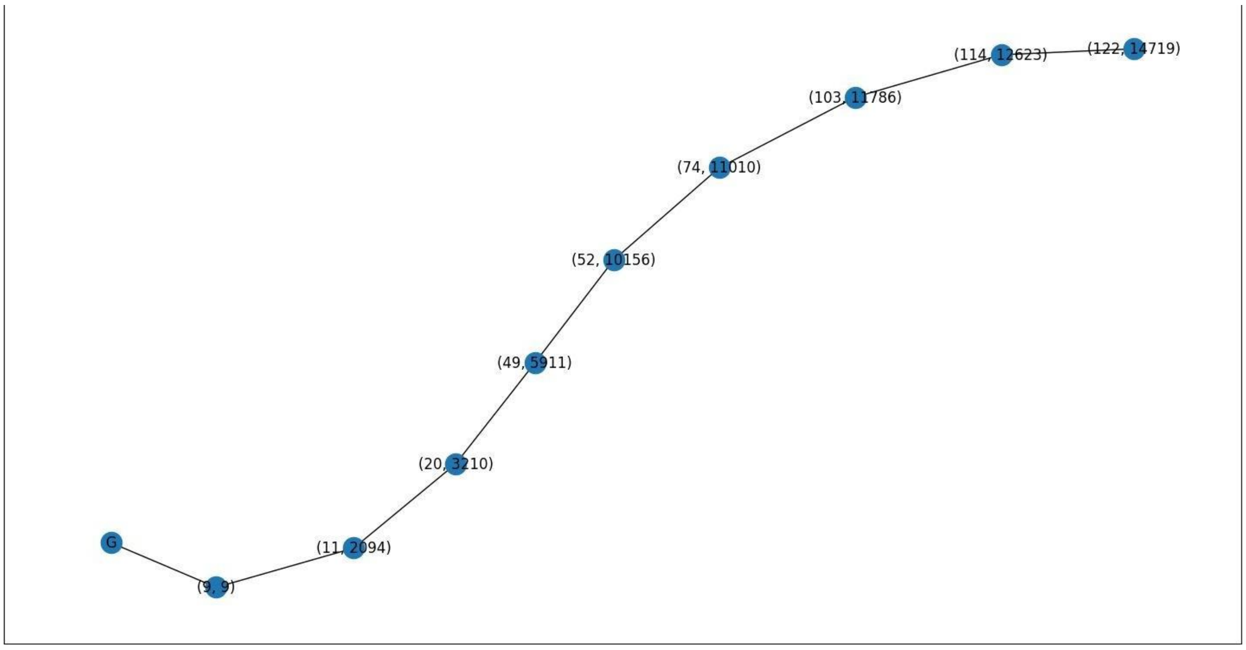
1.2 Longest chain of node 2



2.1 Tree of node 8

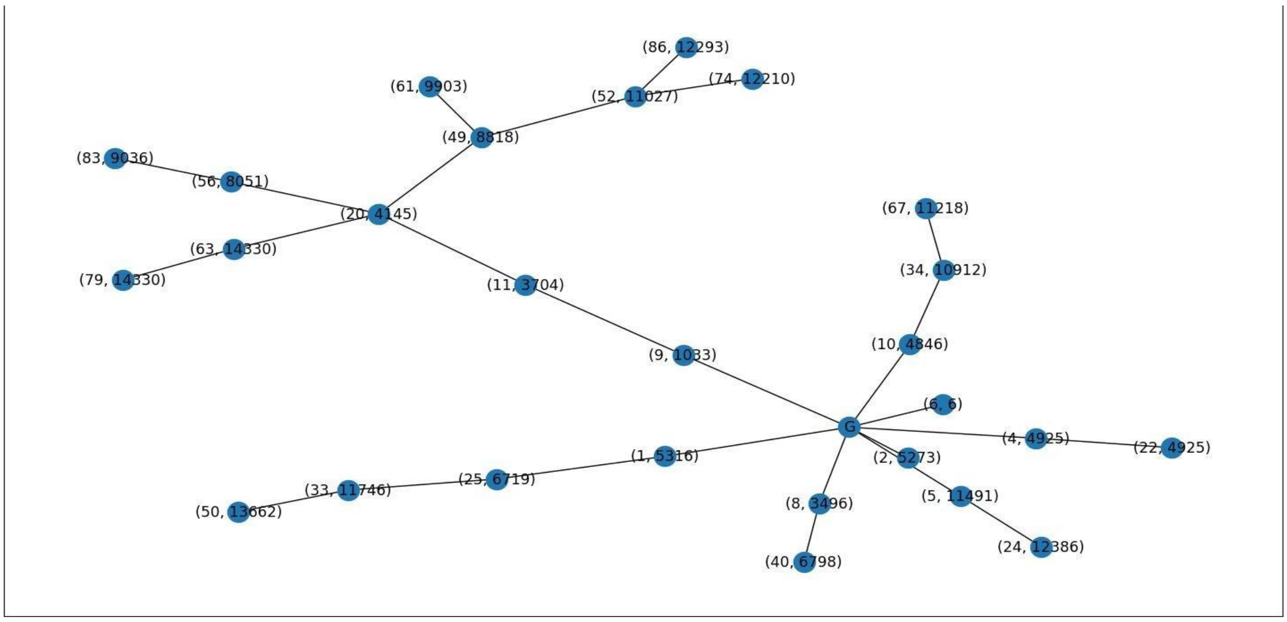


2.2 Longest chain of node 8

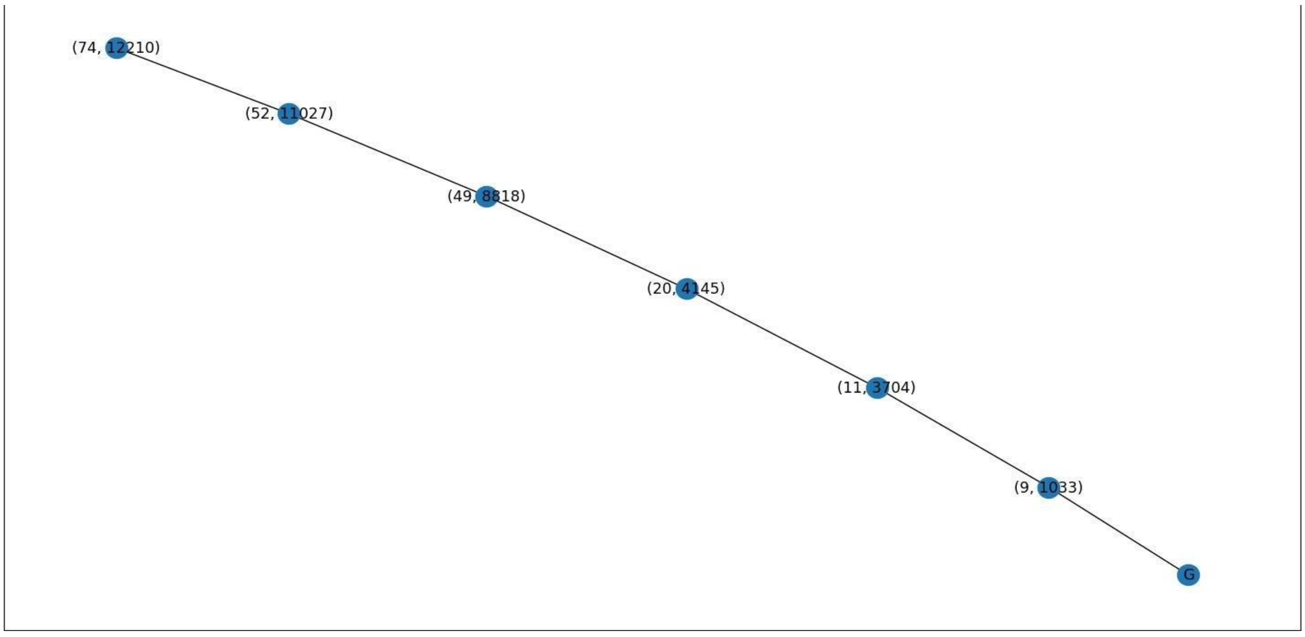


Fast Node + low CPU

3.1 Tree of node 5

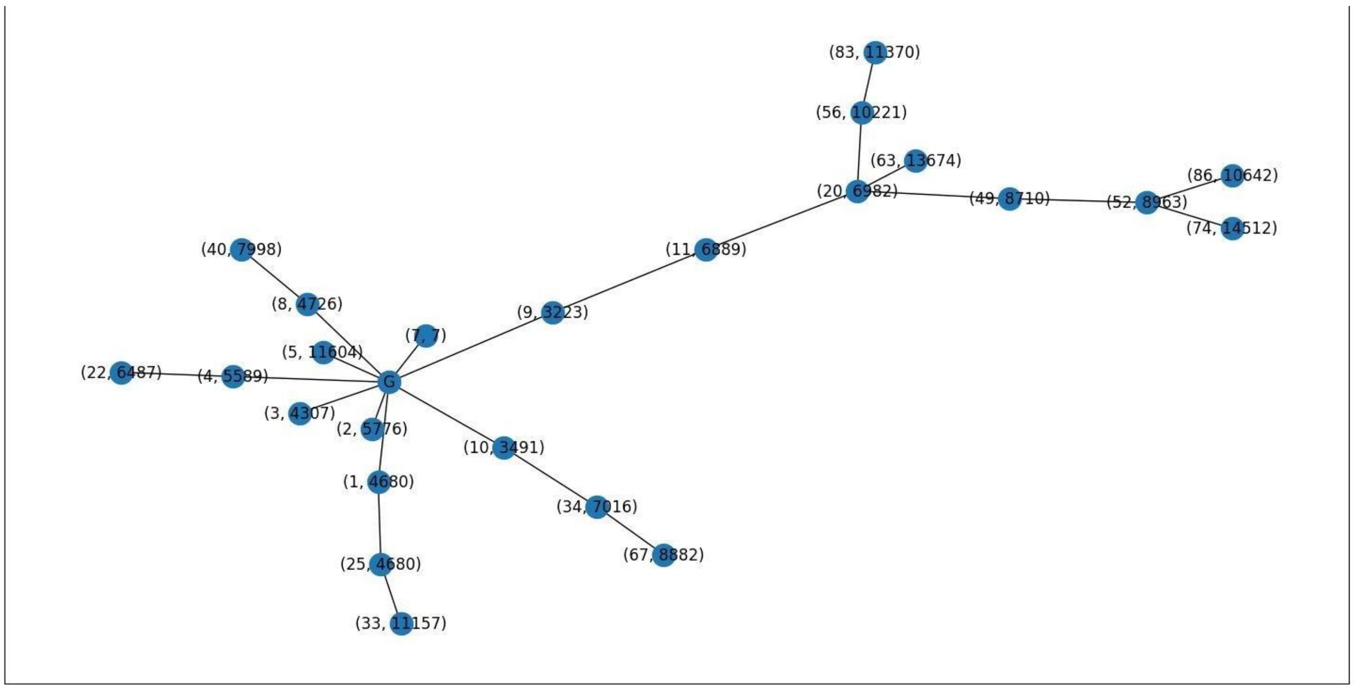


3.2 Longest chain of node 5

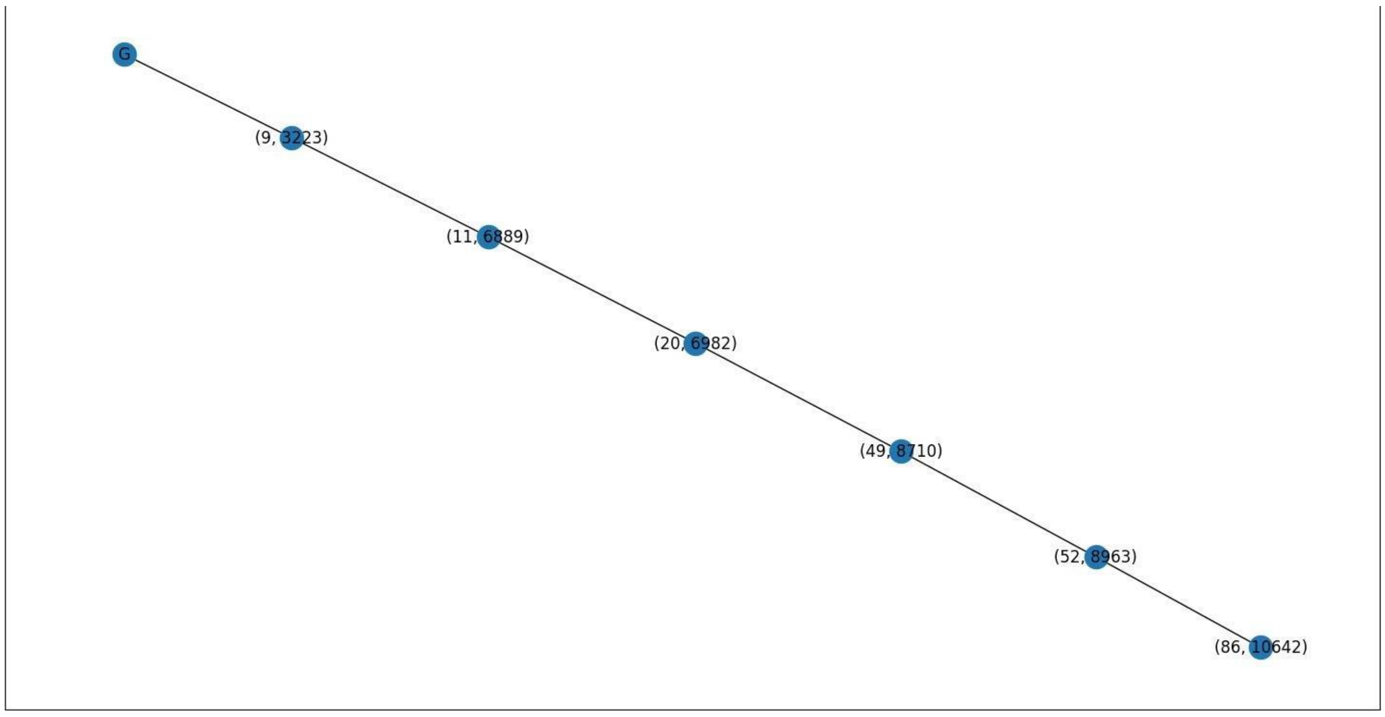


Slow Node + low CPU

1.1 Tree of node 6

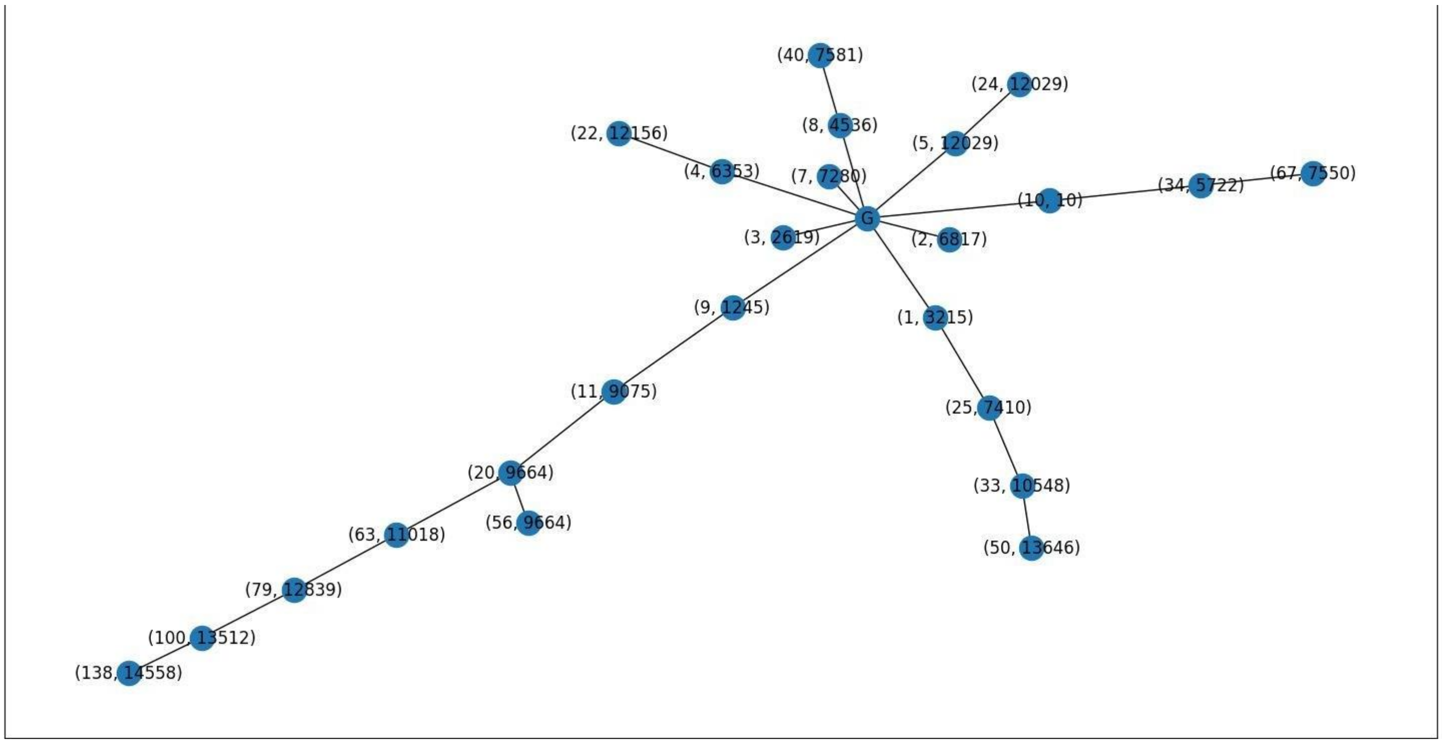


1.2 Longest chain of node 6

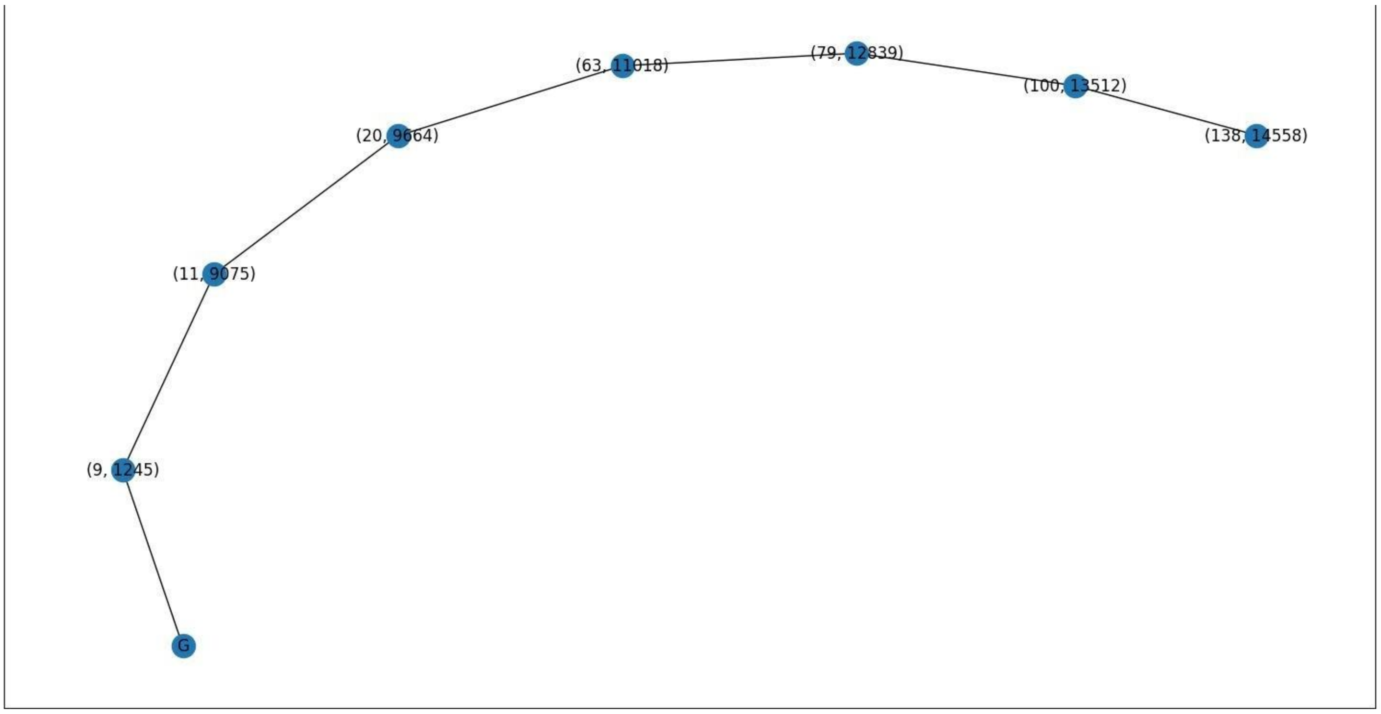


Fast Node + High CPU

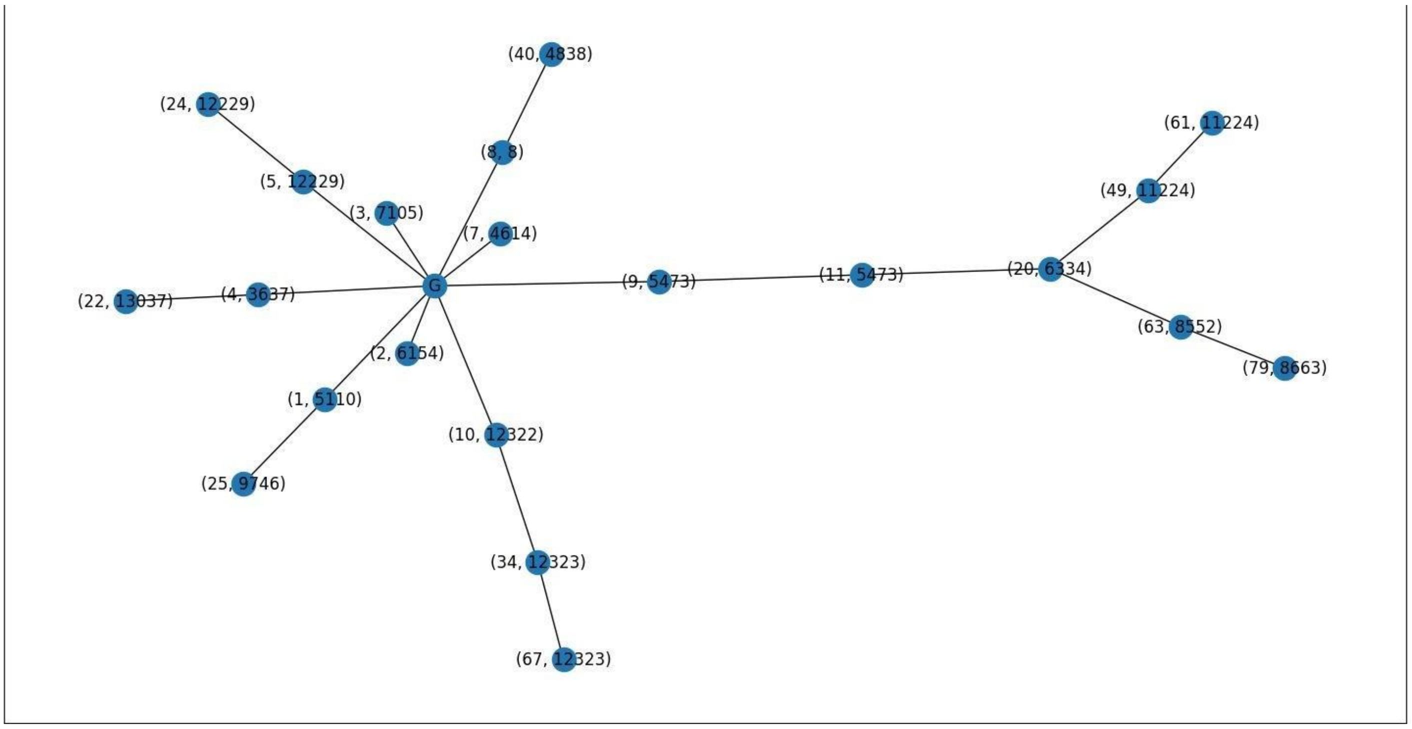
5.1 Tree of node 9



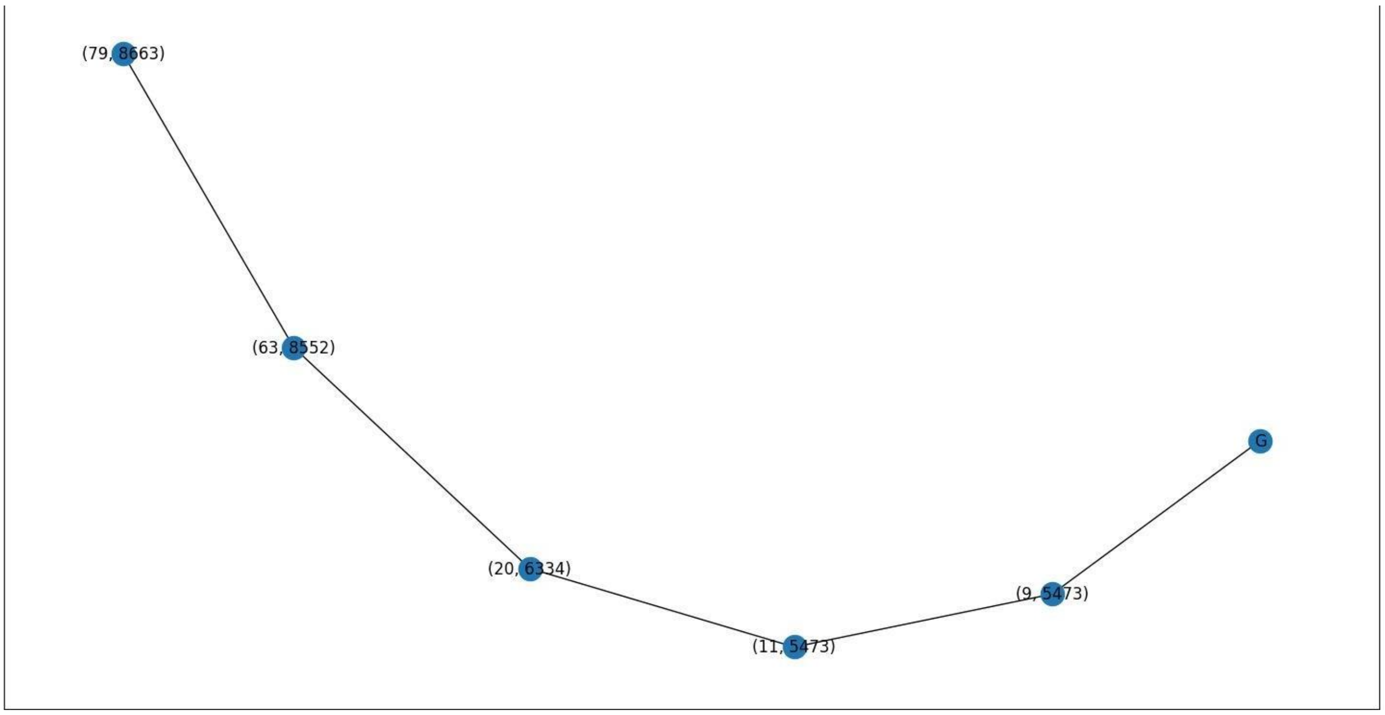
5.2 Longest chain of node 9



6.1 Tree of node 7



6.2 Longest chain of node 7



- **Simulating Selfish Attack**
  - Parameters

    N=number of nodes

    Ttx= Transaction Interarrival time mean

    z0 = Percent of slow nodes

    z1 = Percent of low CPU nodes

    Hashing Power(alpha)

    Fraction of peers that are connected to adversary(zeta)

<u>Observations for selfish Mining</u>

- Hashing power(alpha)=0.15

Zeta	**0.25**	**0.50**	**0.75**
MPU Adversary	0.666666	0.66	0.64
MPU Overall	0.95	0.875	0.8
R_pool	0.105263	0.093	0.125

- Hashing power(alpha)=0.33

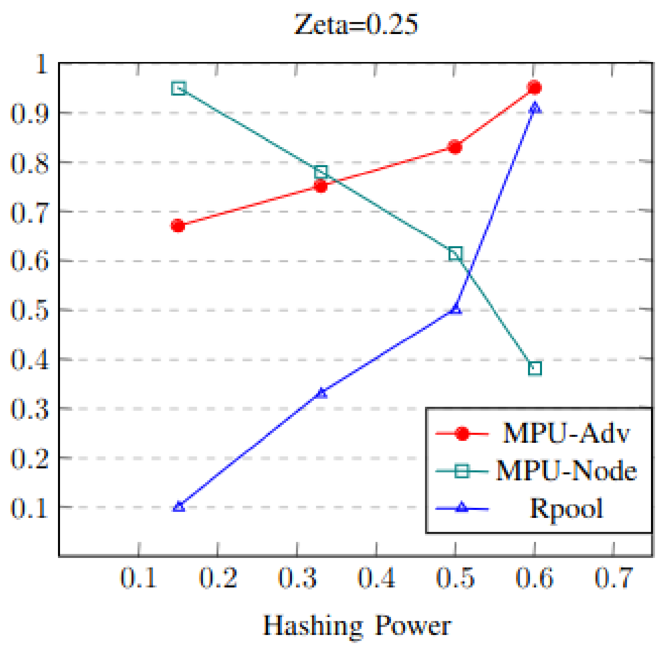
Zeta	**0.25**	**0.50**	**0.75**
MPU Adversary	0.75	0.75	0.66
MPU Overall	0.78	0.80	0.70
R_pool	0.33333333	0.34	0.35

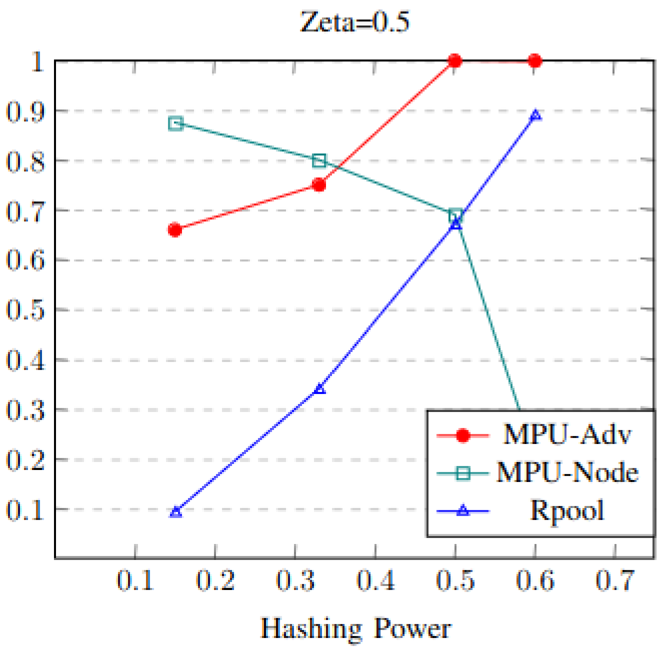
- Hashing power(alpha)=0.50

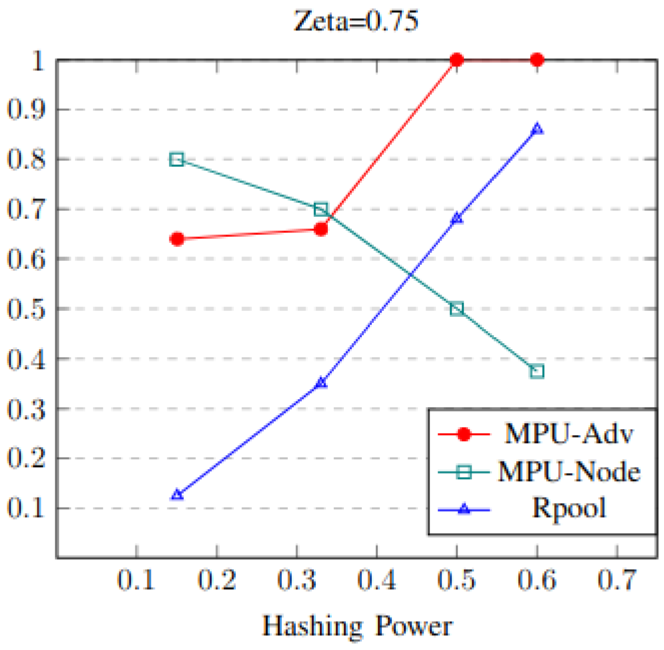
Zeta	0.25	0.50	0.75
MPU Adversary	0.8333	1.0	1.0
MPU Overall	0.615	0.6923076	0.5
R_pool	0.5	0.6666666	0.684210

Hashing power(alpha)=0.60

Zeta	0.25	0.50	0.75
MPU Adversary	0.95	1.0	1.0
MPU Overall	0.382	0.212	0.375
R_pool	0.91	0.89	0.86




Zeta=0.5
Hashing Power
MPU-Adv
MPU-Node
Rpool
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Zeta=0.75
Hashing Power
MPU-Adv
MPU-Node
Rpool
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

<u>Observations for Stubborn Mining</u>

- Hashing power(alpha)=0.15

Zeta	0.25	0.50	0.75
MPU Adversary	0.25	.33	0.52
MPU Overall	0.89	.90	0.905
R_pool	0.052	0.058	0.052

- Hashing power(alpha)=0.33

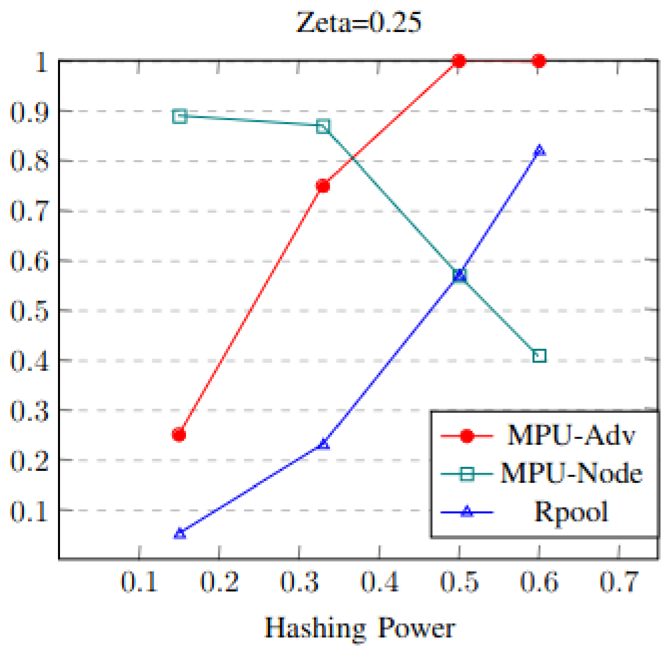
Zeta	0.25	0.50	0.75
MPU Adversary	0.75	0.75	0.80
MPU Overall	0.87	0.78	0.61
R_pool	0.23	0.27	0.44

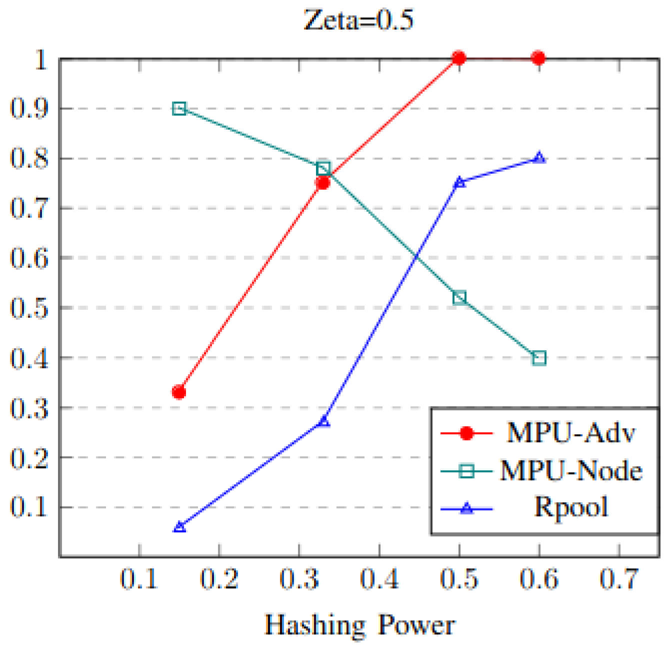
- Hashing power(alpha)=0.50

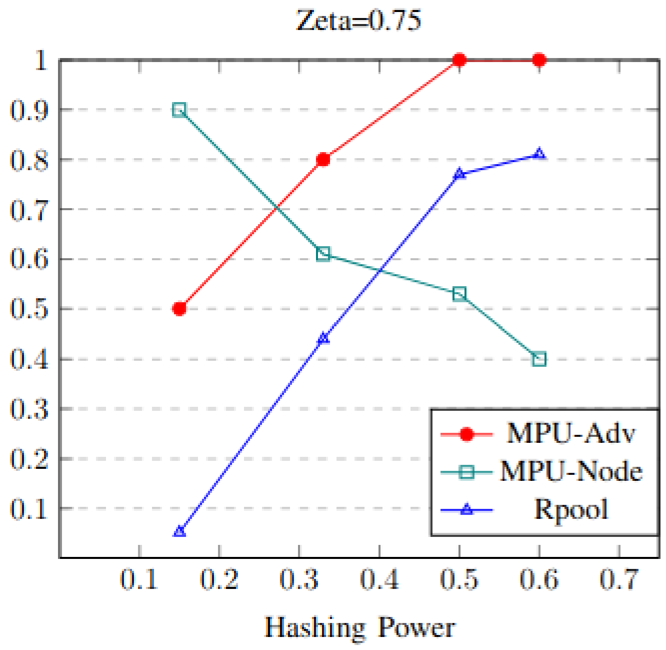
Zeta	0.25	0.50	0.75
MPU Adversary	1	1	1
MPU	0.57	0.52	0.53
Overall			
R_pool	0.57	0.75	0.77

- Hashing power(alpha)=0.60

Zeta	**0.25**	**0.50**	**0.75**
MPU Adversary	1	1	1
MPU **Overall**	0.4	0.42	0.41
R_pool	0.82	0.8	0.81




Zeta=0.5
Hashing Power
MPU-Adv
MPU-Node
Rpool
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

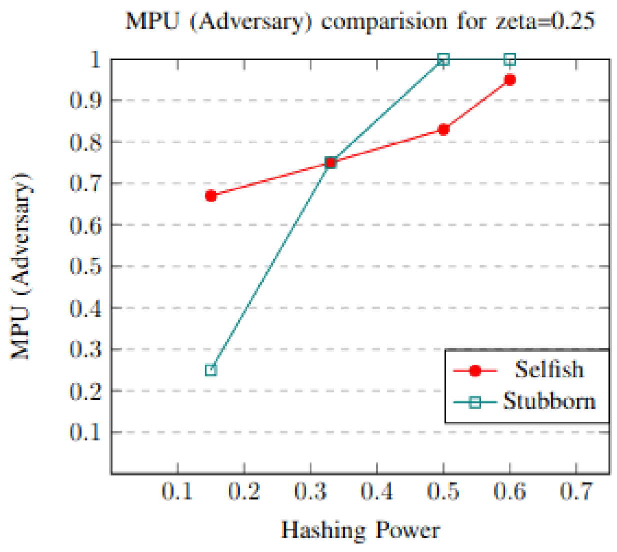

Zeta=0.75
Hashing Power
MPU-Adv
MPU-Node
Rpool
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

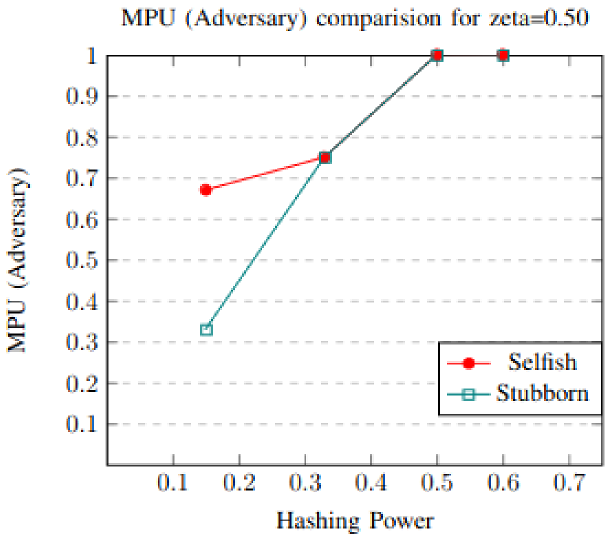
Observation regarding MPU_adv, MPU_overall and R_Pool:

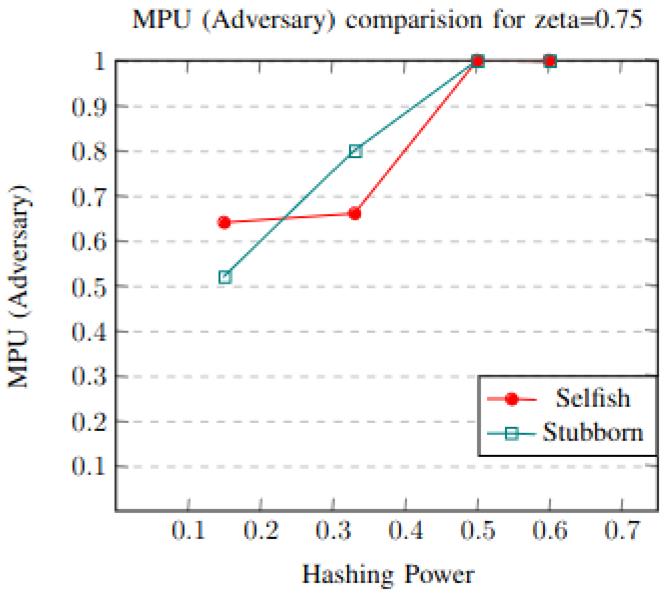
- From the above graphs, we can see that with increased hashing power, the ratio of adversary block in the main chain increases and finally becomes 1. Since the attacker has more power, it has more chances to mine a new block than its peers.
- We see that with increased hashing power, the value of MPU_Node increases as more of the attacker's mined blocks are now becoming part of the public chain. Hence, a lesser number of attackers block are getting into forks, or when in forks, there is a high chance that it gets adopted as public chain in later stages.
- We also observe that MPU_overall decreases with the adversary's hashing power. It is because now more and more of honest miners block's are getting abandoned as the adversary releases a private block to complete with honest miners' block. Due to lesser power than the adversary, the next block on the honest block gets mined later, and hence by the time it gets mined attacker has already mined a block and taken some lead, and thus this honest block gets abandoned.

Variation of MPU Ratio with hashing power for both attacks

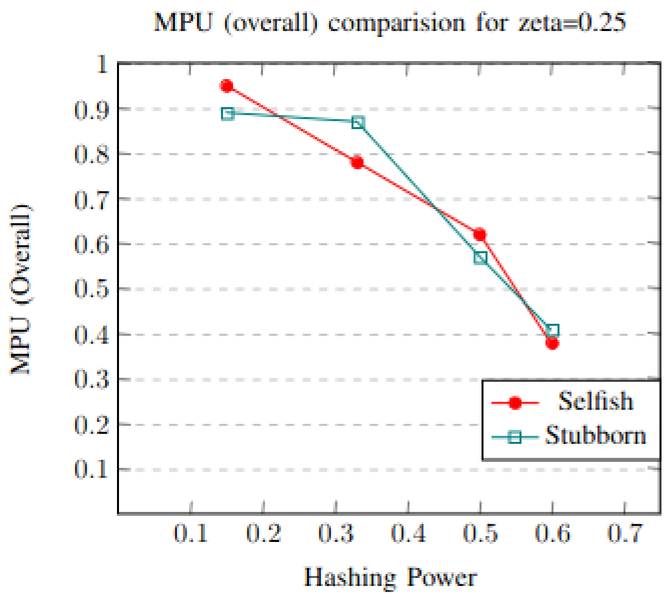
1. **MPU (Adversary) for varying hashing power.**

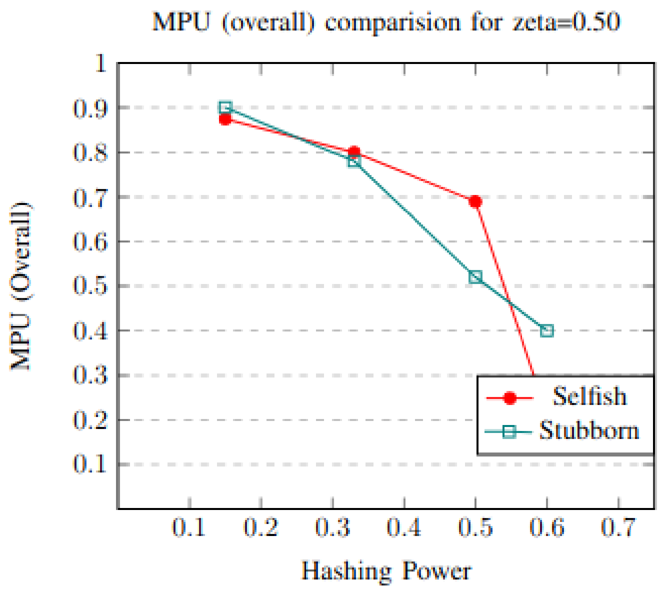


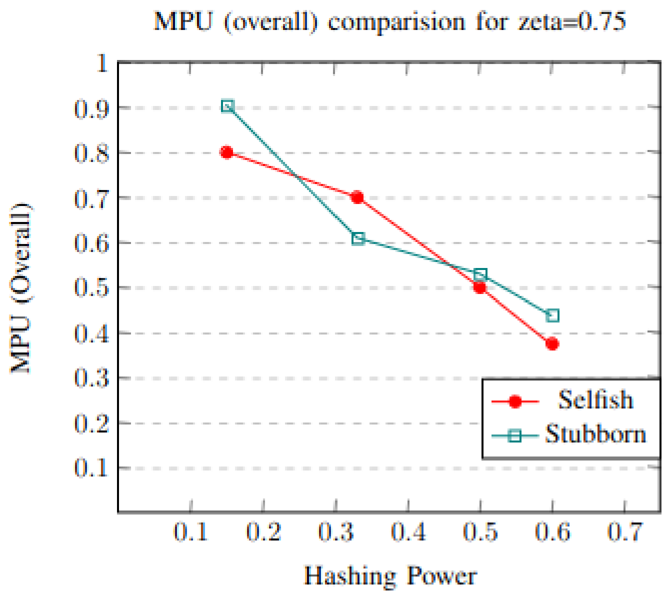

MPU (Adversary) comparision for zeta=0.50
MPU (Adversary)
Hashing Power
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Selfish
Stubborn


MPU (Adversary) comparision for zeta=0.75
MPU (Adversary)
Hashing Power
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Selfish
Stubborn

2. MPU (Overall) for varying hashing power.







We see the following details regarding the Selfish mining attack: MPU overall trend and MPU adv have contrasting patterns. MPU adv grows as the adversary's hashing power increases. This is expected because as the attacker's mining capacity increases, more and more of its blocks are added to the main chain. After all, it can quickly wipe out the honest blocks. If more blocks produced by honest miners are discarded, the number of honest blocks in the main chain will drop, which will have a negative impact on the MPU overall.

In the case of stubborn mining, when the adversary has lesser hashing power and hence will mine blocks at a slower rate. Since the attacker does not release multiple blocks to win the race, then there is a high chance that due to the equal length of the forked public chain, it is possible that the honest miners keep on mining the public block seen earlier. Hence, there are higher chances that attackers mined blocks are not getting added to the public chain, and hence the private chain of the attacker gets orphaned at a higher rate. Therefore MPU adversary is increasing at lower rate then then that of the selfish miner.

Conclusions

On changing hashing power, we observe the following:

- As hashing power increases, the value of MPU node_adv increases on both selfish and stubborn mining.
- As hashing power increases, the value of MPU node_overall decreases on both selfish and stubborn mining.
- As hashing power increases, the ratio of adversary's blocks in the main chain increases.

On changing the fraction of honest nodes ζ, an adversary is connected to:

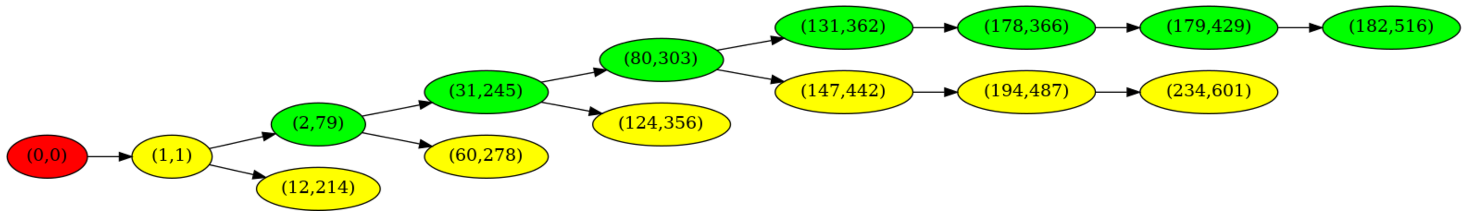
- When Hashing power is low, then there is no significant pattern observed on both MPU node_adv and MPU node_overall
- When Hashing power is more then the MPU node_adv reaches 1.0 at exponentially, and MPU node_overall is low in almost every ζ

Pictures Of Blockchains for some particular cases

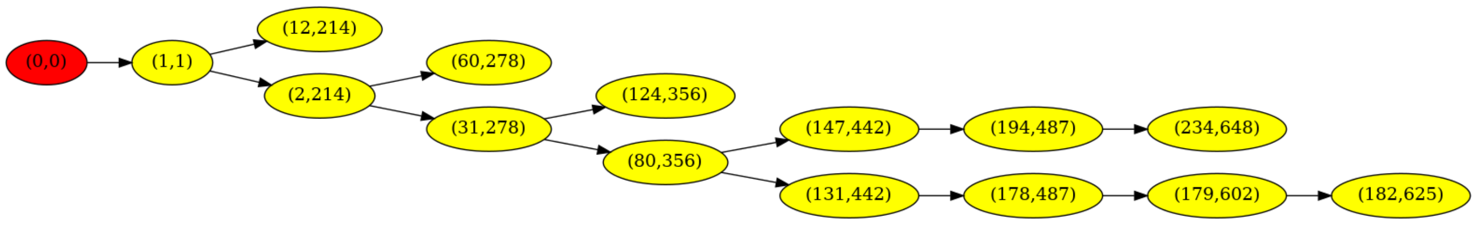
1. Stubborn Mining

a. n=25, Ttx=150, z0=50, z1=10, alpha=50, zeta=75

i. Adversary node

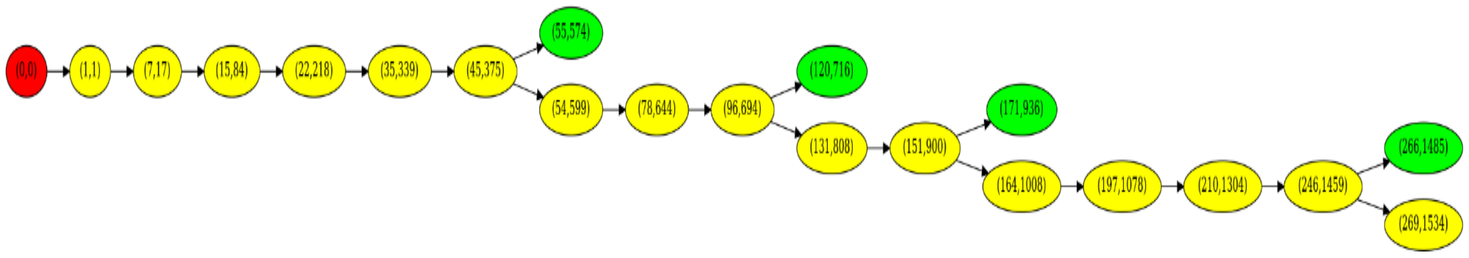


ii. Honest Node

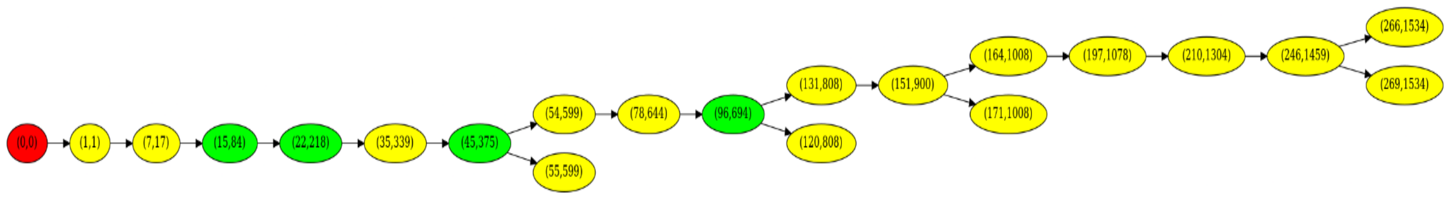


b. n=25, Ttx=150, z0=50, z1=10, alpha=15, zeta=50

i. Adversary Node

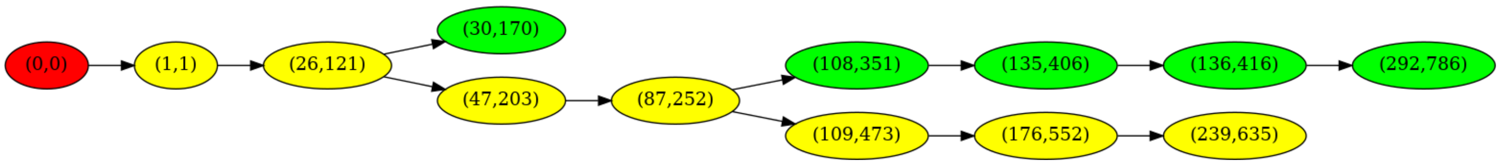


ii. Honest Node

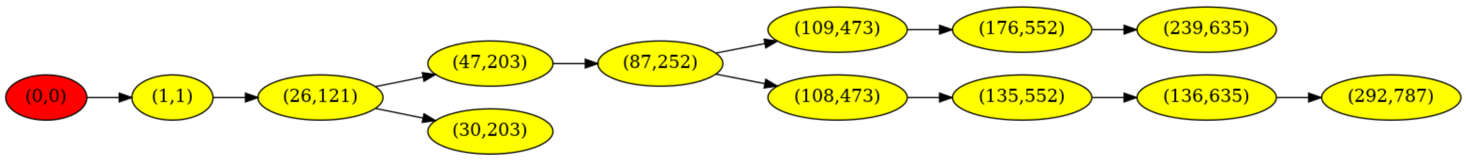


c. **n=25, Ttx=150, z0=50, z1=10, <u>alpha=33, zeta=25</u>**

i. Adversary Node



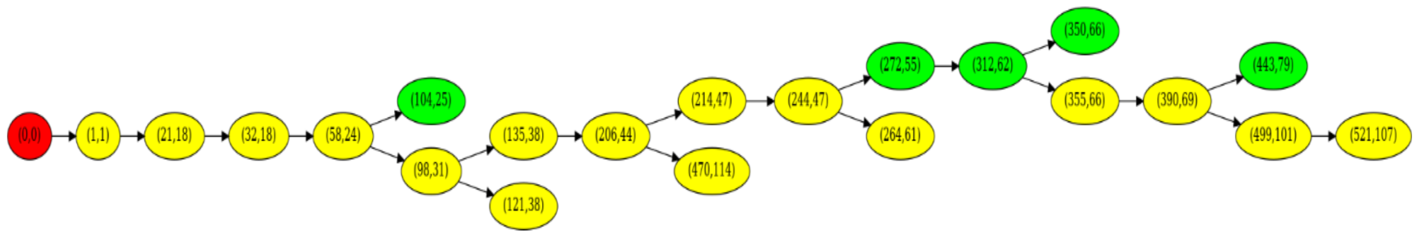
ii. Honest Node



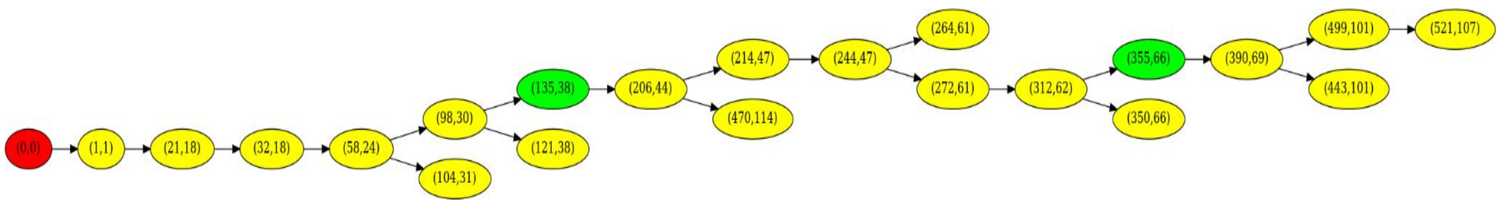
2. **Selfish Mining**

a. **n=25, Ttx=150, z0=50, z1=10, <u>alpha=15, zeta=50</u>**

i. Adversary Node

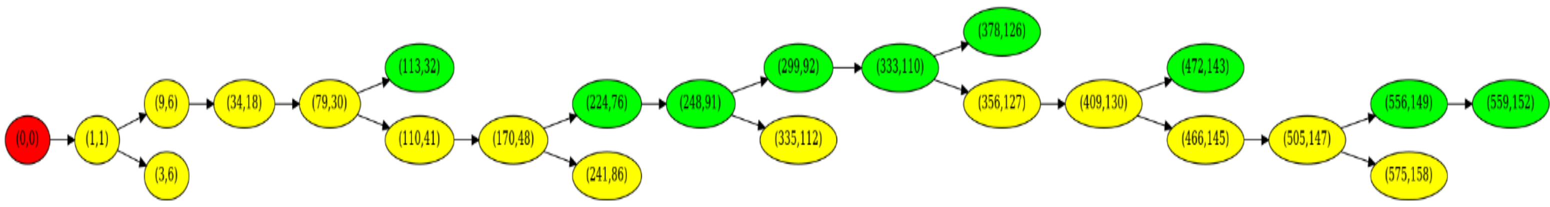


ii. Honest Node

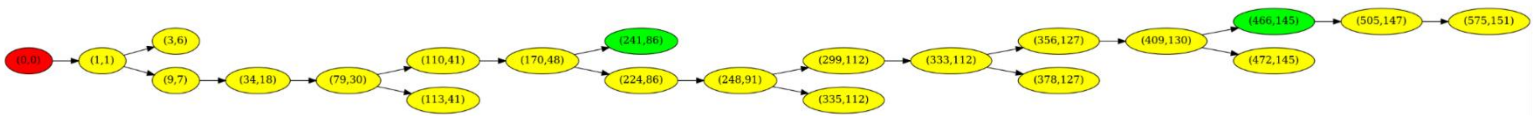


b. **n=25, Ttx=150, z0=50, z1=10, <u>alpha=33, zeta=50</u>**

i. Adversary Node

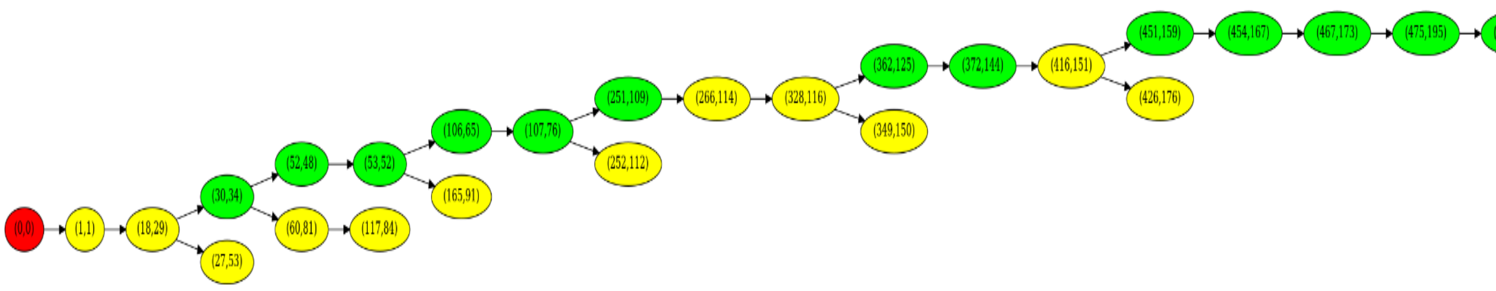


ii. Honest Node

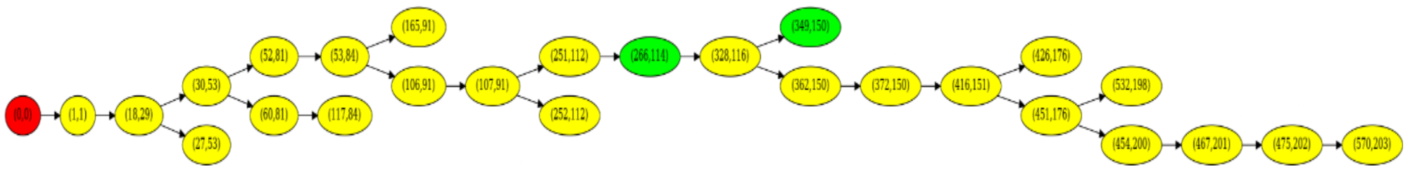


c. **n=25, Ttx=150, z0=50, z1=10, <u>alpha=50, zeta=75</u>**

i. Adversary Node



ii. Honest Node


(0,0)
(1,1)
(18,29)
(30,53)
(27,53)
(52,81)
(60,81)
(53,84)
(117,84)
(165,91)
(106,91)
(107,91)
(251,112)
(252,112)
(266,114)
(328,116)
(349,150)
(362,150)
(372,150)
(416,151)
(426,176)
(451,176)
(532,198)
(454,200)
(467,201)
(475,202)
(570,203)