COL819 A3: Bitcoin Implementation

Sameer Vivek Pande
2017CS10371

Rajat Singh
2020CSZ8507

May 2021

1 Introduction to Bitcoin

Bitcoin is a crypto-currency based on purely peer to peer distributed system which was invented in 2008 by a group of people using the name Satoshi Nakamoto. To prevent the fake transactions and the double spending problem, it is implemented in such a way that there is no central authority or banks which is managing the transactions thus the transactions and issuing of bitcoins is carried out collectively by the network. This network runs on trust between the nodes present in the network and various validity checks. Bitcoin is a kind of digital currency with more security , transparency and no reversal of the transaction.

In this Assignment we tried to implement simple bitcoin crypto-currency system with multi- transaction support, it consists of various integrity checks for transactions such as digital signatures , merkle tree for transactions in blocks and proof of work is used to verify block before adding them to the block-chain. Here is the list of some basic implementations that we used in this assignment.

1. All the nodes are implemented as separate process and in each process we used a thread with will carry-out all the transactions and mining of the block. Initially all the nodes are assigned with the public key and the private key. Each node will share their public key with the rest of the nodes present in the network.

2. Each node is consist of a miner which will start mining the blocks as soon as genesis block is created.

3. In this implementation we have fixed the node which will create the Genesis block and that node is the node with Node id 0.

4. Once the nodes are up for transaction they will perform the transactions (multi-transaction) and these transactions will be signed by private key of sender and broadcast the transaction to all the nodes.

5. All nodes will receive these transactions and save these transactions in a list. Miners of each node will collect these transactions and validate them, and compete with each other for creation of the block.

6. Miners will collect the transaction for a constant amount of time. After that they will create the block and start computing the proof of work.

7. Once the Miner complete the computation of proof of work it will check if any other node has already computed it before the current node or not. If current node is the winner then it will send that block to all the nodes and all the nodes will check its validity and add that block in their local block-chain. Else if some other node have already computed that proof of work before than the current node then it will take the computed block, check its validity and add the block to its block-chain.

8. Meanwhile if some node try any double spending then other nodes will verify the block-chain by checking the longest chain and only go with the longest chain for addition of further blocks. Thus this prevent the nodes form any faulty transactions and all the previous transactions are verified.

There are 7 different types of messages used for communication between the nodes/blocks/transactions in this implementation . We will try to give short description of all the messages :

1. **GENESIS-BLOCK**: This type of message is used by nodes to broadcast the Genesis block to other nodes available in the network. In this implementation this Genesis block is mined and send by node 0.
2. **NORMAL-BLOCK**: This type of message is used by nodes to broadcast the a Normal block to other nodes available in the network. Normal blocks are the blocks which are not Genesis block and can be broadcast by any of the available node in the network.
3. **BLOCK**: This type of message is used by the nodes to broadcast the created blocks to other nodes in the network.
4. **REWARD**: This is for categorizing the transaction as the reward transaction. The reward transaction is created by using block creation reward.
5. **COIN-BASE**: This is for categorizing the transaction as a coin-base transaction. Coin-base transactions are those transactions that are created when there is no source of creation of the coin i.e. no Inputs.
6. **COIN-TXN**: his is for categorizing the transaction as the simple transactions between two nodes.
7. **TXN** : This type of message is used by nodes to broadcast the normal transaction to other nodes in the network.

The detailed Implementation of the Bticoin is covered in section 3 (Implementation Details).

2 How to run the code

To run the code use the command given below.

```
python3 assignment3_2017CS10371_2020CSZ8507/bitcoin.py <Number of nodes> <proof
of work zeros> <arity>

For example :

python3 bitcoin.py 10 4 2
```

Where **Number of nodes** is the total number of the nodes present in the bitcoin network, **proof of work zeros** denotes the difficulty level(number of zeros in the hash) while computing the proof of work, and **arity** is used to create the merkle tree.

3 Implementation Details

3.1 Configuration

We have implemented the GHS algorithm in Python3.6.x. We have used the following Packages in our implementation.

1. **Multiprocessing**: This package is used to create multi-threaded environment (multiprocessing pool) to simulate distributed system, with each node on a separate thread.

2. **Crypto** : This package is a collection of both secure hash functions , and various encryption algorithms (AES, DES, RSA, etc.).

3. **random** : This package is used to create random numbers.

4. **time**: This package is used to detect the running time of the algorithm.

5. There are various other basic packages have been used while implementing the algorithm such as **os, math, enum, sys,** etc.

The specification of the system used to run and analyse the algorithm is as follows:

1. **Operating System** : MacOS Big Sur (Version 11.3.1)

2. **Ram** : 8 GB

3. **Processor** : Intel Core i5 2.3GHz

4. **Number of Cores** : 2

3.2 util.py

The "util.py" file consist of the configuration functions and basic crypto functions that are used for the bitcoin implementation. Some of the functions are given below.

1. **util.create_hash()**
   This function is used to create a hash of an binary message and return the hash in hexadecimal. Here we are using SHA256 to create the hash of the message.

```
def get_enc():
    return SHA256.new()

def create_hash(s):
    uni = s.encode()
    hash_object = get_enc()
    hash_object.update(uni)
    hex_dig = hash_object.hexdigest()

    if get_debug():
        print(hex_dig)

    return(hex_dig)
```

Listing 1: **util.py:** create hash of the message

2. **util.public_private_key()**
   This function is called by each and every node present in the network to get a random generated public and private key pairs for that node. It is important to assign public and private key to each node so that nodes can communicate with each other securely.

```
def public_private_key():
    rng = Random.new().read
    private_key = RSA.generate(1024, rng)
    public_key = private_key.publickey()

    if get_debug():
        print("# Type of private key and public key")
        print("# ",type(private_key), type(public_key))
    private_pem = private_key.export_key()
    public_pem = public_key.export_key()
    return (private_pem, public_pem)
```

Listing 2: **util.py:** Generate the public and private key

3. **util.create_merkle_tree()**
   This function is called by each block class to make the merkle tree for its transactions. This function will create the merkle tree and return the merkle tree in form of list where the last element of the list is the head of the merkle tree.

```
def create_merkle_tree(tot_leaves,leaf_sz):
    leaves = tot_leaves
    len_leaves = len(leaves)
    if len_leaves % leaf_sz != 0 :
        leaves_to_add = leaf_sz - (len_leaves % leaf_sz)
        len_leaves +=leaves_to_add
        for i in range(leaves_to_add):
            leaves.append(leaves[-1])
    merkle_tree = [leaves]
    ..........
```

Listing 3: **util.py:** Creating the merkle tree

4. **Other configuration functions**
   In "util.py" there are various configuration function available from which we can customise the bitcoin implementation. Here **get_initial_amount()** function is used to assign the initial money to each node. **get_block_create_reward()** is used to change the block reward when a miner creates a block. **get_block_create_time()** is used to change the time of addition of a block in the block-chain. **get_transaction_charges()** is used to change the transaction charges for each transaction. etc.

```
def get_initial_amount():
    return 5000
def get_block_create_reward():
    return 5
def get_block_create_time():
    return 20
def get_transaction_charges():
    return 2
def get_smart_contract():
    return True
def get_smart_contract_nodes():
    dict_nodes  = {}
    dict_nodes[0] = 1
    return dict_nodes
def get_smart_contract_balance():
    return 3000
def get_smart_contract_deduction():
    return 50
def get_multi_smart_contract():
    return False
def get_multi_transactions():
    return True
def get_random_multi_transactions():
    return random.randint(1,4)
def get_random_amount():
    return random.randint(1, 100)
def find_random_reciever(tot_nodes,curr_node):
    r = random.randint(0,tot_nodes-1)
    while (r == curr_node):
        r=r+1
        r= r%tot_nodes
    return r
def get_total_time():
    return 50
def print_logs():
    return True
def get_debug():
    return False
```

Listing 4: **util.py:** Other configuration functions

3.3 bitcoin.py

This is the core file from where the bitcoin implementations starts. In this file we have created the given number of nodes and assigned each node with distinct process. These process takes a shared queue which will help in further communication between the nodes.

3.4 Node.py

This file contains the maximum part of the implementation of the bitcoin algorithm. The structure of the class is given below. This class contain **run_node()** function from where a process of node will start. The first thing this function will do is it will share its public key with the other available nodes. After sharing the public key node 0 will create the genesis block and share the created block to other available nodes. After sharing the genesis block it will start a thread which will create automatic random transactions and mine the new block in parallel.

```
class Node():
    def __init__(self,idx,node_cnt,pow_zeros,leaf_sz,message_common_list,
    message_limit,token):
        self.idx = idx
        self.node_cnt = node_cnt
        self.pow_zeros = pow_zeros
        self.leaf_sz = leaf_sz
        self.message_common_list = message_common_list
        self.message_limit = message_limit
        self.token = token
        if(len(self.token)==0):
            self.token.append(True)
        self.block_create_reward = util.get_block_create_reward()
        self.block_create_time = util.get_block_create_time()
        self.private_key,self.public_key = get_keys()
        self.hash_public_key = get_hash(self.public_key)
        self.bitcoin = 0
        self.unspent_bitcoin = {}
        self.transaction_charges = util.get_transaction_charges()
        self.start_time = time.time()
        self.dict_public_key_nodeid = {}
        self.nodeid_public_key = []
        self.prev_trans_key = []
        self.prev_trans_verified = False
        self.prev_trans_time = 0
        self.kk = True
```

Listing 5: **Node class initialization**

3.5 Transaction.py

This class contain the implementation regarding the Transactions. It contain **sign_transaction()** function is used to sign the message which is generated by **txn_message()** function. Transaction class also contain the function that is used for verification of the transaction such as **check_sign_transaction()**

```
class Transaction():
    def __init__(self, txn_input, txn_output, public_key, private_key, txn_type):
        self.txn_output = txn_output
        self.txn_input = txn_input
        self.public_key = public_key
        self.private_key = private_key
        self.txn_type = txn_type
        self.txn_time = time.time()
        self.txn_sign = self.sign_transaction()
        self.txn_id = self.getTxn_id()
```

Listing 6: **Transaction class initialization**

3.6 Block.py

This file contain the Block class which have functions related to a block such as **merkle_tree()** which takes all the transactions and create the merkle tree out of these transactions. This class also contain **proof_of_work()** function that will calculate the proof of work and find the correct nounce. This class also contain some methods to verify the block such as **check_pow_zeros()** and **is_valid_proof_of_work()**.

```
class Block():
    def __init__(self, pow_zeros, leaf_sz, transaction, b_type, prev_block_hash,
    index):
        self.pow_zeros = pow_zeros
        self.leaf_sz = leaf_sz
        self.transaction = transaction
        self.b_type = b_type
        self.prev_block_hash = prev_block_hash
        self.merkle_tree = self.merkle_tree()
        self.index = index
        self.current_block_hash = ''
        self.root_merkle_tree = self.get_root()
        self.current_block_hash,self.nounce = self.proof_of_work()
        self.parent = None
        self.child = []
```

Listing 7: **Block class initialization**

3.7 blockchain.py

The main task of Blockchain class is to store the chain of block that are created by the miners. The Blockchain class contain the function **add_genesis_block()** which is used to add a new generated block to the blockchain. It also contain **isValid_transaction()** to verify the transactions of the block before adding it to the blockchain. This class also take the record of the unspent transactions for verification in future.

```
class Blockchain():
    def __init__(self, pow_zeros, leaf_sz):
        self.pow_zeros = pow_zeros
        self.leaf_sz = leaf_sz
        self.blockchain = []
        self.utxo = set()
        self.utxo_mapping = dict({})
        self.index_of_confirmed_block = None
        self.height_of_current_block = None
```

Listing 8: **Blockchain class initialization**

3.8 miner.py

This file contain the Miner class which contains the function related to the mining of the block. It contains **is_valid_block()** which checks whether the block to be added is valid or not before adding the new block using **add_new_block()** function. Miner is also responsible for the verification of the new blocks that are mined by some other nodes and sent to the current node.

```
class Miner():
    def __init__(self, idx, node_cnt, pow_zeros, leaf_sz, private_key,
    node_public_key, block_create_reward, block_create_time, transaction_charges):
        self.idx = idx
        self.node_cnt = node_cnt
        self.pow_zeros = pow_zeros
        self.leaf_sz = leaf_sz
        self.private_key = private_key
        self.node_public_key = node_public_key
        self.block_create_reward = block_create_reward
        self.block_create_time = block_create_time
```

```
        self.transaction_charges = transaction_charges
        self.current_transactions = []
        self.blockchain = None
```

Listing 9: **Miner class initialization**

4 Result Analysis

4.1 Effect of changing the Hash Function

Since we are using hash function to create the merkle tree, to create id of block, etc. thus hash function plays a very important role in terms of time and space used for the creation of block. In this section we will compare various hash types for block creation time and space.

Note: For this experiment we have fixed the number of nodes to 10, proof of work zeros to 4 , 5 transactions per block and, arity of mrekle tree to 2.

4.1.1 Block size vs Hash Type

Different hash type functions create different size hash thus if we use a hash function which create the hash of more bits will consume more space as shown in the Figure 1. Here SHA1 create the hash of 20 bytes which causes the block to consume less sapace as compared to SHA512 which create the hash of 32 bytes.

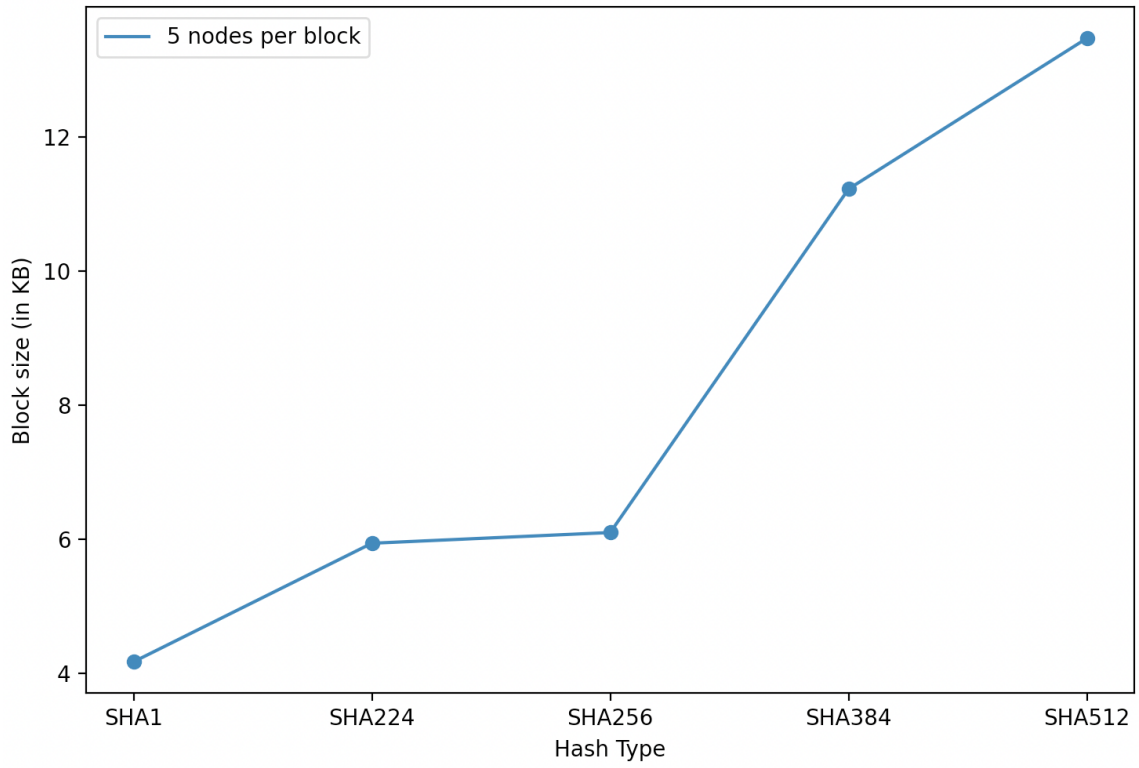

Figure 1: Plot between Block size vs Hash Type

4.1.2 Block creation time vs Hash Type

Different hash type functions are different in complexity and they take different time to create the hash of the message. Thus this causes change in processing time as the hash type is changed as shown in the Figure 2. From the figure we can conclude that the time taken to create the block if we are using SHA512 hash function is greater than the time taken by the block using SHA1. These time can vary from system to system as background process and system configuration effect these time to a great extent.

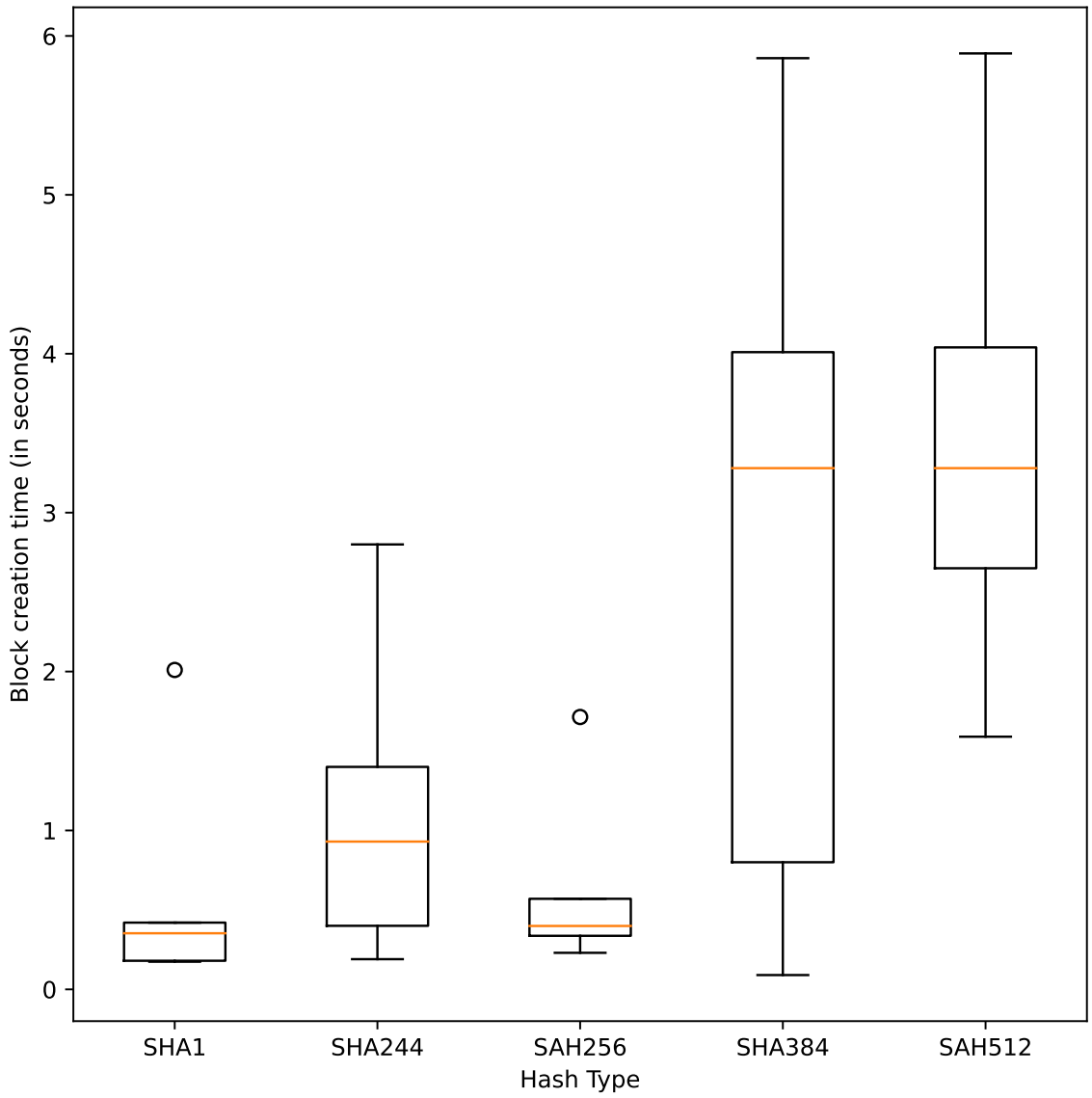

Figure 2: Plot between Block creation time vs Hash Type

4.2 Effect of changing the Arity of merkle tree

Since the size of merkle tree is proportional to the Arity of the merkle tree. Thus when we decrease the Arity the depth of merkle tree will increase and thus increase in list size which will ultimately increase the total size of the node. For the Figure 3

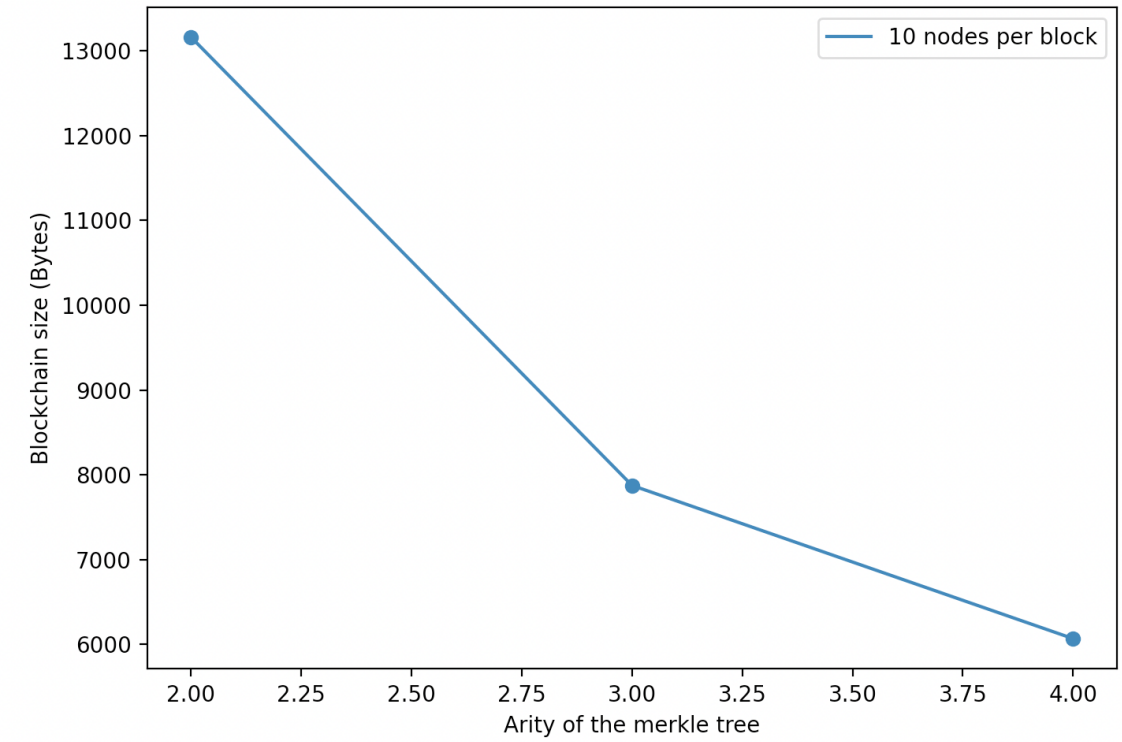

Figure 3: Plot between Blockchain size vs Arity of merkle tree

The above figure show the behaviour of size of the block chain if we change the arity size of the merkle tree. Point to note here is that we are using sha-256 hashing function during this experiment.

Figure 4 show the plot of time taken to create a block vs arity of the merkle tree. For this experiment we have used sha-256 hashing function and number of zero in the proof of work are 4. In the Figure 4 we are not able to get any standard pattern because the effect of time is altered by the proof of the work and the process that that running in our local system. But the ideal graph should be decreasing with the increase in arity in merkle tree as the depth of the tree decreases thus there will be less intermediate nodes in the merkle hash tree.

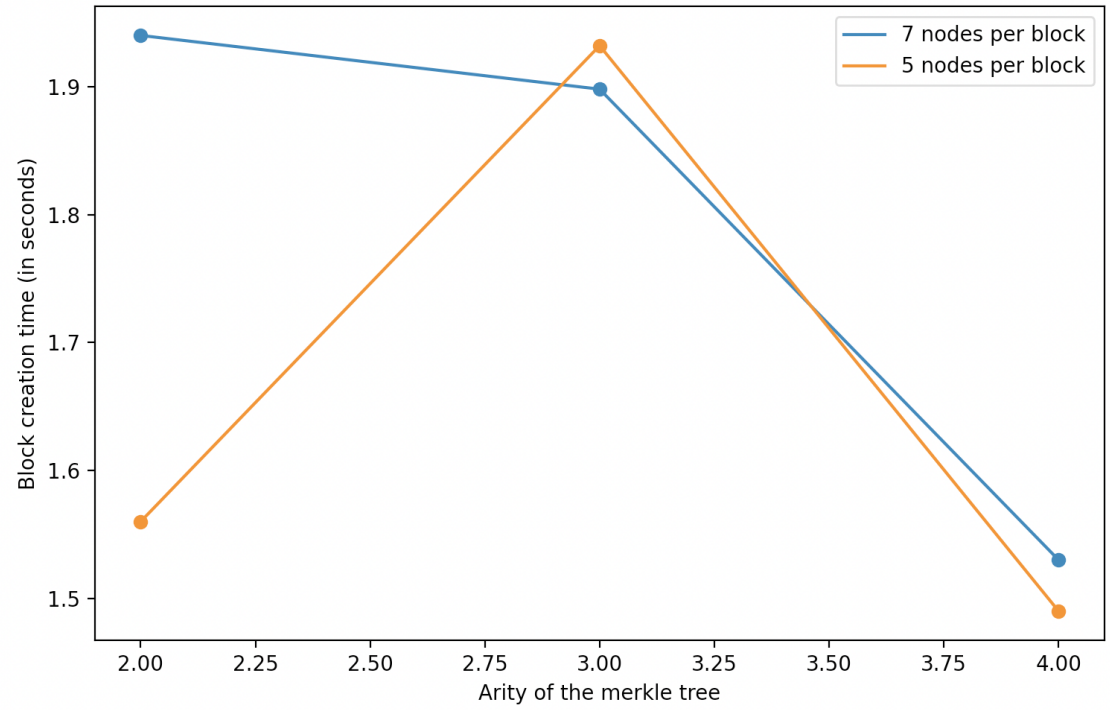

Figure 4: Plot between block creation time vs Arity of merkle tree

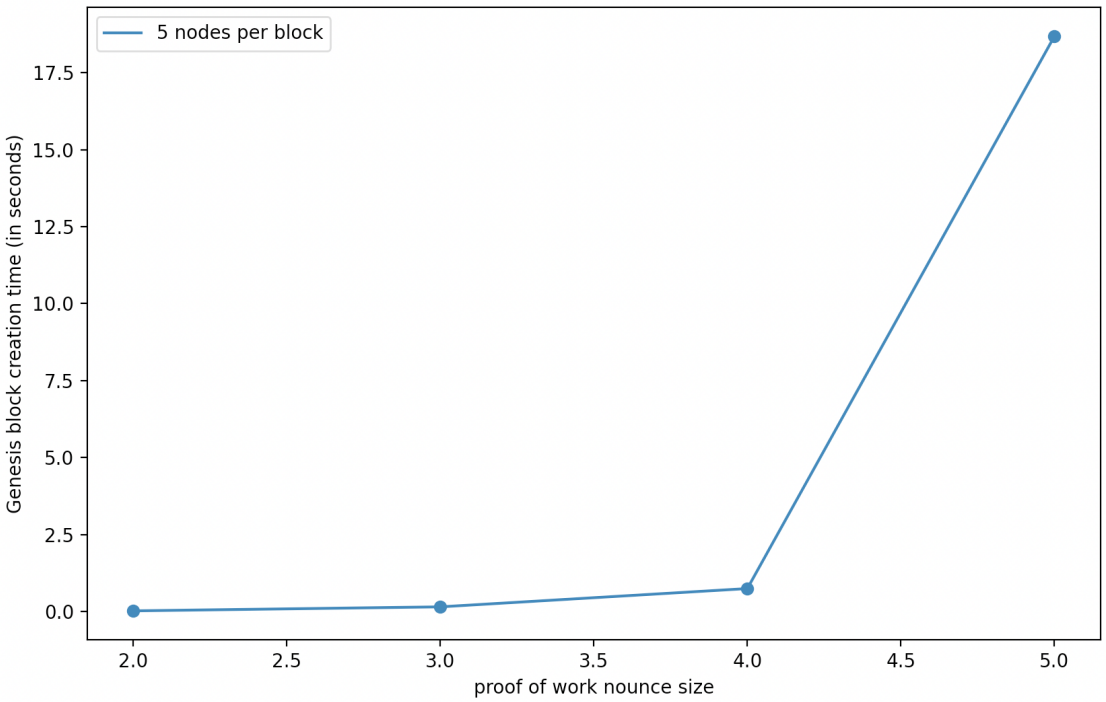

Figure 5: Plot between Genesis block creation time vs Proof of work nounce size

4.3 Effect of change of proof of work size

With the change in the Proof of work the difficulty for the verification of the block also changes. If the proof of work nounce size is increased then node have to do more computation to find the correct hash as shown in Figure 5 and Figure 6.

Note : For this experiment we have fixed the maximum number of transactions per block to 5, fixed the number of nodes to 10, and, arity of mrekle tree to 2.
We have presented the comparison of genesis block creation vs proof of work nounce size and simple block creation and nounce size separately because as we have already stated that during creation of genesis block only node 0 is doing computation and during normal block creation all the nodes must be doing the computation and thus this effect the time of creation of these blocks. In figure 6 we are not able to show the time taken to create the normal block for proof of work nounce at 6 as it was taking huge amount of time.

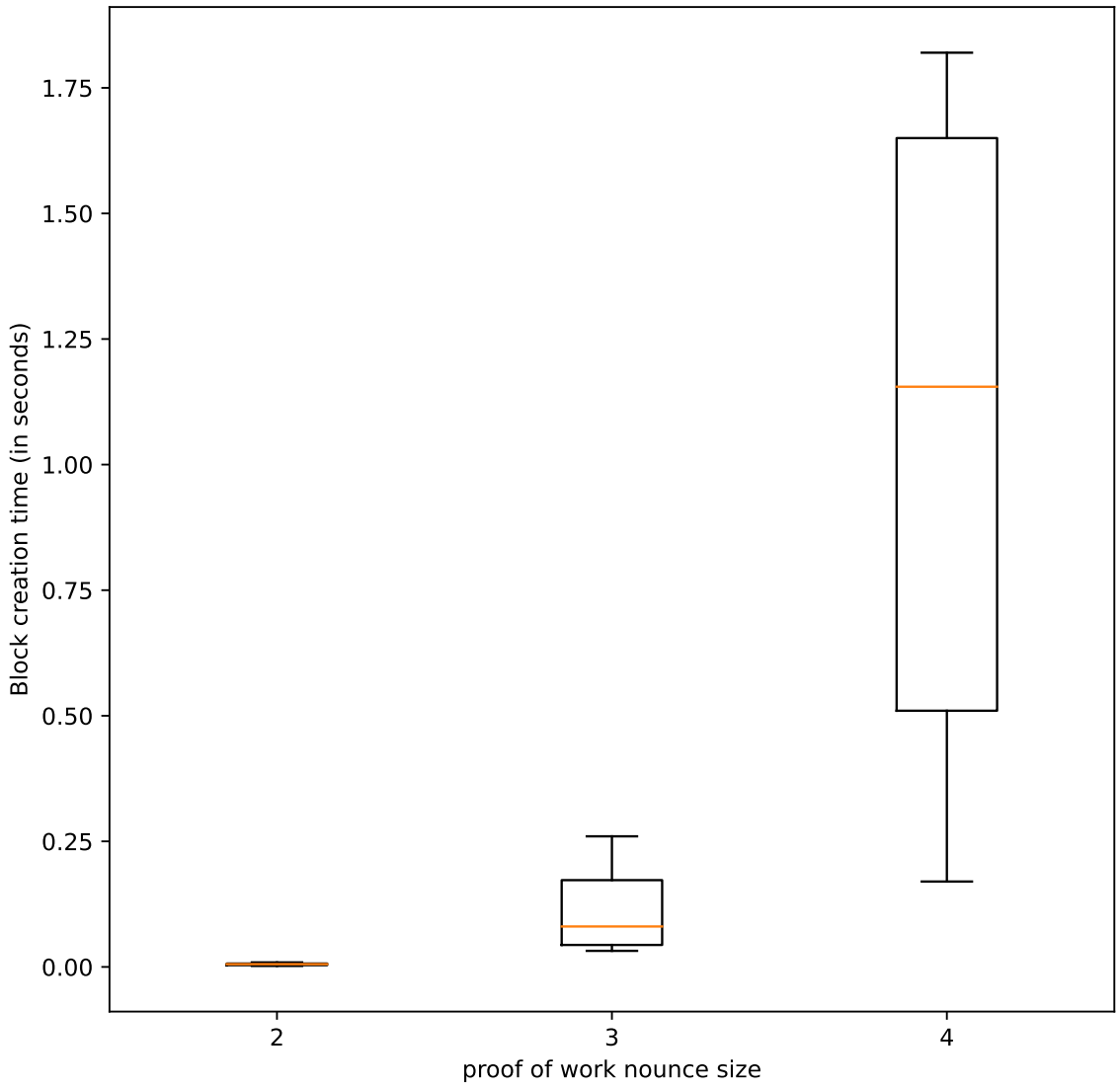

Figure 6: Plot between normal block creation time vs Proof of work nounce size

Size of the block will not change if we change the proof of work nounce size as it dose not affect the hash or anything which effect the size of the block as shown in Figure 7.

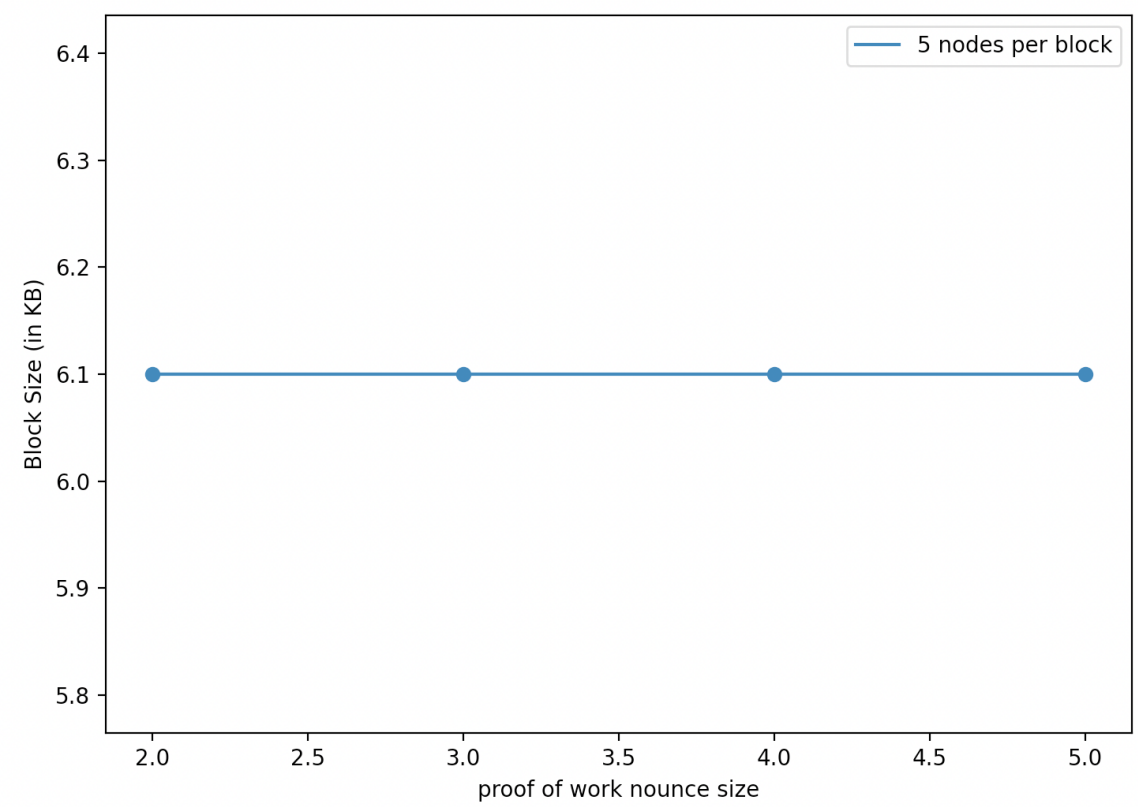

Figure 7: Plot between block size vs Proof of work nounce size

5 Transaction simulation

```
Node 0 is being created.
Node 1 is being created.
Node 2 is being created.
Node 3 is being created.
Node 4 is being created.
Node 5 is being created.
Node 6 is being created.
Node 7 is being created.
Node 8 is being created.
Node 9 is being created.
Node 0 is responsible for creating genesis block.
Time taken to create Genesis block :  0.2560770511627197
Node 0 recieved 5000 btc as initial node amount.
Node 0 recieved 5 btc as Genesis block creation reward.
Node 0 is pushing the genesis block to the stack of node :  1
Node 0 is pushing the genesis block to the stack of node :  2
Node 0 is pushing the genesis block to the stack of node :  3
Node 0 is pushing the genesis block to the stack of node :  4
Node 2 recieved the Genesis Block from Node 0.
Node 0 is pushing the genesis block to the stack of node :  5
Node 0 is pushing the genesis block to the stack of node :  6
```

```
Node 1 recieved the Genesis Block from Node 0.
Node 0 is pushing the genesis block to the stack of node :  7
Node 2 recieved 5000 btc as initial node amount.
Node 3 recieved the Genesis Block from Node 0.
Node 0 is pushing the genesis block to the stack of node :  8
Node 1 recieved 5000 btc as initial node amount.
Node 3 recieved 5000 btc as initial node amount.
Node 0 is pushing the genesis block to the stack of node :  9
Node 5 recieved the Genesis Block from Node 0.
Node 7 recieved the Genesis Block from Node 0.
Node 4 recieved the Genesis Block from Node 0.
Node 5 recieved 5000 btc as initial node amount.
Node 7 recieved 5000 btc as initial node amount.
Node 4 recieved 5000 btc as initial node amount.
Node 9 recieved the Genesis Block from Node 0.
Node 8 recieved the Genesis Block from Node 0.
Node 6 recieved the Genesis Block from Node 0.
Node 9 recieved 5000 btc as initial node amount.
Node 8 recieved 5000 btc as initial node amount.
Node 6 recieved 5000 btc as initial node amount.
Node 5 is sending 77 btc to Node 2 .
Node 4 is sending 100 btc to Node 8 .
Node 9 is sending 36 btc to Node 4 .
Node 7 is sending 90 btc to Node 8 .
Node 6 is sending 23 btc to Node 0 .
Node 2 is sending 40 btc to Node 7 .
Node 2 is started creating the block for the following transactions:
Block creation time : 0.1105489730834961
size of blockchain : 13109
Node 2  Checking if some other node has already created the current block.
# Node 2 Sending the block to all the nodes.
# This is new block message at Node = 4
# This is new block message at Node = 6
# This is new block message at Node = 7
# This is new block message at Node = 8
# This is new block message at Node = 9
# This is new block message at Node = 0
# This is new block message at Node = 3
# This is new block message at Node = 5
# This is new block message at Node = 1
Node 2 recieved 77 from Node 5
Node 2 recieved 4958
Time taken by Node 2 for transaction verification is 3.1677370071411133
Node 8 is sending 8 btc to Node 2 .
Node 3 is sending 12 btc to Node 7 .
Node 5 is sending 40 btc to Node 7 .
Node 6 is sending 39 btc to Node 1 .
Node 2 is started creating the block for the following transactions:
Node 9 is started creating the block for the following transactions:
Block creation time : 0.6437950134277344
size of blockchain : 9593
Node 2  Checking if some other node has already created the current block.
Block creation time : 1.0790231227874756
size of blockchain : 9593
Node 9  Checking if some other node has already created the current block.
# Node 2 Sending the block to all the nodes.
# This is new block message at Node = 3
# This is new block message at Node = 6
# This is new block message at Node = 8
# This is new block message at Node = 7
Already some other node won the race. (Node) 2
# This is new block message at Node = 4
# This is new block message at Node = 5
# This is new block message at Node = 1
# This is new block message at Node = 0
Node 2 recieved 8 from Node 8

Node 0 is sending 50 btc to Node 1 as a part of smart contract.
```

```
90
91  Node 2 is sending 79 btc to Node 1 .
92  Node 1 is sending 77 btc to Node 4 .
93  Node 6 is started creating the block for the following transactions:
94  Node 2 is started creating the block for the following transactions:
95  Block creation time : 0.19209694862365723
96  size of blockchain : 7823
97  Node 2  Checking if some other node has already created the current block.
98  Node 3 is started creating the block for the following transactions:
99  # Node 2 Sending the block to all the nodes.
100 # This is new block message at Node = 4
101 # This is new block message at Node = 5
102 # This is new block message at Node = 7
103 # This is new block message at Node = 9
104 # This is new block message at Node = 0
105 # This is new block message at Node = 1
106 # This is new block message at Node = 8
107 Block creation time : 1.2959938049316406
108 size of blockchain : 7839
109 Node 3  Checking if some other node has already created the current block.
110 Already some other node won the race. (Node) 2
111 Node 3 recieved 4986
112 Time taken by Node 3 for transaction verification is 11.129920959472656
113
114 Node 0 is sending 50 btc to Node 1 as a part of smart contract.
115
116 Node 2 recieved 4954
117 Time taken by Node 2 for transaction verification is 4.237336874008179
118 Time taken by Node 2 for transaction verification is 4.237358808517456
119 Block creation time : 4.242624998092651
120 size of blockchain : 4307
121 Node 6  Checking if some other node has already created the current block.
122 Already some other node won the race. (Node) 2
123 Node 6 recieved 4959
124 Time taken by Node 6 for transaction verification is 10.01668095588684
125 Node 4 is sending 28 btc to Node 1 .
126 Node 1 is sending 81 btc to Node 0 .
127 Node 7 is sending 48 btc to Node 5 .
128 Node 0 is started creating the block for the following transactions:
129 Block creation time : 0.7535960674285889
130 size of blockchain : 14871
131 Node 0  Checking if some other node has already created the current block.
132 Node 6 is started creating the block for the following transactions:
133 Node 9 is started creating the block for the following transactions:
134 Block creation time : 0.21936702728271484
135 size of blockchain : 11355
136 Node 6  Checking if some other node has already created the current block.
137 Block creation time : 0.2505218982696533
138 size of blockchain : 14887
139 Node 9  Checking if some other node has already created the current block.
140 # Node 0 Sending the block to all the nodes.
141 # This is new block message at Node = 5
142 # This is new block message at Node = 1
143 Already some other node won the race. (Node) 0
144 # This is new block message at Node = 2
145 # This is new block message at Node = 4
146 # This is new block message at Node = 3
147 # This is new block message at Node = 7
148 # This is new block message at Node = 8
149 Already some other node won the race. (Node) 0
150 Node 3 is sending 44 btc to Node 6 .
151 Node 8 is sending 25 btc to Node 1 .
152 Node 4 is sending 88 btc to Node 3 .
153
154 Node 0 is sending 50 btc to Node 1 as a part of smart contract.
155
156 Node 7 is started creating the block for the following transactions:
157 Block creation time : 0.6799516677856445
```

```
size of blockchain : 20165
Node 7  Checking if some other node has already created the current block.
Node 6 is started creating the block for the following transactions:
Block creation time : 0.3733649253845215
size of blockchain : 16641
Node 6  Checking if some other node has already created the current block.
# Node 7 Sending the block to all the nodes.
Already some other node won the race. (Node) 7
# This is new block message at Node = 5
# This is new block message at Node = 4
# This is new block message at Node = 8
# This is new block message at Node = 9
# This is new block message at Node = 1
# This is new block message at Node = 0
# This is new block message at Node = 2
# This is new block message at Node = 3
Node 7 recieved 12 from Node 3
Node 7 recieved 40 from Node 5
Node 8 is sending 4 btc to Node 1 .
Node 7 is sending 26 btc to Node 5 .

Node 0 is sending 50 btc to Node 1 as a part of smart contract.

Node 4 is halting due to timeout
Node 1 is halting due to timeout
Node 4 has bitcoins 5000 btc
Node 1 has bitcoins 5000 btc
Node 0 is halting due to timeout
Node 5 is halting due to timeout
Node 0 has bitcoins 5005 btc
Node 5 has bitcoins 5000 btc
Node 2 is halting due to timeout
Node 2 has bitcoins 5003 btc
Node 3 is started creating the block for the following transactions:
Node 9 is started creating the block for the following transactions:
Block creation time : 0.8495488166809082
size of blockchain : 25459
Node 9  Checking if some other node has already created the current block.
Block creation time : 0.945911169052124
size of blockchain : 25451
Node 3  Checking if some other node has already created the current block.
Node 6 is halting due to timeout
Node 7 is halting due to timeout
Node 7 has bitcoins 5052 btc
Node 6 has bitcoins 4959 btc
Node 8 is halting due to timeout
Node 8 has bitcoins 5000 btc
# Node 9 Sending the block to all the nodes.
Already some other node won the race. (Node) 9
Node 3 is halting due to timeout
Node 3 has bitcoins 4986 btc
Node 9 is halting due to timeout
Node 9 has bitcoins 5000 btc
```

Listing 10: **transaction**

6 Smart Contract

In this simulation we can do the Smart contract where a node will send the money to the other node as a part of smart contract. In this simulation it is possible to manage multi-smart contact and single smart contract. To use the smart contract feature we have to just toggle the value of **util.get_smart_contract()** to True and it will start performing the smart contract transactions.I have given the limit of the smart contract to 3000 but we can change in the function **util.get_smart_contract_balance()** to our convenience. During the mutli transaction we will get

the following output. I have set the smart contract between node 0 and node 1 but we can change it form **util.get_smart_contract_nodes()**

```
Node 0 is sending 50 btc to Node 1 as a part of smart contract.
Node 0 is sending 50 btc to Node 1 as a part of smart contract.
```

Listing 11: **smart contract**

7 Multi-Transactions

This simulation is able to perform the multi-transactions where one node will send coins to multiple nodes. To perform multi-transaction we have just toggle the function value **util.get_multi_transactions()** and it will start performing the multi-transactions.

```
Node 3 is sending 71 btc to Node 5 .
Node 3 is sending 8 btc to Node 7 .
Node 3 is sending 1 btc to Node 6 .
```

Listing 12: **multi-transaction**

8 References

1. Sarangi, S. R. (2020). Bitcoin and blockchains.
   Retrieved from *https* : *//www.cse.iitd.ac.in/* ∼ *srsarangi/courses/*2021/*col* _ 819 _ 2021/*docs/bitcoin.pptx*