Assignment 3 : Bitcoin Implementation

Aayush Kumar Singh(2020MCS2444) Akash Kumar(2020MCS2447)

May 14, 2021

1 Introduction to Bitcoin

Bitcoin[3] [5] is a distributed peer-to-peer cryptocurrency introduced by a still unknown person named Satoshi Nakamoto in 2009. It is probably first digital currency that solves the issue of double-spending by means of cryptographic mechanisms. Bitcoin is based on the cost function discussed in Hashcash paper[1]. There is no third party needed to mediate the transactions.

1.1 Transactions

Nodes (computers running bitcoin client) can send/transfer their unspent transaction to other nodes. Structure of simple transaction is similar to Figure 1
Transaction : P1 $\rightarrow$ P2 will contain following details
1. Hash of the previous transaction.
2. Public key of P2: This establishes the identity of the next element on the chain. processes in bitcoin are anonymous. They are only known by their public key.
3. Details of the transaction: amount of money, time, etc.
Hash of these element is calculated and is signed by sender's private key (here P1 in first Txn) to ensure the non-repudiation. And other nodes also can verify that that transaction is actually initiated by the sender only.

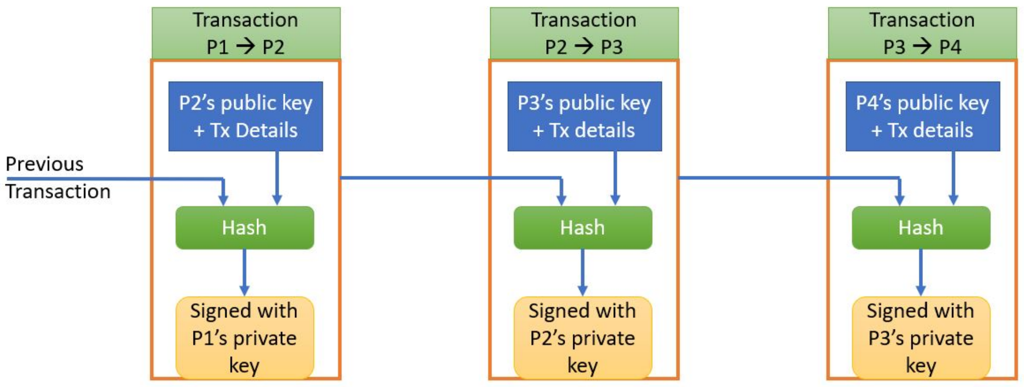

Transaction P1 → P2
P2's public key + Tx Details
Previous Transaction
Hash
Signed with P1's private key
Transaction P2 → P3
P3's public key + Tx details
Hash
Signed with P2's private key
Transaction P3 → P4
P4's public key + Tx details
Hash
Signed with P3's private key

Figure 1: Chain of Transactions

```java
class Transaction{
  public PublicKey sender;
  public int senderId;
  public byte[] txHash;
  public byte[] signature;
  public String timeStamp;
  public Vector<Input> inputTxns;
  public Vector<Output> outputTxns;
  public boolean coinbase;
  //Contructor for Normal Txns
  public Transaction(PublicKey sendersPublickey, int senderId){...}

  //Contructor for coinbase Txns
  public Transaction(boolean coinbase, PublicKey minersPublickey, int
   senderId){...}

  public void addTimeStamp(){...}
  public void addOutputToTxn(PublicKey receiver, double amt){...}

  public void addInputToTxn(byte[] refTxnHash, int indexInTheTxnsOutput){..}

  public byte[] getHash(){ return txHash;}
}
```

Listing 1: Transaction Class

To allow bitcoin value to be split and combined, transactions contain multiple inputs and outputs. Figure 3. This also reduces the number of transaction and reduce overhead of creating a transaction for each cent spent.

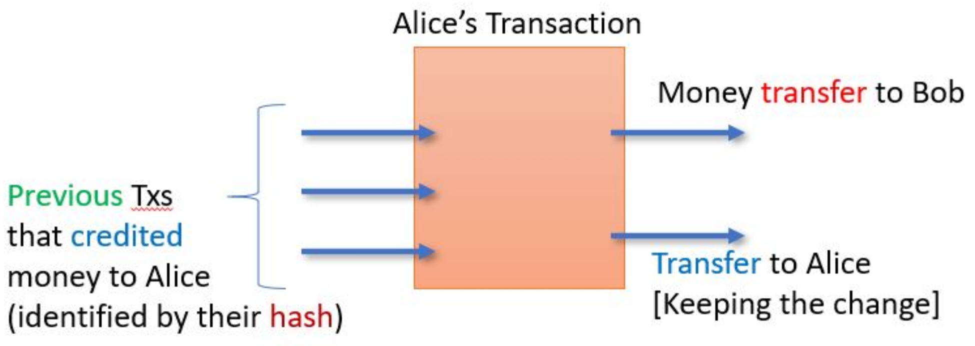

Alice's Transaction
Money transfer to Bob
Previous Txs that credited money to Alice (identified by their hash)
Transfer to Alice [Keeping the change]

Figure 2: Muti Input Multi output Transaction

1.2 Block and Blockchain

Multiple Transactions are grouped together to create a block. Transactions can be arranged linearly (as in Blocks of Figure 2) or using **Merkle tree** (Figure 3) within the block. Chain of these blocks is called blockchain(Figure 2). Here every block header have Block hash pointer of previous block. As chain grows and transactions are buried under enough blocks, transactions and internal hashes of Merkel tree can be discarded and only root hash can be kept to save the disk space.

First block in the blockchain is called **Genesis Block**.Its previous hash is set to all bits 0.

Incentives

First transaction in every block is **Coinbase Transaction** that create new bitcoins. And owner of these coins is the miner who mined the block.

```
class Block{

  //Block header
  public int minerId;
```

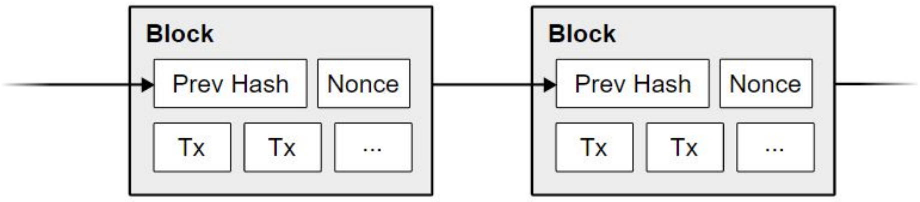


Figure 3: Blockchain

```java
    public byte[] blockHash; //hash(nonce||prev||merkelRoot)
    public int height;
    public byte[] prevBlockHash;
    public BigInteger nonce;
    public String timeStampC; //time of creation
    public String timeStampM; //time of mining
    public long timeTakentomine;  //in miliseconds.
    public Merkle merkleTree; //used to get merkle rootHash
    public Vector<Transaction> transactionsInBlock;

    public Block(Vector<Transaction> vec, int minerId){
        //This constructor is for genesis block only
        ...
    }

    public Block(byte[] prevHash, Vector<Transaction> vec, int minerId, int
     height){...}
    ...
}
```

Listing 2: Block Class

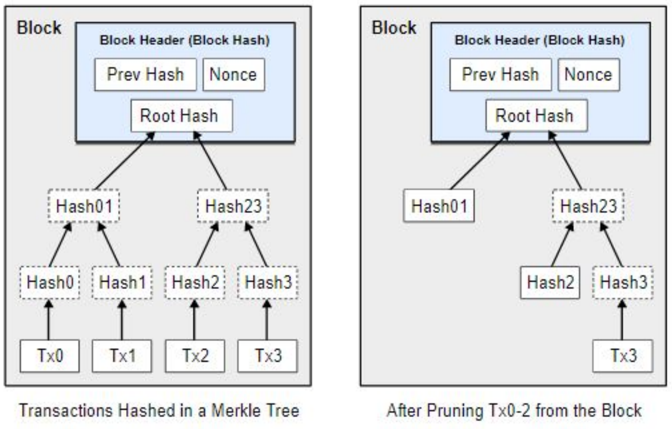


Figure 4: Blocks with Merkel Tree

1.3 Proof of Work

$$H(nonce||prevHash||Txn1||Txn2||Txn3...||Txn) =_n 0^k$$

or in case of merkel tree is being used for arranging transactions in block

$$H(nonce||prevHash||Merkelroot) =_n 0^k$$

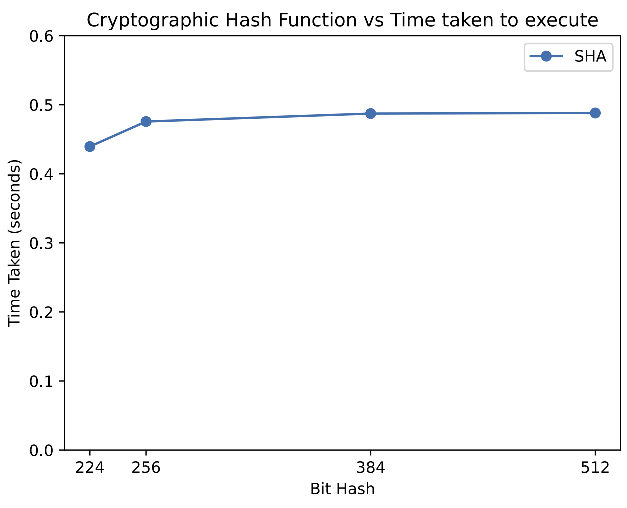
$H(...)$ is a hash function such as SHA256. Finding a *nonce* such that above calculated hash has n initial bits 0 is called **bitcoin mining**. As value of n is increasing computation energy needed to find required *nonce* increases exponentially.This is called **Proof of work**.
As every block contains(except Genesis one) hash of previous block. So if one tries to modify any block it's hash will change and this change will reflect in further blocks. So if one need to change a block he needs to redo all blocks after that also that requires significant amount of energy that is practically not possible for an adversary.

```java
public boolean proofOfWork(Block blk)
{ //while mining it keeps checking whether a block has arrived

    byte[] s = blk.getBlockInFormOfbytes(); //returns bytes of prevHash+Txns
   (merkel root)
    BigInteger nonce = new BigInteger("0");
    boolean didIminedIt = false;
    long start = System.currentTimeMillis();
    while(!Main.mined.get()){
        byte[] nonceBytes = getNonceByte(nonce);
        byte[] toBeHashed = concatTwoByteArray(nonceBytes, s);

        try{

        byte[] hash = Crypto.sha256(toBeHashed);
        if(isitRequiredhash(hash)){
          boolean ismined = Main.mined.compareAndSet(false, true);
          if(ismined){
              blk.nonce = nonce;
              blk.blockHash = hash;
              blk.addTimeStampM();
              didIminedIt = true;
              long end = System.currentTimeMillis();
              long milisec = (end - start);
              blk.timeTakentomine = milisec;
        }
          break;
        }
        nonce = nonce.add(ONE);
    }catch(Exception e){
        System.out.println("Exception");
    }
  }
  return didIminedIt;
}
```

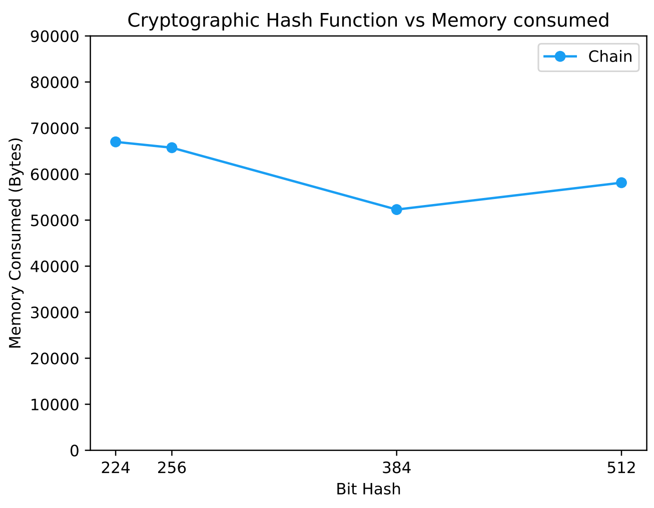
Listing 3: Proof of work method

2 Transactions

Transactions are created (in TransactionThread.java of each node) as objects in our program as shown in Listing 1. These transactions are then broadcasted to every node in the system including itself which then gets stored by them after validation in order to add them in the next block on block chain for which they will initiate mining.

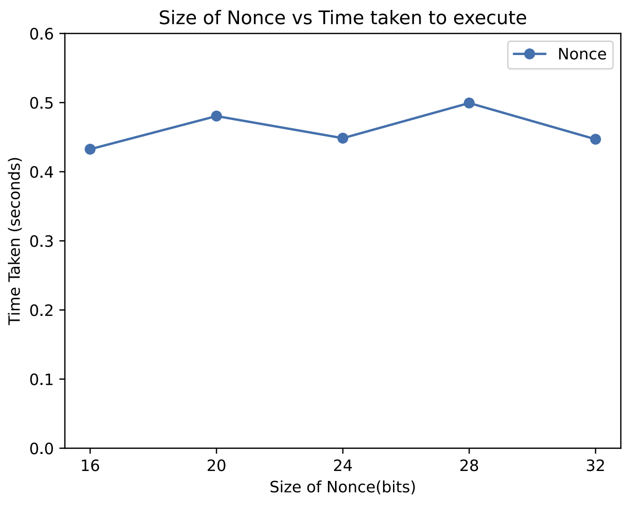


(a) Hash size Vs Times Taken

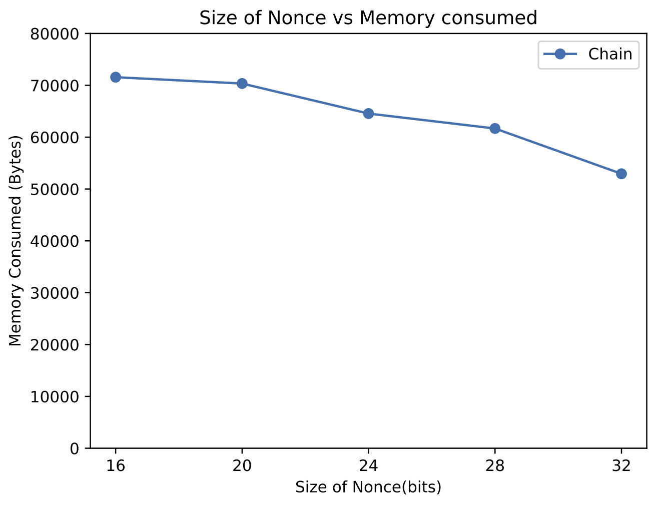


(b) Hash size vs memory required for storing blockchain

Figure 5: Change in time and space with Hash Size

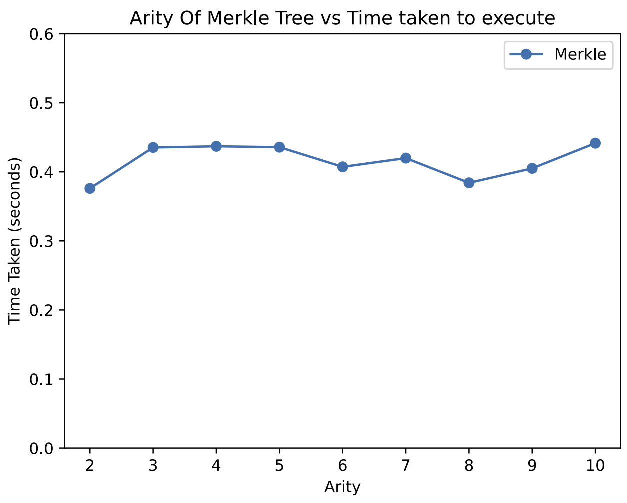


(a) Nonce Size (bits) vs Time(seconds)

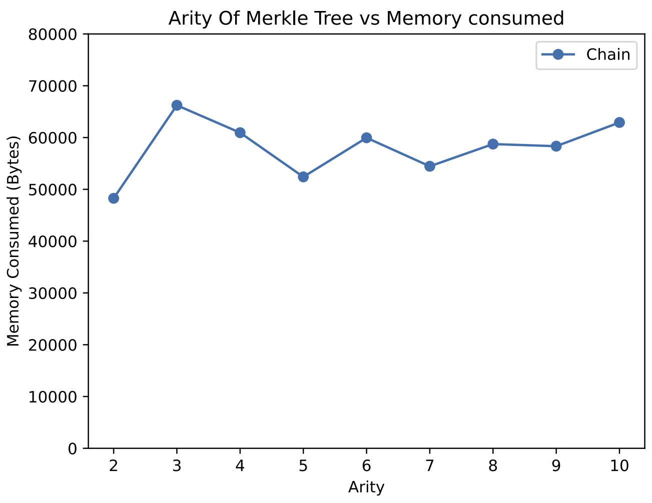


(b) Nonce Size(bits) vs Space(bytes)

Figure 6: Change in time and space with Nonce Size



(a) Arity vs Time(seconds)



(b) Arity vs Space(bytes)

Figure 7: Change in time and space with Arity of Merkle tree

2.1 Trends Observed:

1. Using longer size hash takes more memory but since in our implementation any node can make any random no of transactions that will be added to blockchain, so even though we are increasing the size of hash the time and space consumed doesn't show any identifiable relationship.
2. Changing the size of nonce has no effect on the Execution time as we can see from Fig 6 where slight variation in running time is due to randomness as stated in 1.
3. Similarly, In Figure 7 since we are not storing transactions in our merkle tree it doesn't have any significant effect on time and space consumed.

3 Security

The experiment of finding threshold was conducted by running it 3 times on the given number of blocks where each blocks can contains a max of 30 transactions and till each node had a chain of 25 blocks in case of 10, 50 nodes and 20 blocks in case of 100 nodes.

Below is the result of minimum no. of nodes for which our chain was forked for each number of nodes. Even if sometimes there was no forking after this threshold, those cases are not considered.

i. Number of Nodes: 10
It was ran with no. of malicious nodes equal to 1 and incremented by 1 till the total no of malicious nodes were equal to no. of nodes.

Run 1 (r1)	Run 2 (r2)	Run 3 (r3)	Avg. Threshold $\lceil r1 + r2 + r3/3 \rceil$
7	4	6	6

i. Number of Nodes: 50
It was ran with no. of malicious nodes equal to 5 and incremented by 2 till the total no of malicious nodes were equal to no. of nodes.

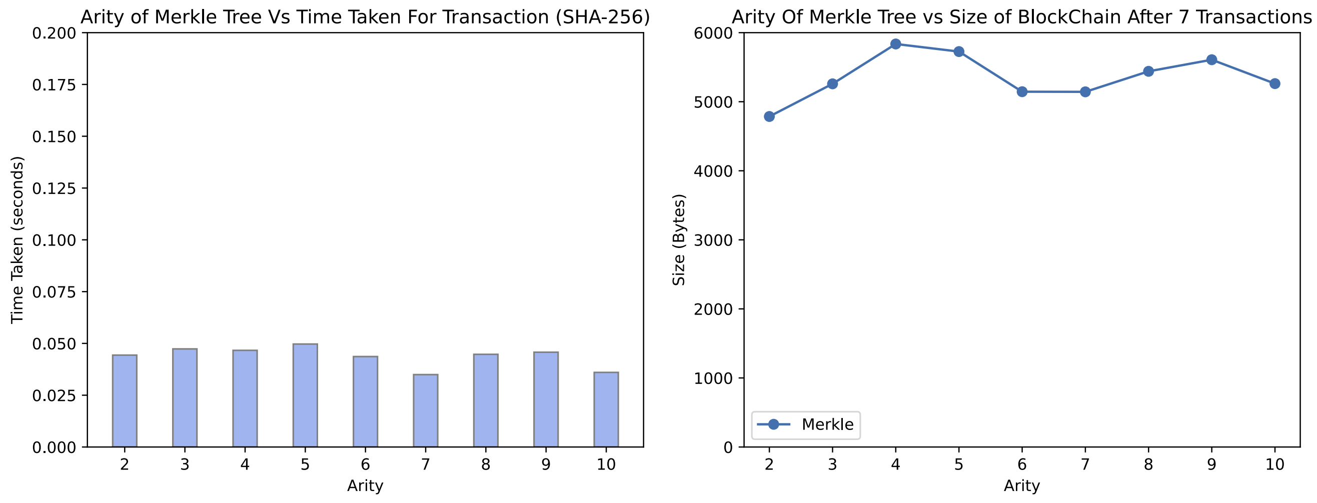
Run 1 (r1)	Run 2 (r2)	Run 3 (r3)	Avg. Threshold $\lceil r1 + r2 + r3/3 \rceil$
19	17	21	19

i. Number of Nodes: 100

It was ran with no. of malicious nodes equal to 5 and incremented by 5 till the total no of malicious nodes were equal to no. of nodes.

Run 1 (r1)	Run 2 (r2)	Run 3 (r3)	Avg. Threshold $\lceil r1 + r2 + r3/3 \rceil$
45	55	35	45

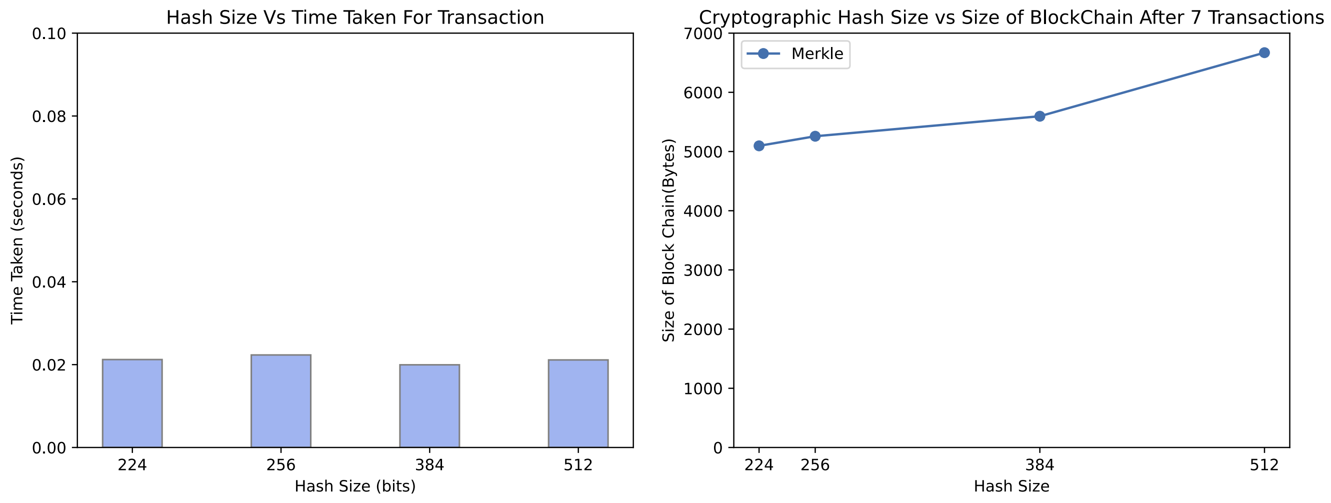
4 Graphs



(a) Merkel arity vs avg time taken to perform transaction

(b) Merkel arity vs Blockchain size

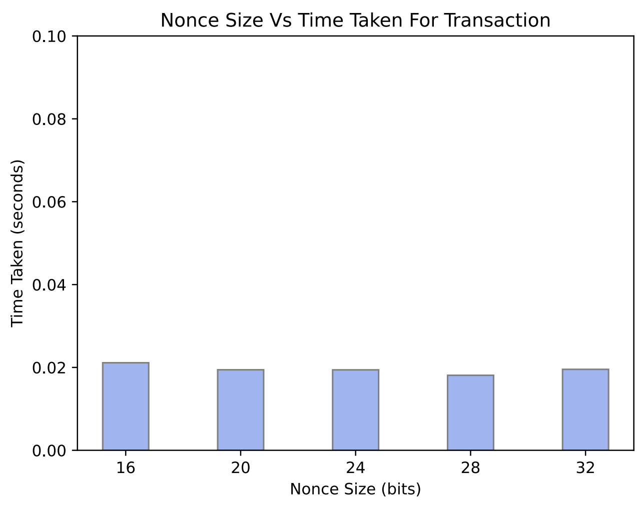
Figure 8: Change in time and space with Arity of Merkle tree



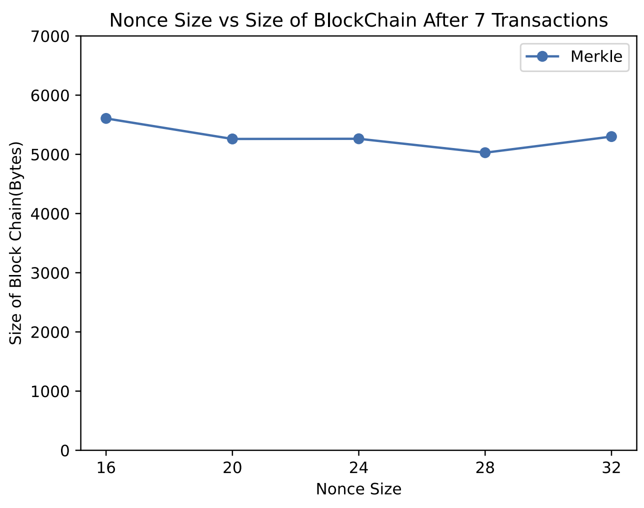
(a) HashSize vs avg Time to perform a txn

(b) Hash size vs Size of Blockchain

Figure 9: Hash Size vs time and size of blockchain

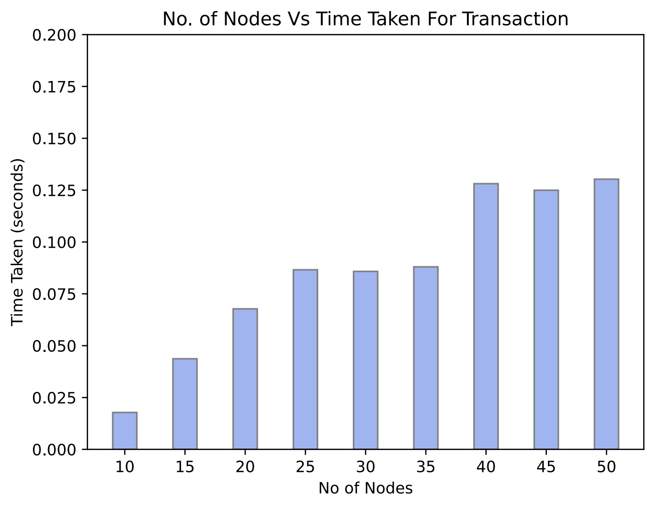


(a) Nonce Size vs avg time taken to perform a txn

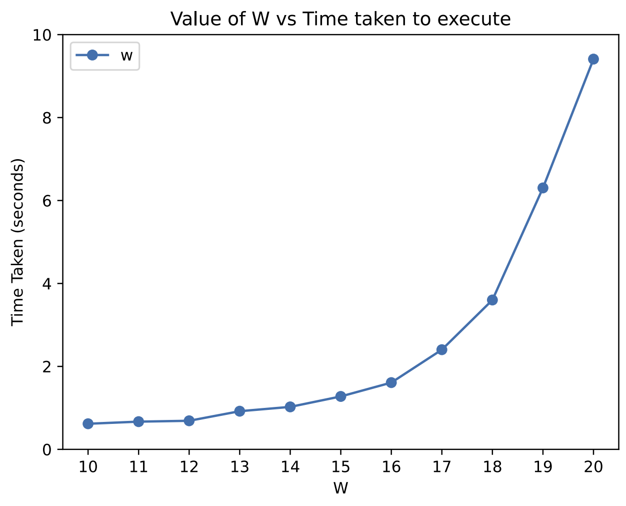


(b) Nonce Size vs Blockchain size

Figure 10: Nonce Size vs time and size of blockchain



(a) No of node vs avg Time to perform a transaction



(b) W (number of initial 0 bits in proof of work) vs avg time to find required hash

Figure 11

4.1 Trends Observed/ Justification

JUSTIFICATION For Figure 8(b), 9(b), 10 (b): Size of k = 7 was chosen as it was observed that at some point of time x no of blocks exactly contained 7 transactions on all runs. Since, any other number was more or less not present in some cases due to randomness of number of transactions in block, it was chosen to evaluate for our implementation.

A Transaction is said to be performed in our implementation when it has been created by the sender and the transaction is finally inserted in our blockchain.

1. Most reasonable trends that can be seen is when size of hash is increased size of blockchain increases.
2. Figure 6 and Figure 10: Size of nonce have no significant effect on size of block chain as we have declared nonce as BigInteger type. if nonce value increases then to prevent overflow one more byte of size is increased when doesn't fit in allocated bytes. So due to nonce change

in block size will be only by few bytes. And it will be almost random because at which nonce required hash will be obtained if mostly depends on block content.
3. In Figure 11 (a), we see as no. of nodes in the network increases there is an increase in time to perform transaction as in our implementation we are choosing 0.7*times the total no of nodes to be the receiver of a transaction. So, having more no. of nodes our implementation will take more time to choose from these nodes. Similarly, on other side the time to process this transaction by other nodes will increase.
4. Figure 11(b): The proof-of-work is obtained by incrementing a nonce until a value is found that gives the block's hash with the required **W** (or n in above Pow formula in section 1.3) zero bits. Time to find such hash increasing exponentially. And the same trend can be seen in Figure 11(b).

5 More Implementation details

```java
public class Main{

    public static final double MINING_REWARD = 50;  //Reward amount
    public static int w = Parameters.W;  //first w bits to be zero
    public static volatile int nonceSize = Parameters.NONCESIZE;
    public static int arity = Parameters.ARITY; //default
    public static int maxTransactionInBlock = 10;
    public static int stop = 20; //least length of blockchain when to stop

    static boolean printOnConsole = false;  //if want logs to print on
   console set it to true.

    public static int numNodes;
    public static Vector<Node> nodes;
    public static double oneSatoshi = 0.005; //min amount of bitcoin that
   can be transferred

    public static double probToSend = 0.2;
    static Vector<Queue<Message>> messagePassingQ;
    static Logger logger;
    static Logger txnLogger;
    static String path = "logfiles";

    static Vector<Transaction>invalidTxns ;
    static HashMap<PublicKey, Integer> publicKeymappings;
    static Vector<HashMap<PublicKey, Vector<UnspentTxn>> > unspentPools;
    static Object printingMutex = new Object();
    static Object unspLock = new Object();
    public static AtomicBoolean mined = new AtomicBoolean(false);
    ...
}
```

Listing 4: Main Class (variables and data structures)

```java
public class Node implements Runnable{
    public final boolean DEBUG = false;
    static final BigInteger ONE = new BigInteger("1");
    private final PrivateKey privateKey;
    private final PublicKey publicKey;

    public Vector<Block> bitcoinChain;
    public Thread tId;
    public TransactionThread txnThreadNode ;

    public int nodeId;
    public HashMap<PublicKey, Vector<UnspentTxn>> unspentTxns;

    public Vector<Transaction> validTransactions;
    public Set<byte[]> transactionsInBlockChain;

    private boolean reqWasSent = false;
    private boolean minedFirst = true;
    private boolean isHonest;
    private int startOfDishonests;
    ...
}
```

Listing 5: Node Class(variables and data structures)

For this assignment we have also referred [2] [4]

6 How to Run code

For this please check the README file in the Assignment3_2020MCS2444_2020MCS2447 folder.

Output of runs are stored in Screenshots folder.

References

[1] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[2] Blockchain.com. Bitcoin explorer Website. https://www.blockchain.com/explorer?view=btc.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[4] Arvind Narayanan, Joseph Bonneau, Edward Felten, et al. *Bitcoin and cryptocurrency technologies: a comprehensive introduction.* Princeton University Press, 2016.

[5] Prof. Smruti R Sarangi. Lecture slides and video on bitcoin, 2021.