Advanced Distributed Systems - Assignment 3(Bitcoin)

Pratik Karia - 2019MCS2568, Vaibhav Kiran Kurhe - 2019MCS2572

August 2020

1 Introduction to Bitcoin[1]

- Bitcoin is a P2P cryptocurrency system that addresses the problem of trust significantly.
- The Bitcoin system has it's core concepts strongly related to Cryptography that provides the security that a currency system needs.
- In addition to that, to build trust, the Bitcoin system using a consensus protocol that almost guarantees trust for people to invest their money in it.
- Along with being a component of currency system, Bitcoin also includes implementation of something known as Smart Contracts.
- Smart Contracts is similar to a normal contract except for the fact that all the terms and conditions are implemented in code and the agreements in the contract exist across a distributed, decentralized blockchain network.
- In this assignment, we have implemented a complete Bitcoin system with Multi-transaction support. Also we have implemented a Smart Contract that will enable a voting process to be held in the bitcoin network.
- This voting process can be used in numerous ways, starting for voting in minor elections and their votes to a potential of undergoing major elections that need a complete anonymity for the voters.

2 Implementation

2.1 Basic Idea Of Implementation

2.1.1 Bitcoin Network

- The basic idea of our implementation is to create multiple processes and perform synchronization between them to simulate a real-life bitcoin network.
- Each process is treated as a Full Node in a bitcoin network.
- We have kept lot of fields as hyper parameters so that we can change their value by doing minimal change in our codebase. Some of these fields are arity of Merkle Tree, number of nodes, number of transactions, smart contract fee and many more.
- Each node generates a random transaction for itself and broadcasts it to all the nodes in the network.
- All the nodes collect the transactions, create the block and then start solving the mathematical puzzle which is the proof of work in which the node has to change the value of nonce so that it is less than a target. This concept in a real-life bitcoin network gives power to the nodes with very high computation power.
- To simulate the difference in computation power of nodes in real life, we decided to create a randomly generated target for each of our process nodes. This randomly generated target will have a random number of leading zeros which will provide a simulation of difference in computation power.
- In addition to that, we also have a system that generates transactions at a node level. By this, we mean that each node generates random transactions and then broadcasts this. By using this and above implementation details, we are able to simulate the bitcoin network.

2.1.2 Smart Contracts

- In Smart Contracts, we built a Smart Contract to perform Electronic Voting.
- Most of people, when it comes to Smart Contracts in Bitcoin, will associate it with some financial matter, but we thought about the basic underlying principles that Bitcoin follows and thought of applying it to something like voting.
- The most important thing that is ensured in this voting smart contract is anonymity, correctly providing the voting rights, prevention of duplicate votes and automatic vote counting.
- Initially there will be one **Initiator** that initiates the voting process and all others act as **Voters**. Some nodes act as voters every time a voting is done while some act as voters only when they are randomly chosen.
- So the idea for this was that we consider each vote as a contract between a voter and initiator.
- In this contract, the Initiator provides the voter it's voting rights and the voter provides the initiator with it's vote.
- To maintain the essence of bitcoin system, each Voter sends the Initiator a voting fee that can be customized.
- Finally, after every vote is cast, the Initiator counts the votes and declares the result of the election.

2.2 List of Code Files

1. **utils.py -** Consists of code for generating public and private keys, creating signature, verifying signatures, creating hashes and verifying Merkle Tree.
2. **constants.py -** Consists of various hyper parameters like coinbase amount, hash size, arity, lockTime, voting fee, candidates for voting in smart contract, number of transactions and number of nodes.
3. **MerkleTree.py -** Consists of code for creation of Merkle Tree
4. **ScriptEngine.py -** Consists of code for implementing the stack based bitcoin scripting language which is used for signing and verifying transactions.
5. **Transaction.py -** Consists of 3 classes - TransactionInput, TransactionOutput and Transaction. This code basically creates list of transaction outputs, takes a transaction input and combines it to form a transaction. Additionally it also creates the **Script Signature**.
6. **Block.py -** Consists of a class to store a block which contains block header, list of transactions and merkle tree.
7. **Blockchain.py -** Consists the class blockchain which verifies the incoming blocks and stores it in it's attributes provided it is verified successfully.
8. **BitCoinNode.py -** The code for implementing the crux of Bitcoin which is the consensus and proof of work. The class BitCoinNode represents a bitcoin node in a real life bitcoin P2P network. It contains various data structures to store it's unspent output, it's blockchain, it's previous transaction hashes, any incoming transaction and any incoming blocks. In addition it also performs the proof of work using a randomly generated target as mentioned above.
9. **SmartContractNode.py -** Implements the smart contract node which can be either an Initiator or a Voter. It contains the code for the initiator the voting contract to the voters, the voters performing a transaction = votingfee to the initiator and casting a vote. Finally the initiator counts the votes and declares the results.
10. **Main.py -** Main function that creates the processes and simulates bitcoin network by setting number of transactions and number of nodes in the constants.py file.
11. **SmartVoting -** Contains code to take the nodes who will always be part of smart contract and randomly selecting few nodes out of other, spawing the processes of initiator and voter and using SmartContractNode.py to perform the Smart Contracts.

3 Experiments

The experiments are conducted to compare block size (total memory needed to store the transactions including hash size, transactions size, merkle tree size in the blockchain) and the transaction time (the time it takes for a transaction to settle into the blockchain) with varying values of the different hyper-parameters.

Following parameters are considered:

- Arity of the Merkle Tree
- Hash Size
- No of nodes
- Nonce size

Figure 1 shows the experimental results for varying transaction time with respect to the arity of the merkle tree. For this experiment, nonce size is 16, no of nodes are 20, hash size is 256 and no of transactions carried out are 10.

As the arity is increased, more and more no of transactions are merged to form a node of merkle tree. This reduces the tree height and hence reducing the no of nodes in the tree. Less no of nodes means less computation and hash overhead involved. Thus for a fixed no of transactions, the transaction time decreased with the arity.

Figure 2 shows the varying block size with different no of transactions w.r.t the value of arity. The fixed parameters for this experiment are nonce - 32, no of nodes - 50 and hash size 256.

The value of nonce is generated randomly in the range (0, pow(2,sizeofnonce)). Every block stores one nonce value and therefore, we see some minor change in block size with the increase in nonce size.

Figure 3 shows the varying block size having different no of transactions w.r.t the size of hash. Fixed parameters are : nonce - 32, no of nodes - 50 and arity - 4.

As the hash size increases, space complexity to store a block increases, as the block stores the hash of every transaction. Higher block size means more memory needed to store the block.

Figure 4: as every block stores one nonce value and thus we see a minor change in block size with the increase in nonce value.

Figure 5 depicts that the on increase of nonce size the time taken to perform transactions also increase as now it gets difficult to fulfil the target requirements.

Figure 6 depicts the smart contract execution time with respect to the number of nodes. As we can see that as the number of nodes increases, the overall time increases as more nodes pay the voting fee to the initiator and hence more number of transactions are broadcasted and processed in blockchain.

4 Dishonest Nodes

Dishonest nodes can cause harm to the Bitcoin System. There are few attacks which are possible if there are a particular number of dishonest nodes in the Bitcoin System:

- Rejecting blocks from a particular user and being biased
- **Double Spend -** An attacker may attempt to spend an already spend money.

But in spite of existence of such dangerous attacks, Bitcoin System and it's underlying cryptography provide a good amount of security from these attacks provided the number of dishonest nodes are below a threshold. This threshold is as follows :

Number of Nodes	Minimum Number of dishonest nodes to harm the system
10	6
50	26
100	51

5 Requirements

- Python 3.6
- Crypto Library (For RSA)

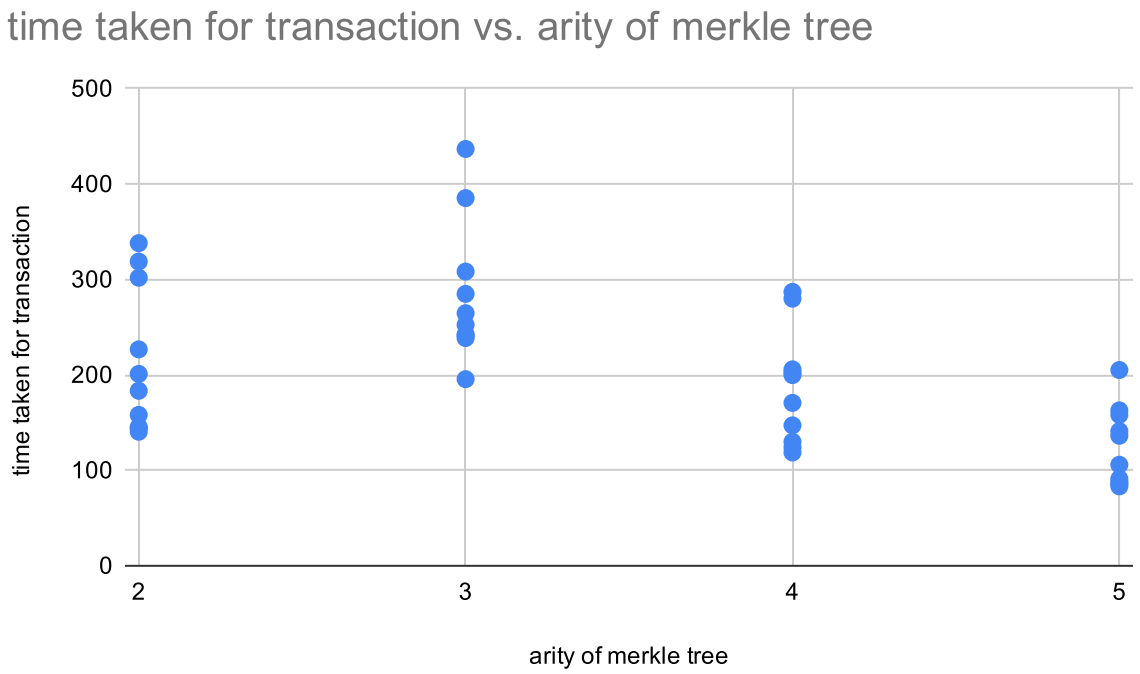

Figure 1: Arity v/s Transaction Time

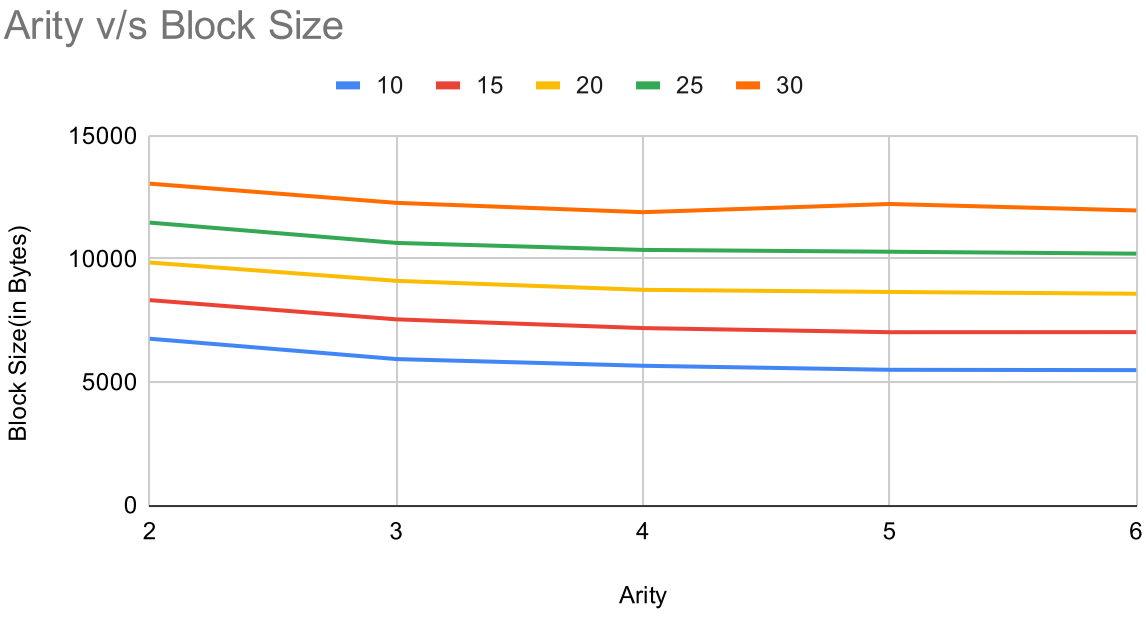

Figure 2: Arity v/s Block size

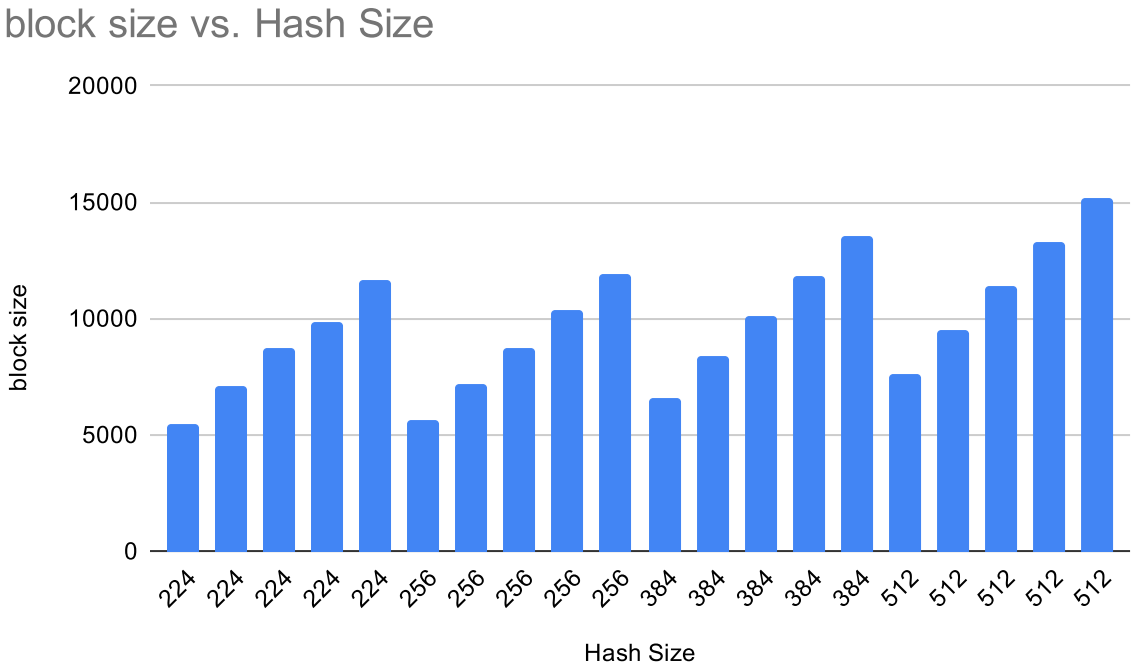

Figure 3: Block size vs hash size

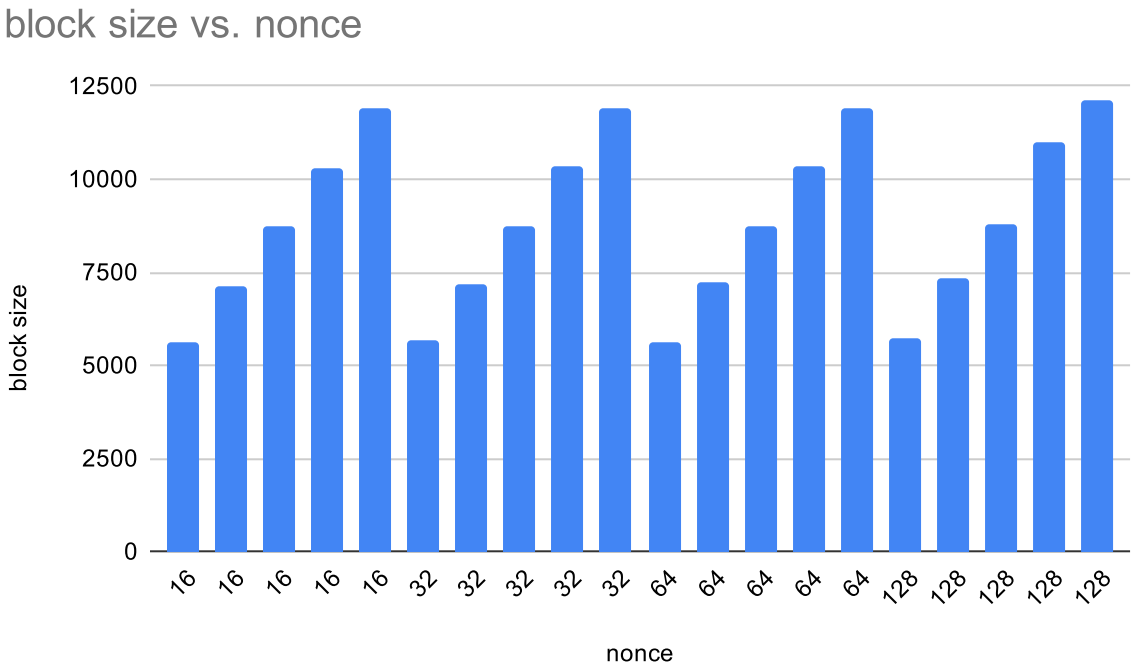

Figure 4: Block size vs Nonce

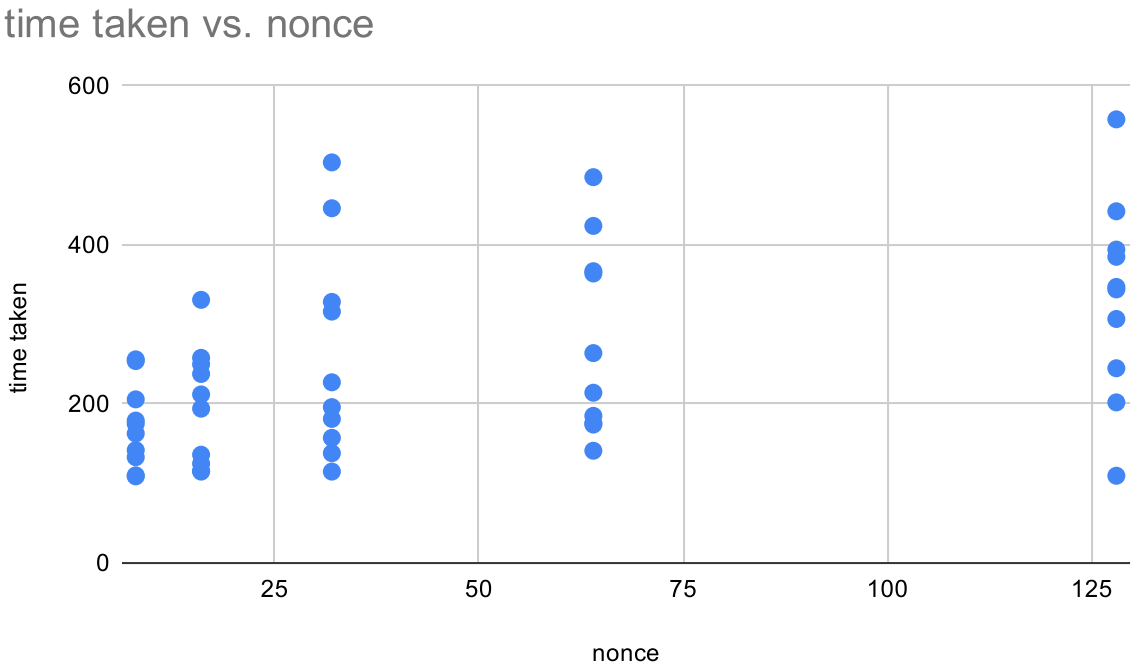

Figure 5: Nonce size vs time

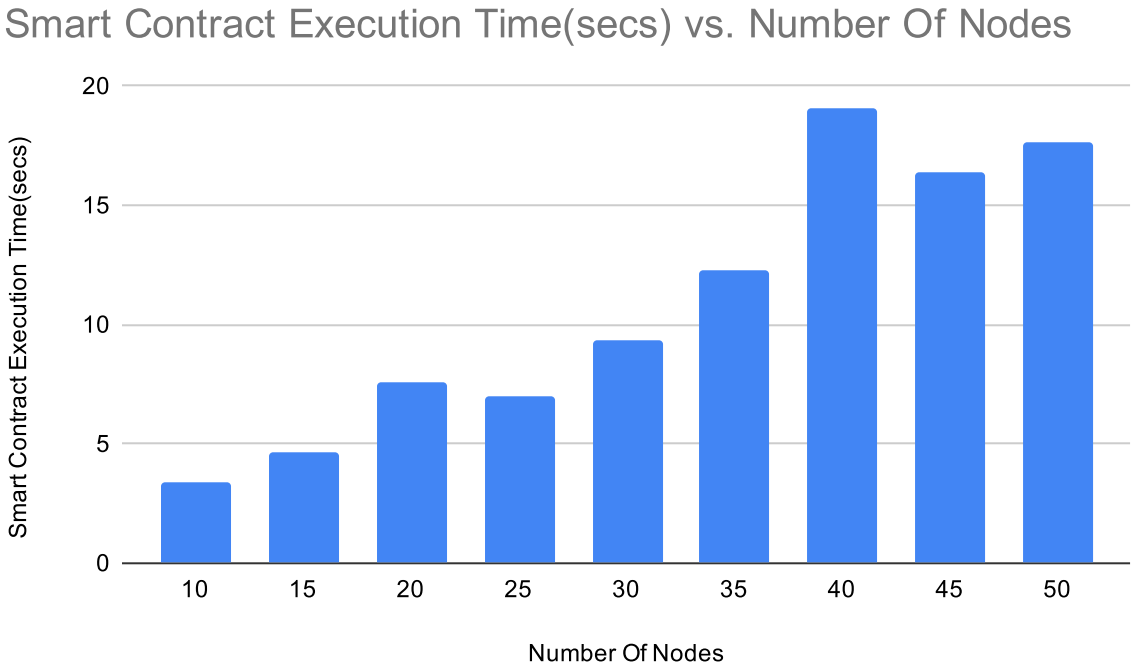

Figure 6: Smart Contract Execution Time vs No of nodes

- Random Library
- Os library
- Typing Library (To enable static type checking in python)

6 How to run

6.1 BitCoin System

- Set the appropriate hyperparameters in constants.py
- Run the bitcoin system using **python Main.py > log** - This redirects output to a log file. Also it runs the code infinitely thus simulating a Bitcoin Network so you can terminate it manually.

6.2 Smart Contract

- Set the appropriate hyperparameters in constants.py
- Run the simulation of Smart Contract using **python SmartVoting.py**. This will run the Smart Contract Voting system and publish the results once voting is complete.

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.