COL819: Programming Assignment 3
Implement Bitcoin

Logistics

1. Due date:
2. Maximum marks: 100
3. **To be done in a group of two.**
4. Submissions on Moodle: Code and report has to be submitted.
5. Languages: Java/C++/Scala/Google Go/Python (We recommend: Java)
6. **Report must be in Latex and use vector graphics for images (graphs, if any).**

1 Poblem Statement

In this assignment, working in a group of two, you have to implement a complete Bitcoin system with multi-transaction support. Broadly, you need to handle the transactions, integrity checks, digital signatures, proof-of-work, and consensus requirements. After the implementation, you need to test it out by carrying out some transactions. Furtermore, you need to evaluate your the security of your Bitcoin implementation by adding dishonest nodes into the system. Your job is to keep adding dishonest nodes till a *fork* attack is successful. Finally, you are required to implement *smart-contracts* in the your Bitcoin implementation.

2 Bitcoin [20 Marks]

In this part of the assignment, you need to implement a complete Bitcoin system. Few things to keep in mind:

1. Nodes must be implemented as separate processes or threads (same as GHS).
2. The proof-of-work can be made simple as per your system's capabilities.

3. You need to handle the creation of the *genesis* block.
4. You need to ensure the integrity of the blocks using Merkle trees.
5. You have to assign a $< public, private >$key pair to every node.
6. You need to ensure that Nakamoto consensus [4] holds.
7. Unless otherwise specified, assume your nodes are fault-free.

3 Transactions [20 Marks]

After successful implementation of Bitcoin, create a network of 10 nodes and perform some transactions using the Bitcoins.

1. Print the initial state of the each node, i.e. after the genesis block creation.
2. Print the transactions performed.
3. Nodes are given a fixed number of Bitcoins as an incentive for adding a block.
4. Print the final state of the node, after all the transactions are done.
5. Print a log of the transactions executed.

Also, please design a test case to show multi-transaction support.

Apart from these, you are required to vary the value of the hyper-parameters of your blockchain, such as the hash size, properties of the Merkle tree like the arity, size of the *nonce*, and report the time and space implications because of these changes using a series of plots. Comment on the trends observed.

This should be scalable for up to 100 nodes. You will be asked to run this during grading.

4 Security: Dishonest nodes [20 Marks]

A dishonest node attempts to fork a chain with malicious intent. However, the consensus protocol of the Bitcoin provides protection against such attacks, till the number of honest nodes in the system is greater than a particular *threshold*. Your task is to find that threshold, i,e, the number of honest nodes required to maintain the security of the Bitcoing, in a 10, 50 and 100 node system.

5 Smart-Contracts [10 Marks]

In this part of the assignment you need to implement a simple form of smart-contracts. A smart contract in a Bitcoin represents an automatic task that gets executed when some pre-defined condition is fulfilled. A typical example is a smart-contract to remove X amount from an account A, as soon as the balance in the account A is more than Y, where $Y > X$.

You need to implement two versions of this smart contract

1. *Execute Once*: In this case the smart contract is executed exactly once.
2. *Repeat*: In this case the smart contract is repeated in-definitely.

Create a network of 10 nodes, and put a *Execute once*, and *Repeat* contract on two different nodes. Generate some test transactions such that these contracts are executed (Repeat contract at least twice).

1. Print the initial state of the nodes.
2. Print the final state of the nodes.
3. Print the transactions carried out, with smart contracts transactions - formatted to stand out.

6 Report [30 Marks]

You need to create a maximum of 10 a4 single spaced page report with references. The report should contain all the details related to the implementation along with the results generated, and their explanations. Every single parameter has to be justified either from first principles or from prior work.

You need to run each experiment until it reaches a steady state. This needs to be visually shown for a few cases.

Finally, be creative. Add as many new experiments as you can with adequate justifications.

The report should contain details of the different design choices that you made during the implementation, such as how to implement transactions at a high level, how to do you ensure proof-of-work, etc.

It should also contain instructions on how to run the code.

6.1 Graphs

1. The effect of changing the arity of the Merkle tree. Plot a box-plot, with the arity on the x-axis, and the time to perform a transaction on the y-axis (perform 10 transactions).
2. Use a simple plot (line or scatter) to show the change in the size in bytes (x-axis) of the blockchain after *k* transactions for different artieis of the Merkle tree (x-axis). Experimentally determine a value for k. Justify it.
3. Repeat the above two experiments to plot the change in a transaction's mean creation time (y-axis) upon changing the hash size (x-axis).
4. Repeat the above two experiments to observe the change in a transaction's mean creation time (y-axis) upon changing the *nonce* size (x-axis).
5. Pick a value for the Merkle tree arity, the hash size and the *nonce* size based on previous experiments. Using them plot the time taken by a transaction with an increase in the number of nodes in the Bitcoin system. Use a box-plot with transaction time on the y-axis and the number of nodes in the x-axis .

Quantify the time it takes to execute smart contracts.

General guidance

- Please stick to basic packages during implementation.
- If you are not sure if a particular package is allowed, ask on Piazza.
- Grading will be done based on the correctness of the code and the report quality. There might be a demo, if required. So please ensure that the submitted code executed on your machine correctly.
- We will run **MOSS** on the submissions. Anyone found with copied code either from the Internet or from another student, will be dealt with as per the class policy.

References

[1] Majority attack - bitcoin wiki. `https://en.bitcoin.it/wiki/Majority_attack`. (Accessed on 04/01/2020).

[2] Merkle tree - wikipedia. `https://en.wikipedia.org/wiki/Merkle_tree`. (Accessed on 04/01/2020).

[3] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[4] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.