CS620 Assignment 2

Due date: October **10**, 2019

Objective: Build a blockchain on top of the P2P network developed in assignment 1.
Overview:
The Blockchain should start with a genesis block whose hash is "0x9e1c". Each hash is 16 bit in length and hence can be represented in hexadecimal with four letters. Every block consist of following fields:

- Block Header
  - Previous block hash (16 bits)
  - Merkel root (16 bits)
  - Timestamp - Unix timestamps, which is the number of seconds that have elapsed since January 1, 1970 at 00:00:00 GMT.
- Block Body : For now we leave the Body empty (with no transactions). So the block is just the block header. The block hash which you put in a block is the last 16 bits of the SHA-3 hash of the previous block. The Merkel root is any arbitrary value for this assignment.

Block Mining
We will simulate PoW mining in the following way. When a miner receives a block at say time t, it will generate an exponential random variable exponential random variable, say τ. The miner will wait up to time $t + \tau$. If it has not yet received another block by then, it will generate a block and broadcast it. If on the other hand, it receives a block (which creates a longer chain than its current chain) before $t + \tau$, it will generate a new exponential random variable and the process repeats.

If $interarrivaltime$ represents the average block interarrival in the entire network, then you can use the following steps to generate the exponential random variable described above.

- Overall block generation rate
  $globalLambda = 1.0/interarrivaltime$
- Scale individual miners lambda parameter based on the percentage of hash power it has.
  $lambda = nodeHashPower * globalLambda/100.0$
- Appropriately scale the the exponential waiting time
  $waitingTime := time.Duration(rand.ExpFloat64()/lambda)$

For simplicity, we will eliminate the concept of node syncing where when a miner joins a blockchain network, it first downloads the current blockchain and then starts mining. Instead, in this assignment, all miners will first join the P2P network and then all simultaneously will start mining.

There will be one designated node which will pass the command "start mine" to all the nodes in the network, on receipt of which they will start mining.

While a node is mining two events can occur a) The node itself comes up with the solution, in our case, the timer expires, or b) The node receives the block from another node in the network and this block creates a longer chain than it had before receiving the block.
a)After receiving a block, it should perform the following steps :

- Validate the received block (check if the hash in the block is the hash of some other exisiting block; timestamp was generated within 1 hour (plus or minus) of current time.) Reject the block if it fails the validation test, else do the following.
- Store it to the database (you are free to use any database which you are comfortable with).
- Broadcast the block to neighbors in the P2P network.
- If the block creates a chain longer than the longest chain currently, reset the timer for next block waiting time.

Note that node should always mine on the **longest chain** available locally.

b)After node creates the block, it should follow the following steps :

- Store the block in the database.
- Broadcast the block to neighbors.

Selfish mining attack : majority is not enough
The key idea behind this strategy is for a selfish miner (Adversary) to keep its discovered blocks private, thereby intentionally forking the chain. The honest nodes continue to mine on the public chain, while the adversary mines on its own private branch. If the adversary discovers more blocks, it develops a longer lead on the public chain, and continues to keep these new blocks private. When the public branch approaches the adversary's private branch in length, it reveal blocks from its private chain to the public. Details are in the paper "Majority is not Enough" by Gun Sirer et al. You can implement this attack by making a node behave as an adversary and test the severity of attack for different mining power assigned to it.

Definition 1. Mining power utilization : *Mining power utilization is the ratio of the number of blocks in the longest chain to the total number of blocks in the blockchain (including forks).*

Program Output
You should submit a report along with your code on moodle (submit a zip file). The report must contain the following plots :

- Mining power utilization(defined in Definition 1) Vs block generation rate.
- Attacker mining power Vs fraction of blocks mined by it that appears in the longest chain.
- A para clearly explaining your criterion determining the longest chain from the blocks stored in your database.

Run the experiment for different starting random seed values and plot the graph showing 90% confidence interval. All the plots should be the result of a minimum of 10 minutes run of the experiment. Please note that length of the longest chain, as well as entire blockchain that individual node see will be different, so you have to take the average over all the nodes while plotting the graph.

While giving your demo, you will have to show using any graphics tool the blockchain tree at any node. You are free to use any graphics tool to draw the tree. We have mentioned few potential graphics tools in the links below.

Useful Links You can follow the links below to brush up your knowledge about Selfish mining attack and some graph plotting tools.

1. Selfish mining attack : Majority is not Enough.
2. Numpy.
3. Latex graph

Submission Instructions:

1. The assignment should be done in a group consisting of a maximum of two students.
2. The code should follow programming etiquette with appropriate comments.
3. Add a **README** file which includes a description of the code and gives detailed steps to compile and run the code.
4. Zip all your code and the report as a single file with the name **rollno1-rollno2.tar.gz** and upload it to Moodle. Only one group member should upload the file.
5. You should be able to **demonstrate** the code on a minimum of three machines.