CS765 Project Part-II

Team Size : At Most 3 students per team

Due date: November **8**, 2020

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. Note that τ depends on the hash power assigned to the miner. 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

  $waitingTime := time.Duration(rand.ExpFloat64()/lambda)$

A newly joined node, say N, should sync with the blockchain's current height in the network. N will receive the recent block B_k where k is the height, from the network immediately after joining the network. N won't push the B_k on to the chain; it will be in the pending queue. N will request blocks from B_0(genesis block) to B_{k-1} from the network and put each on the chain, post validation. All the block received after B_k will enter to the pending queue. After the node N sync till $k-1$, it starts processing the pending queue. For simplicity, the limit on the size of the pending queue is unrestricted. An honest node will begin creating and mining the block once the pending queue is empty.

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 node creates the block, it should follow the following steps :

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

b)After receiving a block, the node should insert the block into the pending queue and abort the mining process if it's running.

Node validates the blocks in the pending queue one at a time. Validation consists of 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 and pending queue is empty, reset the timer for next block waiting time.

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

Block flooding attack :
An adversary can flood the network with invalid blocks(Do not mine) to keep other node's pending queue full and hence suspend their block creation process. Note that an honest node won't forward the invalid blocks; thus, an adversary should explicitly flood a particular set of nodes with the sequence of blocks. Note that an invalid block won't go into the main chain. An adversary also parallelly mines and sends the valid blocks that follow the same process as the honest miner. You can implement this attack by making a node behave as an adversary and test the severity of attack for different inter-arrival times(starting from 2 sec), keeping mining power assigned to the adversary and % of nodes flooded in the network constant. Assume that the node behaving as an adversary consume 33% of the total hashPower. Severity is defined in terms of fractions of blocks mined by an adversary that ends up in the main chain with and without attack. You also have to observe the effect of the mentioned attack strategy on the mining power utilization(defined below).

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) Vs inter-arrival times.
- A Fraction of main chain blocks mined by the adversary Vs inter-arrival times
- A para clearly explaining your criterion determining the longest chain from the blocks stored in your database.

Run the experiment for 10%, 20%, and 30% of node flooded in the network and draw the above plots for each. All the plots should result from a minimum of 10 minutes run of the experiment(which we will verify from the timestamp maintained in the log file). Please note that the length of the longest chain and the entire blockchain that individual node see will be different, so you have to take the average over all the nodes while plotting the graph the mining power utilization.

While giving your demo, you will have to show the blockchain tree at any node using any graphics tool. You are free to use any graphics tool to draw the tree. We have mentioned a few potential graphics tools in the links below. Also, on the program's exit, you should be able to pop out the listed plots for one value of % of the node flooded.

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

1. Numpy.
2. Latex graph

Submission Instructions:

1. The assignment should be done in a group consisting of a maximum of three 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-rollno23.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.