Random Number Protocol and Shuffling Miners

1 Random Number Protocol

Rules for the random number protocol

1. Each MxN miner generates a cryptographic-quality random number.
2. Miners broadcast the signed hash of their random numbers.
3. In the next cycle, the miners broadcast their random number to the other miners once they have received each other's hashes.
4. The miners verify that the other miner's hashes and random numbers match.
5. If yes, all of the random numbers are concatenated in order of the miner's IDs and then hashed.
6. This random number is used to map to the MXN set so that atleast one miner is dropped from the active set and replacd from the bench W miner set.
7. TBD : If the miner does not reveal the random number what happens? Punishing the miner. Can be done with rate limit and miner reputation.
8. TBD : How will the bench miners have access to the old blocks? Like once a bench miner comes to the active mining set. Syncing protocol using bloom filter.

2 Shuffling Miners

Rules for shuffling miners

1. Assume *rand* is the random number from *step* 5 in random number protocol.
2. The first miner is assigned to position *rand* mod N
3. The second miner is assigned to position hash(*rand*) mod N
4. The third miner is assigned to position hash(hash(*rand*)) mod N and so on until all the miners are assigned a position.
5. If a collision occurs then the miner is assigned to the next available position. (To the next least N).

3 Implementation

Shuffling miners

1. Created the field *minerid*.
    - minerid : The ID of the miner in Miner class
    - The minerid is created with the formula : $(p * rowindex) + (columnindex)$
    - Where p : The number of primary miners
    - rowindex : row index of the miner in the 2D array
    - columnindex : column index of the miner in the 2D array
2. Miner generates a cryptographically secure random number. Used the *java.security.SecureRandom* class in Java to generate the random number.
3. Miners broadcast the signed hash of random numbers. Planning to use the signing function similar to Ken's.
4. Use a broadcast function so that each miner sends their random number to other miners.
    - Use a hashtable to look up the MinerID $->$ Generated random number
    - Use a hashtable to look up the MinerID $->$ Hash of the generated random number
5. Use a verify function so that the random numbers and the hashes matches for each miner along with the signatures.
6. Random numbers are concatenated in order of $MinerIDs$ and then hashed to get the random variable *rand*:
    - For concatenation, I am planning to add all the random numbers which may overflow. Another option is to concatenate as a String. Suggestions needed.
7. Create a bench miner of 2D array of size MxW. Use a helper array of size N to track of the filled miner positions.
8. The contents of the helper array will be initialized to -1.
9. The first miner is assigned to position *rand* mod N, update the helper array.
10. The second miner is assigned to position hash(rand) mod N, update the helper array and so on.
11. The helper array will be updated so that if a collision occurs the miner is assigned to the next available position.
12. With the help of the helper array, we shuffle the active mining set.
13. Then we figure out which are the primary miners, the secondary miners and the bench miners.