Bitcoin and Cryptocurrency Technologies

Assignment 4: Exploring Mining Strategies

For this assignment, you will explore game theory by implementing a strategy for a miner in a simplified model of Bitcoin. Your miner will run alongside several of our miners, which we've configured execute various deviant strategies ("attacks") that have been described in literature.

At initialization, every miner begins with the same genesis block and is assigned a number of parameters, including its "hash rate". The proof of work is simulated in the sense that if a miner has X% of the global hash power, it will be able to find X% of blocks.

The simulator runs in steps, where each step represents 1 second. There two main types of events:

- at every step, a new transaction will be generated and immediately distributed to all miners.
- about every 600 steps (Poisson arrivals with rate of 1 block per 600 steps), a miner will find a block. The "winning" miner will be randomly selected according to the hash power distribution--a miner with X% of the global hash power will find ~X% of blocks.

When a miner finds a block, it does not have to immediately publish the block. It can strategically wait before releasing the block to other miners (as in selfish mining). When it does publish the block, the simulator will introduce a randomized delay of 10-30 steps (seconds) before it actually reaches other miners. In essence, miners make three fundamental decisions: which block to extend, which transactions to include when mining, and when to publish a block that's been found.

There is a simplified model of transactions: a transaction just has a ID, an input address, and a transaction fee. So the idea of a transaction graph, for instance, doesn't make sense. The input address and transaction fee may be useful because some strategies may whitelist or blacklist transactions based on those fields.

All miners, including yours will implement the following interface:

```
public interface Miner {
   // hear a new transaction. This is called at every step of the simulation
   // (each step represents 1 second)
   public void hearTransaction(Simulation.Transaction tx);

   // Hear about a new block found by someone else.
   // The simulator ensures that the Block is valid (no double spends)
   public void hearBlock(Block block);

   // called when this Miner has "found" a block. The Miner must return
   // the block that it would like to publish. However, the block is not
```

```
   // actually released to other miners until it is included in publish_block()
   public Block findBlock();

   // returns a list of blocks that the Miner would like to release to
   // other miners. Each Block in the List must have been previously
   // returned on a call to find_block(). The List should be ordered
   // so that the blocks closest to genesis are first
   public ArrayList<Block> publishBlock();
}
```

Further, when the miner returns a block in findBlock(), it must include an ownerId field. Miners have the option of using the same value of ownerID each time, or a different one for every block, or something in between. To prevent a miner from impersonating another, only the simulator can generate valid ownerIDs and it will return different IDs for different miners. Miners must call the function "long getNewId(Miner m)" provided by the simulator. It returns a *long* id which is then registered by the Simulator as belonging to Miner m.

The Simulator will verify that your miner does not:

- publish a block that includes transactions that are contained in ancestor blocks
- try to create a block with an ownerId it does not own
- publish a block before publishing its parent block (if your miner found both those blocks)

The Simulator will quit with an error message if one of the above happens. Further, the Simulator will prohibit your miner from adding a Block whose height differs from the maximum-height block by more than 10. This is to simplify the implementation; you should not be building such long forks anyways.

Your miner will run against the following classes of reference miners:

1. Default behavior
2. Selfish miner -- doesn't publish blocks as soon as it finds them
3. Feather forking against a set of blacklisted transaction input addresses
4. Decide between branches based on the newest max-height block, instead of the oldest
5. Ignore blocks that have transactions with very high transaction fees; instead try to include those transactions in your own blocks. Goes back to longest chain if our fork falls too far behind.

Each of these behaviors can be run in one of 4 configurations. The miner may use the same ownerID for each of its blocks, or a fresh one each time (the simulator generates new ownerIDs randomly, so there's no pattern connecting the ownerIDs assigned to a miner). Further, the miner may or may not announce the strategy it's using as part of the blocks it generates.

We will run tests in a number of settings: different mix of miner types, global hash rates, block reward/transaction fee ratio, and minimum transaction fees.