Problem Set #5 *

Soft Deadline: TBD
Final Deadline: TBD

1 Block and Transaction Validation (contd.)

If you have not already implemented the following yet, do them now:

1. Ensure that a transaction does not have multiple inputs that have the same outpoint. (It should be clear that this is required for a valid transaction.)

2. Ensure if present that the `note` and `miner` fields in a block are ASCII-printable strings up to 128 characters long each. ASCII printable characters are those with decimal values 32 up to 126. If this is not the case, send back an `INVALID_FORMAT` error.

3. Ensure if present that the `studentids` field is an array with at most 10 ASCII-printable strings each containing up to 128 characters. If the `studentids` field is present but does not follow these constraints, send back an `INVALID_FORMAT` error.

2 Mempool UTXO

In this exercise, you will maintain a mempool and update it based on new transactions and blocks.

1. Implement a data structure for the mempool. You should maintain a list of transaction ids in the mempool and also maintain the required state that allows you to update your mempool when you receive new transactions and blocks.

2. Initialize the mempool state by applying the transactions in your longest chain. (On booting, you can first determine your longest chain using the responses to your `getchaintip` requests.)

3. On booting, also send a `getmempool` message to ask your peers for their mempools.

4. On receiving a `mempool` message, request from peers the transactions corresponding to the `txids` in the `mempool` message using `getobject` messages.

*Version: 2 – Last update: July 1

5. Listen for transactions as they are gossiped on the network. If a new transaction is valid with respect to your mempool state, add it to your mempool and update the mempool state. If a transaction cannot be added to your mempool due to an an already spent transaction input, send back an `INVALID_TX_OUTPOINT` error.

6. When a new block arrives that is added to your longest chain, update your mempool by removing transactions that are already included in the block, or are now invalid. Update your mempool state.

7. Deal with mempool updates when your longest chain reorgs. Refer to class notes for the steps involved.

3 Sample Test Cases

IMPORTANT: Make sure that your node is running at all times! Therefore, make sure that there are no bugs that crash your node. If our automatic grading script can not connect to your node, you will not receive any credit. Taking enough time to test your node will help you ensure this.
Below is a (non-exhaustive) list of test cases that your node will be required to pass. We will also use these test cases to grade your submission. Consider two nodes Grader 1 and Grader 2.

0. Reset your transactions/mempool database before submitting for grading. This is so that transactions that your node might have earlier considered valid but are actually invalid are removed from the database.

1. Grader 1 sends one of the following invalid objects in an `object` message. Grader 1 must receive an `INVALID_FORMAT` error message, and Grader 2 must not receive an `ihaveobject` message with the corresponding object id.
   a) A transaction with two inputs that share an outpoint
   b) A block with more than 128 characters in the `note` field

2. Grader 1 sends a valid transaction with two inputs (spending outputs with different public keys). Grader 2 must receive the transaction when it sends a `getobject` with the corresponding transaction id.

3. Grader 1 sends a `getmempool` and `getchaintip` message to obtain your mempool and longest chain.
   a) The mempool must be valid with respect to the UTXO state after the chain.
   b) Grader 1 sends a transaction that is valid with respect to the mempool state. Grader 1 again sends a `getmempool` message and this time the mempool should contain the sent transaction.

c) Grader 1 sends a transaction that is **invalid** with respect to the mempool state. Grader 1 again sends a `getmempool` message and this time the mempool should **not** contain the sent transaction.

d) Grader 1 sends a coinbase transaction. Grader 1 again sends a `getmempool` message and this time the mempool should **not** contain the sent transaction.

e) Grader 1 will send a longer chain (causing a reorg) and then send a `getmempool` message. The received mempool must be consistent with the new chain:

   i. It must not contain any transactions that are already in the new chain or are invalid with respect to the new chain UTXO state.

   ii. It must also contain transactions that were in the old chain but are not in the new chain and valid with respect to the new chain UTXO state.