SPCDSS'25 Prof. Tom Van Cutsem, Dr. Davy Preuveneers

Bitmunt - Task 5

High level Overview

After completing this task, your node should be able to:

- maintain a mempool.

Please also be sure to read the respective sections in the Protocol Description. At the end of this task, your node should implement the complete protocol.

1 Mempool UTXO

Every node should store a list of transactions that are not yet confirmed in a block. This is called the mempool. It should satisfy the following properties:

- It is possible to extend the current longest chain by a block containing precisely all transactions in the mempool.
- If a node receives a transaction `Tx` from the network, it should check whether it can be added to its mempool. This is possible if `Tx` spends only from unspent outputs that are also not spent by any transaction currently in the mempool. The same applies when receiving a `mempool` message - a node should attempt to add every transaction referenced in that message to its mempool.
- If the current longest chain changes, i.e., a new block is found or there is a chain reorganization, the current mempool M needs to be re-computed. Then, attempt to add the transactions of M to M'. This is done in the following way. The node sets the mempool state to be equal to the state $s(B)$ of the latest block B. Then, the node tries to apply all transactions in the old mempool against this new state $s(B)$, one by one, in the order they appeared in its old mempool. If a transaction from the old mempool can be applied, it is added to the new mempool state. Otherwise, if it cannot be applied, it is thrown away. For instance, if block B contains a transaction that was in the old mempool, that transaction is discarded and does not make it into the new mempool because it spends from outputs that are already spent. Similarly, if a transaction that was in the old mempool conflicts with the new state $s(B)$, it is thrown away. This means that if two double spending transactions appear in a block and in the old mempool, the transaction in the block takes precedence.

It is not required to optimize the placement of transactions in the mempool further.

In the case of reorgs, the current state of the chain is rolled back until the latest common ancestor between the current longest chain C' and the new longest chain C. Let us consider

the following example: C' has blocks $\{B_0, B'_1, B'_2\}$, and C' has blocks $\{B_0, B_1, B_2, B_3\}$. When a reorg takes place, this is what happens:

- the latest common ancestor B_0 between C' and C is identified, and its state $s(B_0)$ is the new state of the chain;
- all transactions in blocks $\{B'_1, B'_2\}$ are undone and transactions in blocks $\{B_1, B_2, B_3\}$ are applied against the state $s(B_0)$. If at any point a transaction cannot be applied, the whole block containing it is discarded;
- if C is the new valid longest chain, the chain with state $s(B_3)$ and ending with B_3 is adopted: the new mempool can be computed by trying to apply transactions in $\{B'_1, B'_2\}$ against the new state $s(B_3)$ (in the same order as they appear in $\{B'_1, B'_2\}$) and, finally, by trying to apply transactions in the old mempool (in the same order as they appeared in the old mempool).

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 setting the mempool UTXO to the UTXO after your chain-tip block.
3. Send a `getmempool` message immediately after sending the hello message.
4. On receiving a `mempool` message, request from peers the transactions corresponding to the `txids` in the `mempool` message using `getobject` messages if they are unknown to you.
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 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. More information can be found in the Protocol Description.

2 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 Grader2.

1. Reset your 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. With the exception of the Genesis block, your node should not know any objects.

2. Grader 1 sends a number of blocks and transaction objects.

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:

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

The **soft deadline** for this task is **8th May, 2025, 11.59pm**.
The **hard deadline** for this task is **15th May, 2025, 11.59pm**.
We will not accept any submissions after the hard deadline. Please ensure you correctly tag your commit and push it to the `main` branch.