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

Bitmunt - Task 4

High level Overview

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

- validate any block that references unknown blocks by implementing recursive block fetching.
- implement the longest chain rule.

1 Recursive Block/Chain Validation

Here, you will build upon your block validation logic from the previous homework to recursively validate the parent block, if you have not done so already.

1. When you receive a new block, check if the parent block is available in your database (after the proof-of-work check). If not, request your peers for the parent block using a `getobject` message. Optionally, you may set a timeout to avoid indefinite waiting.
2. Recursively validate the parent block before validating the new block. This would include downloading all missing blocks in the prefix of the new block (until you hit genesis or a previously validated block), validating them, and updating the UTXO set after all these blocks.
3. Validating the parent block also ensures that you have an updated UTXO set, allowing you to validate transactions in the new block. If a parent block is not found or invalid, send back an `UNFINDABLE_OBJECT` error.
4. The height of a block is the number of blocks it is away from the genesis. You need to calculate the height of each block when recursively validating them. The genesis has height 0 and heights increase along the chain. When you receive a new block, determine its height and check that if the block has a coinbase transaction, its `height` field matches the height of the block. If not, send back an `INVALID_BLOCK_COINBASE` error.
5. Check that the timestamp of each block (the created field) is later than that of its parent, and earlier than the current time. If not, send back an `INVALID_BLOCK_TIMESTAMP` error.

Intuition behind recursive object fetching

Assume your node connects to the network for the first time (thus not knowing any objects except the genesis block) and there are already many transactions and blocks in the blockchain. Using *getobject* messages, your node can request objects. But how does your node know which objects to request? How does downloading and verifying the blockchain state work?

The protocol specifies a message `getchaintip` which lets you request the id of the topmost block from a peer (i.e., the topmost block in their view of the blockchain). Since each block points to its parent block, therefore, you will start downloading and verifying a chain of blocks!

2 Longest Chain Rule

In this exercise, you will implement the longest chain rule, and sync the chain tip with your peers.

- Implement the longest chain rule, i.e. keep track of the longest chain of valid blocks that you have.
- Upon connecting to a peer, send a `getchaintip` message directly after the `getpeers` message you are supposed to send.
- On receiving a `getchaintip` message from a peer, respond with a `chaintip` message with the blockid of the tip of your longest chain.
- On receiving a `chaintip` message which references a block that you do not know, request the object from the network.

  Should the blockid already show that the block is invalid because of invalid PoW, send a `INVALID_BLOCK_POW` error and disconnect.
- On receiving a new block in an `object` message (either by your request or not), which references a block that you do not know, validate the chain by recursively downloading and validating each block in the chain. Update your longest chain if required.

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:

1. Reset your block/object database before submitting for grading. This is so that blocks that your node might have earlier considered valid but are actually invalid (because, for example, you had not implemented coinbase height check earlier) are removed from the database. This is to prevent your node from adopting a longer chain which is actually invalid.

2. Grader 1 sends one of the following invalid blockchains by advertising a new block with the `object` message. Grader 1 must receive the corresponding error message, and Grader 2 must not receive an `ihaveobject` message with the corresponding blockid. If Grader 2 requests this object, it must receive an error message UNKNOWN_OBJECT and must not receive the object.

   (a) A blockchain that points to an unavailable block.

   (b) A blockchain with non-increasing timestamps.

   (c) A blockchain with a block in the year 2077.

   (d) A blockchain with an invalid proof-of-work in one of the blocks.

   (e) A blokchain that does not go back to the real genesis but stops at a different genesis (with valid PoW but a null previd).

   (f) A blockchain with an incorrect height in the coinbase transaction in one of the blocks.

3. Grader 1 sends a number of valid blockchains. When Grader 1 subsequently sends a `getchaintip` message, it must receive a `chaintip` message with the blockid of the tip of the longest chain.

4. What if Grader 1 and Grader 2 have different views of the chain tip?

The **soft deadline** for this task is **1st 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.