Contents

1. Introduction

Proof of work (PoW) means that nodes compete in solving hash puzzles, and the one that has more restricted and non-monopolized resources than others has more chances to propose the next block and get a reward for it. In case of Bitcoin, nodes compete by using their computing power. Because of proof of work, it becomes hard to create new identities, because computing power will serve as a "tax" on identity creation and therefore on the Sybil attack.

In this assignment, a template of the code that implements the PoW functionality for a given blockchain application was completed. The implementation adjusts the difficulty of the PoW to match a given time interval.

2. Description of the implemented functionality

1. **make_empty_block(self, bits):** set the values for the fields in the block.

2. **mine(self, block):** get the bits from the block, compute the target, compute the hash of the block, update (increment) the nonce while the hash is not smaller than the target, add this hash to the block, and add the block at the end of the blockchain.

3. **datetime_to_seconds(time):** added an "if-elif-else" condition to check the type of the input parameter, because this function was used twice: one time, in the function *change_target(prev_bits, start_time, end_time, target_time)* to calculate the time span of two input string parameters that then were converted to the type 'datetime.datetime'; another time, in the main loop where it was applied for converting the type 'datetime.timedelta'.

4. **get_target_from_bits(bits):** split the bits into two parts - bits 0...23 represent the header, bits 24...31 represent the exponent; calculate the target as

$$target = bits[23:0] \times 2^{8\times(bits[31:24]-3)}.$$

5. **get_bits_from_target(target):** calculate the bitlength of the target, take the left 24 bits as the header to keep the exponent to a minimum, calculate bits.

6. **change_target(prev_bits, start_time, end_time, target_time):** calculate the previous target using the previous 32-bit bits, calculate the time interval between the first block and the last block, recalculate the target to adjust the time span between the start and the end of a round to the targeted time interval.

7. **main loop**: corrected the divider used for the calculation of average time from *number_of_blocks* to *(number_of_blocks - 1)*. It was done in order to exclude the time of last mining and keep only one timestamp per block.

3. Results and Conclusions

The main code creates empty blocks (32 blocks in a round for the targeted time interval) and measure the time it creates a single block on average. 4 targeted time intervals (4 difficulty levels) were tested: times = [2, 4, 6, 10] seconds per block on average.

The results are as following:

average time = 0.44567022580645166

difficulty = 1.0

average time = 1.7960395483870968

difficulty = 4.571429544565894

average time = 5.575911580645162

difficulty = 10.448985476737805

average time = 5.74961229032258

difficulty = 11.596565381478358

average time = 7.774320032258064

difficulty = 20.847761227413766

Since we have a small runtime, the error in the averaging of the times seems to be big and the above timings are different from the desired ones.

The blockchain is stored in the json file attached to this report.

4. References

1. Bitcoin Wiki "Difficulty", https://en.bitcoin.it/wiki/Difficulty, accessed on April 6, 2020.

2. Yarkin Doroz. Bitcoin Mining: a course presentation, 2020.