Peer to Peer Systems and Blockchains
Academic Year 2019/2020

Final Term

Jacopo Massa - 543870

April 2020

1 Analysis of the Bitcoin protocol

1.1 Interaction with the Bitcoin Blockchain

1.1.1 Block 351846

- **How much in total did the miner who found this block receive for doing so?**
  It has received 25.00946136 BT **in total**.
- **How many inputs and outputs are there in the second transaction?**
  There are 1 input and 2 outputs for this transaction.
- **Give a likely explanation for why the two recipients do not receive the same amount.**
  The two outputs differs in their amount because of two possible scenarios:
  1. The smallest one is the change of the transaction, which is redirected to another address owned by the same user who owns the unique input address.
  2. It could be simply a payment to another address of another user.
- **What are the total sum of the input/output for this transaction?**

  Input: 6.086934 BTC.

  Output: 6.086734 BTC.
- **Why is there a difference?**
  The difference is justified by the fee of the transaction, whose amount is 0.0002 BTC.
- **What are the first 6 characters of the recipient who received the fee?**
  The first six characters are 1LH3Qt. They belong to the user who mined the block.

1.1.2 Block 351833

- **What is odd about this block? Why do you think this occurred?**
  The block is made by a single transaction, with no input and one output, which corresponds to the address of the user who mined the block. In particular, this is a *Coinbase transaction*, which consists in generating bitcoins to reward the miner who found the *Proof of Work (PoW)*, and transfer these BTC to the address of the miner.
- **Who owns the output address of the transaction with id 6841c.....?**
  The owner is the miner of the block (AntPool).

- **Look at all of the transactions that the miner who found that block has received. Roughly how much USD is this?**
  Roughly 1 billion and 250 million dollars (when I visited the page, but it constantly change due to the BTC/USD change).

- **How has this miner managed to win so many blocks? Describe how the miner has likely managed the rewards it received form all these transactions**
  The miner is probably part of a mining pool, so a joint group of miners who combine their computational resources over the Bitcoin Blockchain. The rewards for the mined blocks are then divided among all the members of the mining pool that have contributed according to the proportion of each individual's processing power.

- **How "hard" was the block to find?**
  The difficulty to find the hash for this block was 49.446.390.688,24

- **On average, how many nonces would the miners executing the PoW contemporary to the winning miner had to try before finding a satisfactory nonce?** Following the formula at this link, and assuming that `time = 10 min` (the average time to mine a block):

$$\begin{aligned} hashrate &= 49.446.390.688,24 * 2^{32}/600 \\ &= 354 \text{ petahashes/second} \end{aligned}$$

  As the wiki states, the formula (so the usage of 2^{32} factor) is a simplification.

1.2 Answering some questions

- **Name three typical spending conditions that are supported in Bitcoin.**

  Pay-To-Publick-Key (`P2PK`) is a script pattern that locks an output to a public key. This script contains a public key and a `CHECKSIG` opcode, and if a node wants to unlock it, it has to provide a valid signature.
  When the script runs the `CHECKSIG` opcode compares the signature with the public key, and pushes a 1 into the stack if the comparison is valid. These scripts are not very used; they can be commonly found in coinbase transactions in the earlier blocks of the blockchain.

  Pay-To-Publick-Key-Hash (`P2PKH`) is the most common script pattern. It is similar to `P2PK`, but the lock contains the hash of a public key (and not the public key itself). The main advantage of this approach is the shorter version of the public key (hashed one), so, for example, we can easily share it with other users.

Pay-To-MultiSignature (`P2MS`) is a script pattern that locks bitcoins to multiple public keys, and require signatures for some (or all) of those public keys to unlock it. The most common practical usages are:

- **1-of-n**: anyone of the n different parties can approve the transaction.
- **2-of-2**: both the separate parties must approve the transaction.
- **2-of-3**: any two out of three parties can approve the transaction (the most common in escrow contacts).

When this script is executed, all of the signatures and public keys are pushed into the stack. Then the `CHECKMULTISIG` compares each signature with each public key.

- **Explain the three-step protocol used for exchanging new transactions/blocks in the Bitcoin P2P network.**

1. A node creates a transaction, which is signed with one or more signatures;
2. The transaction is broadcasted on the Bitcoin P2P network, by each node that propagates the transaction until it reaches almost every node in the network.
3. When it reaches a mining node, the node includes the transaction in a block of the blockchain, and after it has been confirmed by some subsequent blocks, the transaction is a permanent part of the blockchain.

- **List at least 3 differences between Bitcoin's UTXO model and Ethereum's account-based**

1. The first difference is that in Ethereum accounts and data exist as ordered data in a state database, and transactions modify the state. Instead in Bitcoin, the consumption of old UTXOs and the creation of new UTXOs leads to the change of an implicit state.
2. There is another difference between smart contracts and bitcoin scripts. Smart contracts are written with a quasi-Turing-complete language (Solidity). The "quasi" is because of the *Gas Cost*, that limits the language. The language used for bitcoin scripts is a Turing-incomplete language.
3. Ethereum has a faster mining rate (10-20 seconds) with respect to the one of Bitcoin (10 minutes).
4. The mining reward of Ethereum remains constant due to the inflationary monetary policy, so the fact that no new ETH is created, unlike bitcoin which halves periodically its reward.

5. The *nonce* in Bitcoin is the number useful to mine a block. In Ethereum it is a number attached to a transaction and identifies also the number of transactions sent from a given address. It can be used to prevent the reply attack (always in Ethereum).

- **What information is required to prove that a transaction is in a block? And how does a Bitcoin light client verify it?**
  The only kind of node that can prove this is a full node, which requires the whole Blockchain (headers and transactions for each block). Then the full node will send the block header and the merkle branch linking the transactions to its block header to the SPV node (light node). The SPV will compute the merkle proof applying recursively an hashing function, starting with the received transaction to check. Then it will compare the obtained result with the merkle tree root stored in the block header (the only part of the blockchain the lightweight node stores).

2 Pay eggs' supply chain with Ethereum smart contracts

- **Implement the missing portions of the *complete()* function. They need to pay the transporter and, eventually, refund the penalty to the receiver.**

```
//// @notice Change the state from SHIPPING to ACCEPTED if the minimal price is not met; REFUSED otherwise. Callable by the receiver
function complete() public is_allowed(receiver) payable {

    uint amount = msg.value;

    require(state == State.SHIPPING, "Error, invalid initial state");
    require(amount == transporter_price, "Error, funds not match the expected amount");

    uint penalty = penalty_function.compute_penalty(samples_temperature, samples_bump, temperature_threshold, bump_threshold);

    if((transporter_price < penalty) || (transporter_price - penalty) < minimal_price) {
        // The penalty overcomes the difference between initial and minimal price

        (bool success, ) = receiver.call.value(amount)("");
        require(success == true, "Error while paying the receiver");

        state = State.REFUSED;
        emit refused(penalty);
    }
    else {
        // Accept the box and pay the transporter
        uint to_pay = 0;

        to_pay = transporter_price - penalty;
        (bool success, ) = transporter.call.value(to_pay)("");
        require(success == true, "Error while paying the transporter");

        state = State.RECEIVED;
        emit received(to_pay);
    }

    // Refund the receiver, and destroy the contract.
    selfdestruct(msg.sender);
}
```

Figure 1: The implemented method *complete()*.

- **Evaluate the cost in gas of the functions provided by the smart contract.**
  Using `Truffle` with `Ganache`, I've written a simple test in Javascript, that retrieved the gas used for three possible scenarios, shown in figure 2. The obtained average for each function is (in gas):

  Ship: 53906

  Push_data: 4278435

  Complete: 289187

  As can be seen from figure 2, the *ship* function never change because it's not related to any external data. The *push_data* function costs almost the same; small variations are due to the higher/smaller order of the pushed data. As last, the *complete* function costs differenty in the three scenarios, due to the calculation of different amounts of penalties.

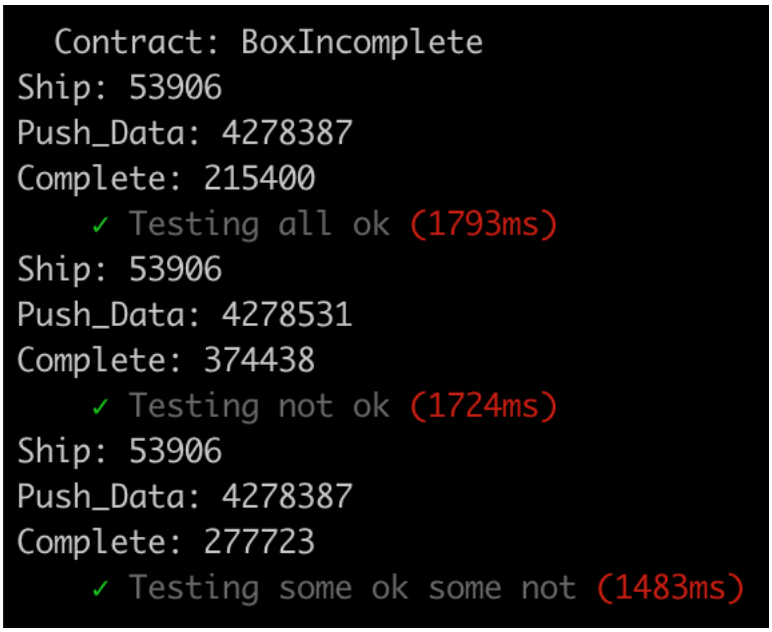

Figure 2: Snapshot of the jasvascript test's results.

- **Compute the fees to be paid by B to store the samples, *push_data()*, for a single shipment.**
  Considering that a shipment lasts for 8 hours, and the data are sampled every 5 minutes, I've followed the case where data are pushed all togegther (so two arrays of 96 samples).
  To calculate the fee:

$$fee = gas\_price * gas\_used$$

  I've used the Eth Gas Station tool, that provides also a fee calculator and the updated ETH/EUR converter. I obtained that for a single shipment, considering the average gas price (34 Gwei when I visited the site), the fee amounts to about €30:

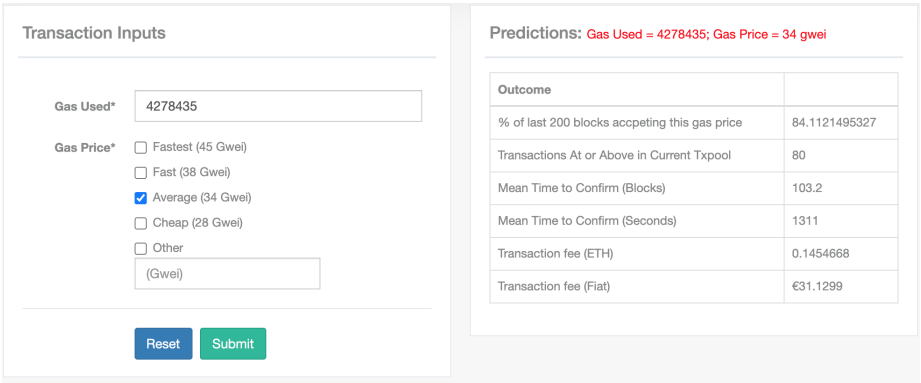

Figure 3: Snapshot of *eth gas station* tool.

- **What are the security concerns to keep into consideration while developing a smart contract similar to this one? List the vulnerabilities that might arise.**

  - The first vulnerability, which I noticed immediately during the completion of the required code, concerns the **integer overflow**[1], specifically for the calculation of the penalties, if too many "bad" data are loaded, then the "if" guard will be always false, as the variable "`penalty`" assumes a value similar to a MAXINT (compared to other languages). To avoid this, I added an additional condition to the if guard (see figure 1).
  - Always in the context of the eggs' supply chain, a **Re-entrancy** vulnerability may arise, so when a contract sends Ether before having updated its internal state. If the destination address is another contract, it will be executed and can call the function to request Ether many times. In the `Box` contract this is not possible due to the requirements imposed by the `require` command, more or less at the beginning of each function.
    These controls avoid also a possible **unpredictable state** when the *complete* function calls a `PenaltyFunction` contract's method. So we're sure that the state before and after the sending of the transaction to invoke the other contract, will be the same, even if the `Box` contract has received other calls or transactions.
  - Obviously, when the producer creates a contract, it has to specify valid addresses for the transporter and the receiver in the constructor of the contract. In fact, if the selected addresses are *orphan* ones (they are not associated to a contract or a user), the Ether in the transaction involving them will be lost forever (**Ether lost**).

[1] https://arxiv.org/pdf/1902.06710.pdf

3 Complex Network Analysis

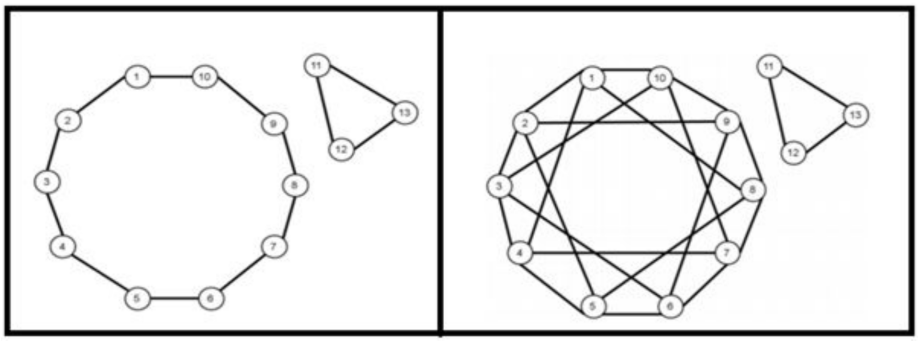

Figure 4: Pair of graphs to analyze.

- **Compute the clustering coefficient. What do you observe? Is the value of the clustering coefficient a good measure to characterize the structures of the graphs?**

  The local clustering coefficient `CC` for a vertex `v` is then given by the proportion of links between the vertices within its neighbourhood divided by the number of links that could possibly exist between them.
  For both graphs, following the previous definition, all the nodes from 1 to 10 has a `CC` equal to:

  $$CC = \frac{0}{1} = 0$$

  The nodes 11,12,13 has:

  $$CC = \frac{1}{1} = 1$$

  So, if we consider the *global clustering coefficient* as the average of the clustering coefficients of the single nodes of the graph, we obtain, always for both graphs:

  $$GCC = \frac{0+0..+0+1+1+1}{13} = 0,23$$

  This approach doesn't adequately characterize the two graphs. As can be seen from the figure, they have similar topologies, but different links among the nodes. The same `CC` is obtained due to the definition itself: in fact, we can notice how in the graph B, even if there are more links, each node has its neighbors not connected one with the other, canceling the numerator in the definition of `CC`.

- **Describe a better way to characterize these structures.**

  A better way to distinguish the two graphs could be in terms of **graph density**, defined as the ratio of the number of edges with respect to the maximum number of possible edges. Considering `m` the number of edges, and `n` the number of nodes:

  $$p = \frac{m}{n(n-1)/2} = \frac{2m}{n(n-1)}$$

  Applying the previous formula on both the graphs, we obtain:

  - **First graph:** $p = \frac{2*13}{13(13-1)} = 0,1\overline{6}$
  - **Second graph:** $p = \frac{2*23}{13(13-1)} = 0,29487179$

  This approach had better characterized the two different structures. The higher value for the graph B shows its more particular topology.