Cryptocurrencies and Blockchain

Sheet 1, submission by Konstantin Bork

Assignment 1: Bitcoin Various Topics

1. A malicious ISP can launch such an attack, it just has to send a transaction to the blockchain at the same time the
   user sends one. To succeed with the attack, the ISP can try to bribe the next node, which can belong to the ISP, to
   extend the block containing the double spend, or find the next block by itself.

2.
a) The next block to be found determines which block ends up in the consensus branch. If Minnie's block has a successor
first, all miners switch to this branch and her block is in the consensus branch, same applies to Mynie's block.
In the case blocks are found at the same time again, this process repeats until one chain is longer than the other.
b) The rate of orphan blocks increases with the amount of any latency. This includes at first the network latency.
Furthermore, any form of buffering and querying at the nodes, may it be in the hardware or in the software,
affects the rate of orphan blocks. Because of all these factors to be taken into account, it is difficult to develop
a solid formula.
c) Taking data from here (https://api.blockchain.info/charts/n-orphaned-blocks?timespan=all&format=json), there are 527
orphan blocks (29th November 2018). The rate of orphan blocks therefore is less then 0.3%.
d) She has not wasted her effort. In fact, she can confirm Minnies block and so tells other participants to trust this
block.

3. As the mining is a Poisson process, we can use this formula:

$$p(x|\lambda) = \frac{e^{-\lambda} * \lambda^x}{x!}$$

We set $\lambda = 1$

as this variable determines the difference to the mean time of 10 minutes.
The probability to find a block in the next 10 minutes then is:

$$P(1) = 1 - P(0) = 1 - e^{-1} \approx 63.2$$

4. We want to know:

$$p(x \geq 6|\lambda) = 1 - p(5|\lambda) - p(4|\lambda) - p(3|\lambda) - p(2|\lambda) - p(1|\lambda) - p(0|\lambda) = 0.99$$

When we do all the math, we get $\lambda \approx 13.1$
which means Bob should wait 131 minutes as λ
tells us how many times we should take the mean time to find a block of 10 minutes.

Assignment 2: Validation of transactions

scroogeCoin.TxHandler.java

```
package scroogeCoin;

import java.util.Arrays;
import java.util.List;

public class TxHandler {

    /**
     * Current collection of unspent transaction outputs
     */
    private UTXOPool utxoPool;

    /**
     * Creates a public ledger whose current scroogeCoin.UTXOPool (collection of un
     * {@code utxoPool}. This should make a copy of utxoPool by using the scroogeCo
     * constructor.
     */
    public TxHandler(UTXOPool utxoPool) {
        this.utxoPool = new UTXOPool(utxoPool);
    }

    /**
     * @return true if:
     * (1) all outputs claimed by {@code tx} are in the current scroogeCoin.UTXO po
     * (2) the signatures on each input of {@code tx} are valid,
     * (3) no scroogeCoin.UTXO is claimed multiple times by {@code tx},
     * (4) all of {@code tx}s output values are non-negative, and
     * (5) the sum of {@code tx}s input values is greater than or equal to the sum
     * values; and false otherwise.
     */
    public boolean isValidTx(Transaction tx) {
```

```
            List<Transaction.Output> allTxOutputs = tx.getOutputs();
            boolean case3Matched = allTxOutputs.stream().distinct().count() == tx.numOu
            if (!case3Matched) {
                return false;
            }

            boolean case1And4Matched = allTxOutputs.stream()
                    .allMatch(output ->
                            utxoPool.contains(new UTXO(tx.getHash(), allTxOutputs.index
                                    && output.value >= 0.0D); // check case 4
            if (!case1And4Matched) {
                return false;
            }

            List<Transaction.Input> allTxInputs = tx.getInputs();
            boolean case2Matched = allTxInputs.stream()
                    .allMatch(input -> Crypto.verifySignature(tx.getOutput(input.output
            if (!case2Matched) {
                return false;
            }

            // At last, check if sum of input values are at least as big as output valu
            return allTxOutputs.stream().mapToDouble(output -> output.value).sum()
                    <= allTxInputs.stream().mapToDouble(input -> tx.getOutput(input.out
        }

        /**
         * Handles each epoch by receiving an unordered array of proposed transactions,
         * transaction for correctness, returning a mutually valid array of accepted tr
         * updating the current scroogeCoin.UTXO pool as appropriate.
         */
        public Transaction[] handleTxs(Transaction[] possibleTxs) {
            Transaction[] validTransactions = Arrays.stream(possibleTxs)
                    .filter(this::isValidTx) // Only get all valid transactions
                    .toArray(Transaction[]::new); // Return the filtered transactions a

            // Update the UTXOPool
            Arrays.stream(validTransactions).forEach(tx -> {
                List<Transaction.Output> outputs = tx.getOutputs(); // get all outputs
                outputs.forEach(output -> utxoPool.removeUTXO(new UTXO(tx.getHash(), ou
            });

            // Return the valid transactions
            return validTransactions;
        }
    }
```

You can find the source code of the whole program in the src folder.