COP5615 – Fall 2018

Project 4.1 MitBits (Money Inventory Transfer Bits)

- Siddhant Mohan Mittal UFID: 6061-8545
- Prafful Mehrotra UFID: 1099-5311

Project Directory Structure

```
├── Mitbits_Cryptocurrency
│   ├── _build/
│   ├── config/
│   ├── lib
│   │   └── mitbits
│   │       ├── application.ex
│   │       ├── driver.ex
│   │       ├── Node.ex
│   │       ├── NodeSupervisor.ex
│   │       └── MinerSupervisor.ex
│   │       └── Miner.ex
│   │       └── Utility.ex
│   ├── main.exs
│   ├── mix.exs
│   └── test/
│   │   └── mitbits_test.exs
│   │   └── test_helper.exs
└── README.md
```

About the project

We have created a new crypto-currency named MitBits or MB using the blockchain principal. Decentralizing the money exachange process with no trust relying strongly on maths based proves of crytography, proof of work and irriversability.

Defining Application Architecture

- The Application consist of a supervisor which manages miner supervisor and node supervisor under it. This protects us application for getting terminated, as it restarts the application in such situation.
- Miner supervisor and node supervisor hold beneath them miners and nodes genservers respectively. Why two different genservers for miners and nodes? This is because miners will be mining blocks asynchronous; competeting with each other to solve the puzzle of the block,

this will slow down or maybe create a time out situations for the other processes like transactions, wallet updating etc.

- Each node has a public key to identify them. Also each node also have private key(secret key) which is not accessible to other nodes.
- Only the nodes participate in transaction but to make a miner participate in the bitcoin exchange process, each miner node has a connected node which the same public key. The only difference it that the name of miner node is "miner_public_key" and connected node is "node_public_key"
- Each node maintains following states; sk (private key), pk (public key), list of blocks(blockchain), Indexed block chain, all pending transaction pool, wallet amount.
- Each node handles functions like adding transactions, updating wallet, validating blocks, performing forking and deleting transaction.
- Each miner only has mining functions. Where it validates sum pool of transaction and solve the block puzzle to mine mitbits. For transactions, block chain etc it calls its associated node.

Implementation

- How it really works under the hood? To start the application off, we created a gensis block by just spawning a single miner. This first miner mined the first block by hashing a random string "This is starting of something big in the year 2018" with a reward transaction of 1000 mitbits and thus creating our first genesis block.
- Later we generate numNodes and numMiners. Since numNodes are some of the first nodes to join our application and to promote it further we award 10 mitbits to each numNodes from the first miner and add these transaction to pending transaction pool of each node. The miner now competes to find the proof of work of new block. If a miner is successful, this reward transaction of 100 mitbits is made part of the blockchain thus mining these 100 mitbits out of thin air.
- Whenever a new block is mined it is broadcasted to all the nodes but since this can be fraud block all nodes first validate the transactions in the block, if there are correct all miners moves to never set of transaction otherwise they still mine on the previous hash of the block.
- Then we give two different functions, one which allows you to make a random transaction between two nodes and other allows any node to join our system.
- This random transaction is used for simulation for 10K transactions.
- The other function is used to add new node which copy a blockchain from any node and makes a indexed block chain and then join other node is listening transactions and mined blocks.

Protocol

Public key and private key generation

- Public key and private key pairs are generating using Elliptic-curve cryptography.
- Private key is used by nodes to digitally sign the node. using a sign(messgae,sk) function. This sign function takes a transaction and convert it into a string, then it used sk to sign it. Changing even the smallest thing changes this digital signature making the once signed transaction irreversible and also digital signature protects the information of who created
- Public key which is used to identify a node, is also you to protect the authenticity of a transaction which we talked above. A verify (message, signature, pk) function turns true if the signature is created by the sk of this pk
- We create a sha256 hash of pk to the base 16 to name a node and also store in the :ets so that very node is aware of the other public key, but note the sk is only known to the node itself not the application.

Transactions

- A transaction is map having following keys: message that is a map log which contains keys from, to and amount; signature that is the digitally signature given to the transaction by sender using its private key; timestamp that is the time of creation.
- When a new transaction is created it is broadcasted to all the nodes in the system.

Blocks

- A block in a block chain contains following values; Its hash value, hash value of previous block, some set of transactions and timestamp.
- A block in our application have a maximum size restriction of 6 transactions that is 5 pending transaction and 1 reward transaction.

Mining

- Mining is a process of doing proof of work to add a set of pending transaction into the block chain and get rewarded for your computation work.
- In mining a miner picks up 5 transaction, add previous block hash to it , add a reward transaction and add a random number called as nonce. Then pass this created string into sha256 to create a string in base 16.
- If the value of the created sha256 is less than a target value then the miner has hit the jackpot and creating a block of approved transaction and adding a reward transaction of 100 mitbits to his account into blockchain.
- If the value is greater than target he increase the nonce by one and redo the same process. If some other miner comes up with a valid block, he deletes transactions of new block from pending transaction pool and try the mining process again on the new set of transactions.
- Miner mining a block and hitting jackpot is actually beneficial to the system as this is a way application adds transaction to block chain and approve them.
- Target value in our system is sha256 string having first four bits as zeros
- The importance of hashing previous block hash in next block is that if a attacker goes and modify a block then he has to do proof of work for every other block after it. Hence making it computationally impossible.

Blockchain

- Blockchain is a list which is maintained by all the nodes. Each participant nodes first joining the system starts with the genesis block. But later on new node is joining system copies the blockchain from any other node.
- Whenever a new block is mined it is broadcasted to all the nodes. Every node updates it's block chain.
- If we iterate over the complete blockchain we will notice that it contains all the valid transactions so far. And going through each message of transactions we can see how mitbits are transferred using from, to and amount keys.

Wallets

- Wallet is the amount each node has. Each node initially have 10 mitbits for loyality of joining the system at start. The first miner node contains 1000 mitbits for creating genesis block.
- For any futher transactions as they get approved and the block is broadcasted to all the nodes, every node updates it wallet in following manner. If the block contains transaction having pk of this node in "from" value of the message then the amount is subtracted from the wallet; otherwise if it is in "to" value then amount is added to the wallet.

Below Protocols are implemented as the bonus part of this project to make the system more efficient and attack resistance

Indexed Blockchain

- Indexed blockchain is an extra protocol we have implemented to make the system more efficient. The idea is that each node maintains a state of indexed blockchain on "from" and "to" values of public keys of the node. - Before going into the importance of this first let us see how this is maintained. Whenever a new block is broadcasted the nodes updates it's indexed blockchain by just reading one block.
- If a new node joins the application it iterates over the complete Blockchain once to create a indexed blockchain. Now it behaving like other nodes which update indexed blockchain by just reading one broadcasted block
- This helps us in validation transactions that is we don’t have to iterate over blockchain again and again. Creating a indexed blockchain is one time effort. And validating is an important part which can't be ignored due to no trust model of our application. Hence if every node validates new blocks and transaction by iterating over the blockchain will make the taken. slower

Authentication

- Authentication is an important protocol to protect our application from fraud nodes and fraud miners.
- Authentication is done by both miners and nodes. in this step verify() function is called to verify the digital signature; if is signed by the correct sender and on the correct transaction message
- Miners perform authentication while selecting transaction for their mining process. Thus eliminating fraud transactions from there blocks and void their block getting disapproved by Consensus protocol
- Node performs authentication when receive a broadcasted block, if it contains any wrong signed transactions it gets disapproved. This ensures that no fraud miners broadcast a block of wrong transactions.

Validation

- This is as important as authentication and both goes hand in hand and are performed in the same way at nodes and miners.
- But the way of doing this is different. The validation checks every transaction according to the indexed blockchain to ensure the sender making the transaction actual had the balance in this account to perform this. Thus no random coins are generated on the way. Also node validate the reward transactions in the block to verify that no miner cheated and gave himself more rewards.

Consensus

- The consensus protocol is used to avoid addition of fraud blocks to blockchain. The main idea is that if an attacker miner tries to send fraud blocks to the nodes the validation step at nodes will ensure they don't add the fraud block to their blockchain and other miners still hash the new block on previous block's hash
- Due to fact that more than 50% CPU computation power is of honest node the honest blockchain always grows faster than the fraud chain which the attacker was trying to create.

How to test and run ExUnit tests

Run the following commands

```
$ cd Mitbits_Cryptocurrency
$ mix deps.get
```

- For running all the test cases together run following commands, all the test cases will run in parallel. It will take around 4 minutes to run all the test cases. If a process terminates don't worry the supervisor will take care of it and respawn the terminates processes. To run individual process, see next section

```
$ mix test
```

We have written 5 test cases to cover all the major functionality and scenarios

Test Case 1: Genesis block test

- Expectation: This test case will print the genesis block (that is the first block) which will have a random string and a reward transaction of 1000 Mitbits to miner
- To run it

```
$ mix test --only test1
```

Test Case 2: Creating 10 participants public key, private key pairs using Elliptic-curve cryptography

- Expectation: Creating public keys and private keys for each user node. This test case will print 10 private keys and public keys in following format [sk,pk]. Following that will be printed the sha256 hash of public keys of the nodes visible to all other nodes
- To run it

```
$ mix test --only test2
```

Test Case 3: Creating 10 digitally signed transaction between 10 participants when they join the system.

- Expectation: This test case shows the structure of the transactions, also since they are the first nodes to join the system hence as incentive 10 mitbits are awarded from first genesis miner. Note the signature will be different in all txn even they are signed with same private key proving the irreversibility of the txn. This is the signature made with the private key of the first miner.

- To run it

```
$ mix test --only test3
```

Test Case 4: Mining bitcoin and creating block chain

- Expectation: Since mining is process creating a proof of work to approve transactions which takes computation power of each miner and all run asynchronously, this can be on-going process hence simulation is terminated after some time. Output is the mined blocks and after termination the blockchain.

- To run it

```
$ mix test --only test4
```

- **Test Case 5: Testing of wallet. We run a simulation between 20 user nodes and 5 miners. 10,000 random transactions are made between any two nodes.**
  - Expectation: Terminated after some time the blockchain and updated wallets of each node is printed. Updated wallet is tested and compared with the txn in block of the blockchain
  - To run it

```
$ mix test --only test5
```

- **Test Case 6: Authentication, Validation and Consensus Test. We run s simulation between 20 user nodes and 5 miners. 10k random transactions are made any two nodes. These transactions can be fraud ones where a user tries to add send more mitbits than he have in this account or miner include invalid transaction to create a fraud block.**
  - Expectation: Terminated after some time as the process is infinite and we are just testing the functionality. The blockchain and updated wallets of each node is printed. The blockchain contains all the valid and authentic transactions. The wallet balance printed at the end shows no transaction with fraud and unrealistic amount. We created random transaction having amount as large as 100000, which we know won't be possible in less than <10k transactions as no node won't have such large balances in their wallets. Also note see the balances of no node goes in negative.
  - To run it

```
$ mix test --only test6
```