CSE 598 – Engineering Blockchain Applications – Project 2 Part 1

Title: **Dash Public Blockchain transactions**

Student Name: Abhishek Patil

ASU ID: 1216126522

Abstract:

This report first introduces blockchain transactions and explains the important elements to discover. It then lists some terms and their meanings to equip the reader with a better understanding of important words. Then, it describes the solution for implementing a transaction from one sender to another in the Dash Public Blockchain network. After this, it discusses some issues and the solution taken. Finally, the report cites some related works and provides a conclusion presenting some lessons learnt.

Keywords:

Dash, Blockchain, Blockchain Network, Transaction, UTXO, ChainRider

Introduction:

In today's world, we hear a lot of people exchanging goods and services with the use of cryptocurrencies. Such exchanges/transactions must be carried out somewhere with help of some software. It is natural to be curious about what happens in the background to keep the security of the parties involved in the transaction, successfully transfer the funds, and ensure no faulty plays happened.

These exchanges are called transactions. We would like to know how such transactions are carried out and written in code. In a more detailed way, we want to understand how the senders and receivers are recognized, what API and methods are needed, what procedure must be followed and much more.

Project 2 Part 1 of the course provides a very simple example to practice writing a transaction between two users in the Dash Public Blockchain.

Terminology:

Some important terms involved in this part of project are:

Transaction: A transaction is transfer of rights to some amount of funds (example Bitcoin, Dash etc.) from sender to receiver. Transaction has at-least one input and one output. The input(s) are outputs from previous transactions. The funds available from the inputs is used in the transaction to send to the receiver.

Unspent Transaction Output (UTXO): UTXO is a set of transaction outputs that have not been spent by accounts in the blockchain network yet. As they are unspent, they can be seen as the current balance for the accounts in the network. A later transaction will use some of the outputs in the UTXO as its inputs.

Dash Public Blockchain: In this project, we leverage the Dash public blockchain network which uses Dash as its currency. This is a peer-to-peer (also called p2p) payments network where transactions are facilitated and recorded on a decentralized, distributed, public ledger [1].

ChainRider: To communicate with the Dash Public Blockchain we need an API. This is provided by ChainRider [2] and within the API we focus on Transactions section.

Testnet: This is the name of blockchain network we will be using for testing transactions.

Goal Description:

The goal of this part of the project 2 is to write a transaction that successfully sends some amount of Dash using the Dash Public Blockchain and given APIs.

Description of proposed solution:

The solution starts with first acquiring the project folder codebase, and then running 'npm install' in that folder to install the necessary libraries. The libraries we want are dashcore and got which are already included by the codebase.

The codebase has two addresses along with their private keys. I started with the address of '*yTYZjnTuepHbVAcoWq4g7f5teXru4KSJMa*' as sender and '*yNpEzKCvS2Vn3WYhXeG11it5wEWMButDvq*' as receiver. I stored the addresses and respective private keys in *send_info* and *recv_info* dictionaries. Also, a *send_amount* variable is set with value of 10000 satoshis. Moreover, I added code to swap between the sender and receiver addesses in case the given sender account has insufficient funds.

Next, I created an account on the ChainRider website at https://chainrider.io/register and got my own authorization token. I put this value in the *token* variable.

Then, I created a function that takes the sender and receiver's info as dicitonaries (mentioned before). In the function, I first declared a *retry* dictionary object that sets *limit* to 10 and *statusCodes* to a list of 429 and 500. This is used with the *got* call to access a *url* 'https://api.chainrider.io/v1/dash/testnet/addr/${send_data_address}/utxo?token=${token}'. This way the *url* has the sender's address and my authentication token. As we provided with the *retry* dictionary, the *got* call retries until it is successful.

The *got* call is asynchronous, and thus, I used the *Promise.then()* function to track the response along with a *catch* block to identify errors. The response has multiple unspent transaction outputs to choose (UTXOs) from. I iterated through these UTXOs to find the first transaction having at least the *send_amountI* amount of satoshis. I used the *txId*, *outputIndex*, *address*, *script*, and *satoshis* fields from the selected transaction and created the *utxo_obj* dictionary object. A balance amount to be sent back to the sender is then calculated in *bal_amount*. Initially, *bal_amount* is set to -1 to indicate. A negative value indicates that not enough funds are available and is useful on an if condition alter in the code.

There is a possibility that none of the UTXOs have the *send_amount* value for the transaction. Thus, *utxo_obj* stays empty and in that case, I re-loop through the UTXOs and create a list of UTXO objects which would sum up to at least the *send_amount*. Like the previous case, I also included code to calculate *bal_amount* for this case.

If sufficient amount is present, then the *bal_amount* must be greater than or equal to 0. Otherwise, a console message is printed to address the fact that sender does not have enough amount. In the former case, I used the *dashcore.Transaction()* available by the dashcore library to create a new transaction. This transaction used the *utxo_obj* (created previously) as the *from()* method parameter, receiver address and *send_amount* as the *to()* method parameters, sender address and *bal_amount* in the *change()* method parameters, and sender private key as

the *sign()* method parameter. The receiver address, sender address and sender private key were passed to the function through its parameters *send_data* and *recv_data*.

The resulting transaction's *String* is stored in the *rawtx* field of a new dictionary which I named as *raw_trxn_dict*. This dictionary also needs the *token* field which we obtained at the beginning. The *raw_trxn_dict* object is put into the *json* field in options dictionary. This *options* also has the *retry* and *responseType* fields. With the use of *options* and the ChainRider *url* 'https://api.chainrider.io/v1/dash/testnet/tx/send' I used it in *got.post()* to send raw transaction as described in [3].

Finally, when successful, the body field of the response is printed to show the transaction ID. I was able to get a successful transaction from '*yNpEzKCvS2Vn3WYhXeG11it5wEWMButDvq*' to '*yTYZjnTuepHbVAcoWq4g7f5teXru4KSJMa*'. A screenshot of the transaction in Dashevo [4] is included in figure 1.

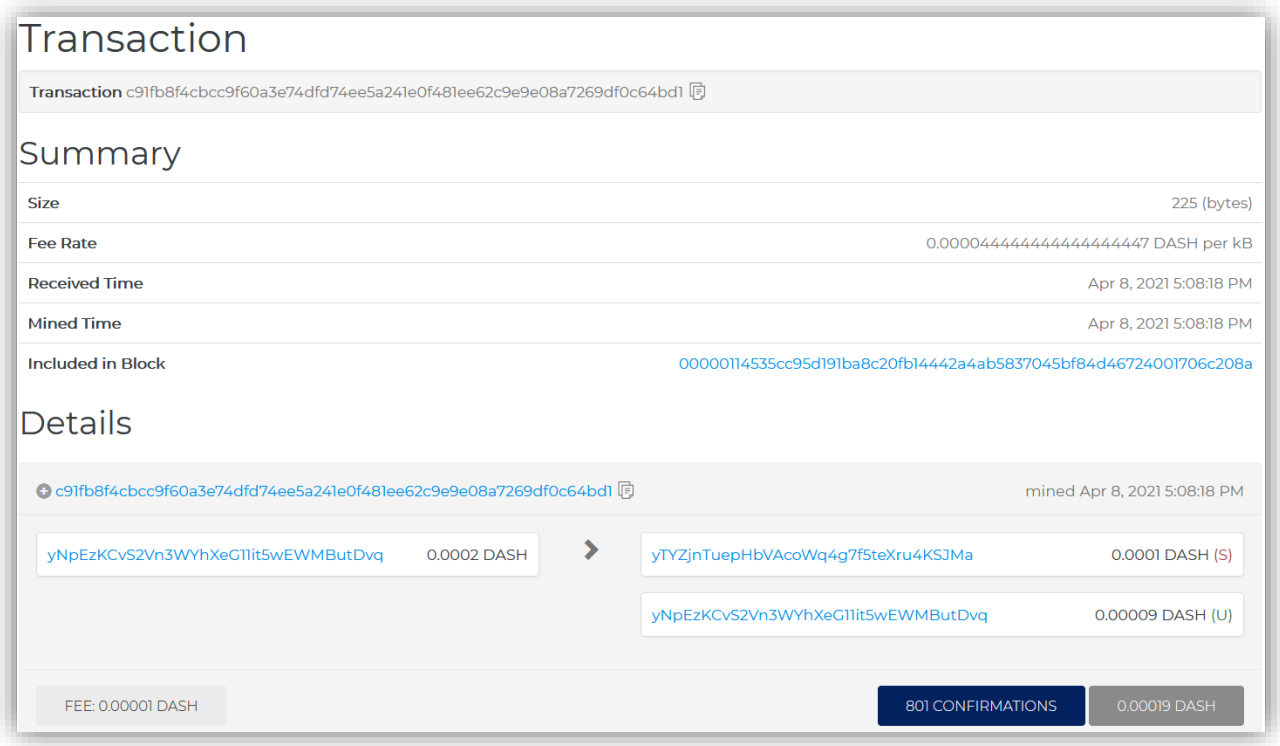

Figure 1. My Dash transaction:
'c91fb8f4cbcc9f60a3e74dfd74ee5a241e0f481ee62c9e9e08a7269df0c64bd1' on Dashevo

Issues Faced and Methods used to resolve them:

The first issue I faced was with proper 'npm install' execution. This was because I had the previous installation from project 1. An update to a recent version solved the problem.

The next one was not an issue, but just getting a hook on a proper way to use the *got* library, which I had never used before. Specifically, it was hard to debug *got* request parts of the code with the constant errors returned by the ChainRider API. The only solution, which I used, was to thoroughly look through the *got* documentation [5], and this way I was able to make it work.

The other major issue was the API kept returning 429, 500 and 400 errors. For 429 and 500 status codes I used the *retry* option in *got* calls. The 400 'Bad Request' was not avoidable, so I kept running and was able to get a valid transaction in the end.

One final issue was to test my code logic for the case where I make a list of UTXO objects for using multiple transactions that sum up to the *send_amount*. The problem involves creating a specific scenario which would not be good as the entire class is using same two addresses.

Related Works (if any):

[1]. "What is Dash?," *Dash Platform*. [Online]. Available: https://dashplatform.readme.io/docs/introduction-what-is-dash. [Accessed: 07-Apr-2021].

[2]. "Introduction," *Chainrider*. [Online]. Available: https://www.chainrider.io/docs/dash/#transaction-apis. [Accessed: 07-Apr-2021].

[3]. "Introduction," *Chainrider*. [Online]. Available: https://www.chainrider.io/docs/dash/#send-raw-transaction. [Accessed: 08-Apr-2021].

[4]. "Dash Insight," *Home | Insight*. [Online]. Available: https://testnet-insight.dashevo.org/insight/. [Accessed: 09-Apr-2021].

[5]. "got," *got - npm*, Mar-2021. [Online]. Available: https://www.npmjs.com/package/got. [Accessed: 08-Apr-2021].

Conclusions:

In conclusion, the project was completely new to me and hence, I learned a lot about how the Dash public blockchain transactions work behind the scenes. I now have an idea of how the sender and receiver are recognized, the correct steps to carry out a simple transaction, and also use of the *got* library to send and receive responses from the ChainRider API. In general, this project has equipped me with more knowledge and tools which could be useful in the expanding world of Blockchain and its industry applications.

Bibliography: none