Blockchain

Trayson Keli'i
Arizona State University

Abstract

The purpose of this paper is to exemplify the practical application of the concepts that were taught in CSE 598. This class consisted of two major projects. The first dealing with the creation of a private network via the Hyperledger Framework and the second dealing with the manipulation of the Dash public testnet network.

1 Introduction

The adoption of blockchain technologies has increased in recent years, with various initiatives being deployed by both private and public organizations. It is, though, an emerging technology that is increasingly developing. It prevents the indulgence of central servers and enables peer-to-peer interaction by building a decentralized framework. It has the potential to build a completely accessible and open archive, bringing accountability to government and elections. The two distinct blockchain paradigms discussed are public and private blockchains.

1.1 Private Blockchains

A consortium blockchain is another name for a private blockchain. A private blockchain is one that is only accessible by invitation. A single party is in charge of the blockchain. Permission to read, write, or audit the blockchain is required by all participants. To keep those pieces of data private, the blockchain may provide several layers of data access. As a result, private blockchains provide greater protection, privacy, and efficiency. Private blockchains may be tailored to particular industries, such as finance and government services, due to their private existence. The transactions and data are not accessible to the general public and can only be viewed by the parties involved.

1.2 Public Blockchain

The public blockchain is an innovation as to how transactions are dealt with on a public network. In terms of handling transactions, a methodology focused and used in the project is keeping track of unspent transaction outputs known commonly as UTXOs. This UTXO paradigm was established to avoid the double-spending problem and to validate that transactions are successful. A UTXO can be used once in a transaction. After it is used the transaction is recorded on the public ledger and the "value" of that transaction no longer exists.

2 Description of Solutions

This section describes the implementation and solution of both the public (Dash) and private (Hyperledger) blockchains.

2.1 Private Blockchain

The bulk of the time was spent going over the lab specs to be sure the problem was well understood. After understanding the 7 tasks required in terms of implementation, it was realized that there were some inconsistencies regarding the naming of certain variables and functions. Cleaning the initial project preceded the initial implementation of tasks. The tasks 1 – 3 were straight forward implementations of getter, setter, and update functionality. The minimal understanding required to implement the logic of these functions came through a key parameter named 'ctx'. This parameter was to be passed in as the first argument for any function surrounding the blockchain and it represented the context in terms of the transaction which was being executed by the function. The later tasks, 4 – 6, the use of a persistence layer was necessary. This led to understanding the documentation and project implementation of couch-db. The final task was simple error handling and only required a one line text change to return an error string to notify the user of any unexpected behavior. With the foundation in place, the network was ready to be tested. This phase of development proved to be a lot more difficult than the creation. The initial setup of the environment was trivial and provided on the Hyperledger documentation site. Navigating the use cases of the sample projects provided from Hyperledger is where the bulk of the time was spent. This in turn revealed the necessity of the creation of environment variables so that each peer-node had all the proper dependencies needed to function on the network. Essentially, a startup script was developed in bash to automate this process. The steps below describe this script:

1. Start up the network and initialize couch-db.
2. Create a channel with two peers Org1 and Org2.

3. Deploy smart contracts from the project to both Org1 and Org2.
4. Write commands to test the functionality of the smart contracts deployed.

Going through this process validated the logical/syntactic errors in the code. For the blockchain to get these updates the network would need to be restarted and the script would be run again, creating a fresh build with the altered logic with two peer-to-peer nodes. All requirements were able to be created and validated through this iterative process.

2.2 Public Blockchain solution

Before beginning any code, familiarization with the ChainRider API was necessary. Some time was spent going through the documentation to understand the use of functions, especially the functionality of new Transaction(). An API key was required to perform any http requests. Registering for an account gave access to this API and forming the url with the query param token would track all the API calls made specific to the key for the created account. To determine which address would be the sender and which would be the receiver a GET call was made to both addresses via a REST library called axios. The lists returned were then sorted by the aggregated satoshi for each list and the one that had the most would be the sender while the other would be the receiver. As the coding continued a need for reusable functions led to the creation of the following helper functions:

1. *parseUrl*: Receives one parameter that is a string of the address in question. This would simply return the amount of UTXOs for the given address.
2. *createTx*: Receives six parameters (utxos, fromAddr, toAddr, amount, pKey, and fee) which essentially creates a valid transaction based off of input.
3. *sendRawTx*: Receives one parameter rawTx which is the formed transaction from the createTx function (in a string of hexadecimal representation of the transaction). This would ultimately send the transaction to the testnet blockchain via the following endpoint: POST dash/testnet/tx/send. Note that a request body was sent that had the rawTx value and the API token.

With this core functionality in place, higher level problem solving could be used to determine the flow of the program. Something to note, there was no thought of testing and validating functions at this point. It was a lot of trial and error until the proper results were achieved. During development it was noted that sending the requests to the server was slow and that the API queue was backed up for the shared accounts provided. This led to the creation of helper functions to write the UTXOs of given addresses into local json files. With this in place, the API could be avoided by having a local cache of the UTXOs for the specified address (which could then be filtered, sorted, and modified for testing purposes). Ultimately, this led to the creation of two helper functions made to support the fine tuning of the core functionality:

1. *writeToFile*: Received two parameters, the first being the name of the file, the second a json string of the data to persist.
2. *readFromFile*: Received one parameter, the name of the file. The json data would then be loaded into the project upon return.

After working out the kinks of the functions (properly serializing the transaction, validating the amount that should be sent, etc) an established process was formed regarding the transfer of funds. This logic was encapsulated in a function called *transfer* and it received one parameter *sendAmount*, the amount being sent in satoshi (note: sender and receiver are global static constants previously defined and used throughout the method). The basic flow went as follows:

1. Get UTXOs for sender
2. Write UTXOs to file for safe keeping
3. Filter UTXOs based upon the sendAmount (UTXO must be greater than it).
4. Determine the fee for transaction
5. Create the transaction
6. Send newly created transaction
7. Write txid returned from the call to local json file

This concludes the implementation of the purposed solution for the public blockchain.

3 Results

3.1 Private Blockchain Results

Deploying smart contracts via Hyperledger Fabric on a private network is somewhat trivial. The implementation described in this paper is not scalable but met requirement expectations. In terms of a production ready product using a private block chain a lot of other criteria would need to be met (security, versioning, etc).The project itself merely scratched the surface of what the Hyperledger Framework is capable of. In the end simple business logic was able to be deployed to its own distributed system with a proper persistence layer, which was the goal of the assignment.

3.2 Public Blockchain Results

At the end of the project a single transaction ID was the sum total of all the work put in regarding the transfer of UTXOs from two separate accounts. These transactions are logged on the testnet of DASH and reveal the power of a public ledger (imagine the applications surrounding voting, supply chain, and similar problems). The resulting transaction ID from this specific project is: *cd7afeed47516ebc5744ad3ffc2cdb687473ada798382c2898f839e0793ab903* It should also be noted just how new and cutting edge the practical application of public blockchains are. Testnet and helper APIs are hard to maintain and tend to receive less support than the main public network, thus making it hard to experiment. It was also learned that more than 30 people should not be sharing an address to perform transactions because it leads to unexpected problems in queuing the UTXOs and the lack of message descriptions provided for such an edge case proved to be difficult to perform root cause analysis on the problem (basically all the issues faced were due to the amount of users performing API calls with the same address around

the same time). The validation process of UTXOs on public blockchains are very secure and accurately depict the transferring of securities.

4 Lesson Learned

4.1 Public Blockchain

I've seen how modern and cutting-edge public blockchain applications are in practice. I've also realized that testnet and helper APIs are difficult to manage (based on some of the ways I've seen when debugging errors in my code). I found that sharing an address for transactions with more than 30 people is not a good idea because it can lead to unexpected problems (basically all the issues faced were due to the amount of users performing API calls with the same address). Throughout it all, I can see how the UTXO validation process on public blockchains is extremely reliable and accurately depicts the transfer of securities. I've also gained a better understanding of the satoshi, the cryptographic unit of measurement. This allows for proper representation of a crypto asset, and we can submit UTXOs as a type of currency by manipulating satoshis. I've also read that many of the issues that plague most testnets aren't really issues in mainnet, and that code deployed to a public blockchain from main should behave similarly. In terms of APIs, I've discovered that mainnet has a lot more support. This only strengthens my conviction that blockchain is the way of the future.

4.2 Private Blockchains

To summarize, using Hyperledger Fabric to implement smart contracts is relatively easy. I'm not sure if my solution is sustainable in terms of a production-ready product (the use of bash commands to handle the creation and deployment of smart contracts). I'd also like to know what rights a client requires to participate in the blockchain. If this is just scratching the surface of Hyperledger Fabric's capabilities, then I am certain there are far more use cases that can be imagined from IoT to micro-architecture of video games. At a glace, it seemed that private blockchains deal more with system level understanding while the public blockchain dealt with web and transactions. Both of which are powerful and will have an impact on our foreseeable future.

References

[1] Axios (2021) [Website]. Retrieved from https://www.npmjs.com/package/axios

[2] Chainrider (2021). [Website]. Retrieved from https://www.chainrider.io/docs/dash/

[3] Couch DB (2021) [Website]. Retrieved from https://docs.couchdb.org/en/stable/

[4] Dash (2020). [Website]. Retrieved from https://docs.dash.org/en/stable/introduction/about.html#

[5] Fabric (2020) [Website]. Retrieved from https://www.hyperledger.org/use/fabric

[6] Hyperledger (2020). [Website]. Retrieved from https://hyperledger-fabric.readthedocs.io/en/release-2.2/getting_started.html