CS Capstone: Bradycoin

Brady Bhalla

***Abstract*—This paper is a reflection of my entire project and research, which explains each step of how I created Bradycoin as well as applications of blockchain technology and future steps I could take with Bradycoin.**

1 INTRODUCTION

For my capstone project, I built a blockchain from scratch which is used to run a crytpocurrency I call Bradycoin. To complete this project, I needed to understand the cryptographic foundations that are used to make the system secure, as well as the protocol for creating transactions and blocks and the Proof of Work consensus algorithm. Finally, I had to make a program that would allow nodes on the network to communicate with each other. Much of this project is modeled after Bitcoin, the most popular (and one of the earliest) cryptocurrency.

2 PROCESS

2.1 Language

I wrote the code for Bradycoin in Python because that is the language I am most comfortable with and I am able to quickly write functional code without worrying too much about variable types, memory, etc. I also stuck with only the standard library of Python, so anyone can run it without needing to install extra packages. Python doesn't run the most efficiently of all languages, but for the scale I care about, it works fine. As I will explain later, Bradycoin nodes use HTTP to communicate with each other, so theoretically someone could write a node in a different language and it would work fine with my Python nodes.

2.2 Cryptographic Foundations

2.2.1 *Digital Signatures*

In the physical world, a signature is used to verify that a certain person approved a document. Each person's signature is unique, so it is very difficult to forge this signature on other documents.

Digital signatures mimic this functionality. A signer can take a digital document and produce a signature that anyone else can verify corresponds to that signer

and that document. It is very difficult for anyone else to produce a signature from that signer for a different document.

In Bradycoin, I use the ECDSA (Elliptic Curve Digital Signature Algorithm) to sign and verify messages. Elliptic curves can be used to describe a finite field with its own special addition rules. The ECDSA algorithm relies on the fact that it is very easy to perform repeated addition (or multiplication by a constant) on points in an elliptic curve but extremely difficult to undo this operation. The specific elliptic curve used in Bradycoin is the same elliptic curve used in Bitcoin, called Secp256k1.

2.2.2 *Hash Functions*

Hash functions are one way functions which map a message of any size to a single integer within a certain range. The hash, or result of the hash function of a message, contains no information about the message itself, and it is essentially impossible to generate a message which has a specific hash.

Hash functions are used in conjunction with the ECDSA to create digital signatures and also play an essential role in the Proof of Work consensus algorithm.

Throughout Bradycoin, the SHA-256 hash function is used, which is one of the most popular and secure hash functions used in the real world.

2.3 Transactions and Blockcahin

2.3.1 *Identity*

Before getting into how cryptocurrency works, there needs to be a way to establish identity online. In the real world, we can say something like "Ryan pays Brady $10," and we can tell which people "Ryan" and "Brady" are because their identities are tied to their names.

In the online world, there are multiple steps to creating an identity. In addition, since messages can be sent by anyone, there should be a way to sign them. To create an identity, start by picking a random private key. This is like a password that gives access to the identity. In the context of cryptocurrency, a private key allows one to send money from an account.

After the private key is picked, a public key can be generated using the ECDSA algorithm.

The public key can then have some operation done to it involving hashing, which results in an address. This address is the "name" corresponding to an online identity.

Because of the security of ECDSA and hashing, the path from a private key to an address is one-way; it is easy to find the public key from the private key or the address from the public key, but essentially impossible to do either of those the other way around. This idea can be used to create transactions.

2.3.2 *Transactions*

Let's return to the statement "Ryan pays Brady $10." However, since we will be in the digital world, say that Ryan and Brady have both generated private keys, public keys, and addresses. The statement might instead say something like "duwtqWHyj pays a0Wi1e2tc $10." Since Ryan is paying, he would also attach a signature to the message, which is generated using his private key.

Anyone is able to see this statement and signature, but it is possible to verify that it is from Ryan. First, the public key must match the address. If this is the case, then the signature can be checked. If the signature, which is verified using the message and public key, checks out, the message was from Ryan (or at least someone with the private key corresponding to the address "duwtqWHyj")

2.3.3 *Ledgers and Verification*

A ledger is essentially a list of transactions which determines how much cryptocurrency each person has. To verify a transaction before adding it to the ledger, not only must the signature match, but the money must be available from the sending account and the exact transaction must not already exist. This last condition is extremely important, because if we have the signed message "Ryan pays Brady $10", there is nothing stopping me from copy and pasting that dozens of times. In Bradycoin, each transaction is accompanied by a random number, which should never occur more than once.

The Bradycoin ledger is divided into blocks, which are small chunks of transactions that are added together. Each block contains the hash of the previous block, so the ledger is forced into one specific order. This ordered list of blocks is called a blockchain.

2.3.4 *Proof of Work*

However, there are often different ideas of which blocks should exist to make up the ledger. The method used by many cryptocurrencies (including Bradycoin) to solve this problem is called the Proof of Work consensus algorithm. This works by requiring a large amount of computation to be done in order to make a valid block. A random number, called a nonce, must be chosen such that the hash of the block including the nonce has a specific property. Since the hash is a random-seeming one-way function, the only way to match the property is to try different nonce values over and over.

After this is established, the method of choosing which blocks make up the ledger is very simple: always choose the longest chain. This works because the longest chain has had the most computation done on it, so the most people think it is valid. In order to trick the system and rewrite the history of the blockcahin, an entity would need over 50% of the entire computing power of everyone using Bradycoin.

2.3.5 *Programming the Blockcahin Structure*

One of the biggest programming challenges I had while making Bradycoin was creating a system of keeping all chains of the blockchain organized and always choosing the longest one. To add a new block, the state of the ledger needs to be known at the previous block. Unfortunately, it is much too expensive to store a copy of the entire ledger at every block, so something else needed to be done.

I ended up storing a single copy of the ledger, which could be transformed. When adding a block to a different chain than the longest one, the blockchain is searched like a graph and builds a list of actions which can transform the current ledger into the needed state.

The actions can be executed one by one, and then the new block can be verified and added. If after adding, this block is in the new longest chain, nothing more will occur. Otherwise, the list of actions from earlier will be executed in reverse and the ledger state will revert back to the longest chain.

I had to use a piece of paper to figure out an algorithm that would accomplish this efficiently, but after a lot of debugging, it worked!

2.4 Mining

When Bradycoin starts running, there is no cryptocurrency available, so it is impossible to send any money. All Bradycoin is initially created through a process called mining, which happens when blocks are created. Since so much computation is needed to create a block, the person who mined it gets newly made Bradycoin, which they are then able to send. The amount starts at 50 and slowly decreases after enough blocks are added. The total amount of Bradycoin that could ever exist is 4,716,950, which would happen after 389,999 blocks have been mined.

2.5 Network and Communication

At this point, all aspects of the Bradycoin protocol itself exist, but there is still the problem of allowing different Bradycoin nodes to communicate with each other to manage the blockchain. This was one of my biggest challenges because the program got very complicated very quickly after I started writing the code.

I used HTTP for communication between nodes because it is a very common protocol that Python has built-in libraries for. It also made debugging much easier because I could send requests to the nodes using a web browser. Basically, each node would set up an HTTP server which could be used by other nodes to request blocks and lists of node addresses using GET. It would also use the same server to allow information about new blocks and transactions to be sent to it using POST.

After a new block was introduced to the network, each node would broadcast it to other nodes, which guaranteed that every node connected to the network would receive the block.

Each node also had many threads that would run in the background, periodically polling other nodes for any new sources or new blocks it hadn't heard about yet. This allowed the entire network to fairly quickly learn the addresses of every node and stay up to date.

With all the background threads, it was important to protect resources such as the ledger from being changed twice at the same time. To solve this problem, locking had to be used. This ensured that each resource would only be accessed once, but also made the program more complex. I had to take extreme caution not

to use locks incorrectly, because that could result in something called deadlock, which would make the program freeze forever.

3 APPLICATIONS

Cryptocurrency is a huge industry, and is projected to continue to grow rapidly over the next years. Because of how impactful cryptocurrency is, I think it is very important to have at least some understanding of how it works, which is part of the reason I wanted to do this project.

While cryptocurrency can be controversial, it also seeks to solve important problems in the world. Decentralization, for example, means that no one entity is in control. If someone is transferring money using a bank, they have to trust that the bank will execute the transaction and not steal any money. Using a blockchain however, people can be confident that their transaction is secure without having to trust anyone; it is all backed by cryptography.

There are numerous real world examples of cryptocurrency and blockchain technology being used in the real world. Bitcoin and Ethereum are two extremely widespread cryptocurrencies which can be used to transfer money securely. The Ethereum blockchain also has smart contracts, which allow for more complicated functionality. An example of this is NFTs, which record ownership of digital data on the blockchain. Blockchain technology can also be used for secure voting, record keeping, tracking, and more.

4 CONCLUSION

Overall, I have learned so much while working on this project. I have learned about how digital signatures work to verify things online, I have learned how to make a blockchain and why it is secure enough to use for sending money, and I have learned more about how to organize code so it can scale for complex projects. I have also gained an appreciation for the simplicity of the basic ideas behind blockchain; I, as a high schooler, was able to make a functional cryptocurrency, which is the same basic technology used to power a multi-billion dollar industry.

There is still much more I can do with this project (if I have enough motivation). The next step would probably be to set up servers that can run globally as Bradycoin nodes, so the network wouldn't cease to exist when I leave a local network. After this, I would want to make a website that would allow for easily

sending transactions to the network and extracting information from it. There is a website in existence that can be used to see the latest bitcoin blocks, what transactions are happening, etc., so it would be interesting to make a version of this for Bradycoin.

Whether I continue with this project or not, I am very satisfied with what I have accomplished this quarter and feel like I know so much more about blockchain.

Thanks for running this class and being such an amazing teacher over the past four years!