Metis Montessori Lyceum

Informatica · Amsterdam

A Robot That Owns Money

Joppe Koers (4VWO · N&G N&T)
and Noah Loomans (5HAVO · N&T)

supervised by
Jurjen Hickmann

22nd January 2018

Contents

1 Introduction

A Robot that owns Money seems like a strange concept. But rapidly developing technologies like the blockchain will make this possible in the near feature. We, as Informatica students are particularly intrigued by these technologies. This is why we will try to *test* and *implement* these technologies. We will have two goals. First, we want to see which/if cryptocurrencies are ready to be used right now, our requirement will be, among other things, that you are able to interact with the cryptocurrency from a light-weight computer. And secondly, we want to visualize this concept with a robot.

Our research question will be; Are cryptocurrencies ready to be interacted with on a light-weight computer?

I imagine most people will be pretty lost when reading that last paragraph, which is why a word register is. Therefore, we will explain a couple of terms in-short before continuing.

2 Technical Word Definitions

Blockchain A blockchain is a way to store and verify data *without* the use of a central authority. Technologies built on a blockchain usually have specific properties that make it virtually impossible to alter data that is already on a blockchain and also virtually impossible to add "invalid" data to a blockchain.

Cryptocurrencies Cryptocurrencies are blockchains designed to hold digital currencies with real-world value.

Light-weight computer Computers with very little processing power, memory, and storage. Take as an example the Raspberry Pi 3 Model B, this is a computer that is about the size of a credit card that you can use to browse the internet and watch YouTube videos.

P2P Network Normally, if two computers want to share data they connect to the same server, but in this case, they connect directly to each other, eliminating the in-between server. This connection is usually faster and more secure than the traditional way.

SSH This allows computers to send commands to other commuters over the internet.

3 The goal

The term blockchain and the term cryptocurrencies will be further explained in section 4.

So, what is the goal of this project then? **We will test to see if cryptocurrencies are ready to be used on a Raspberry Pi 3 Model B.** In order to do that, we will build a robot that uses the Raspberry Pi as it's brain. This robot will be able to receive payments using cryptocurrencies.

We will build a small robot (20cm x 20cm x 20cm) with a screen on it. On the screen, it will ask to send money using a cryptocurrency. When this money is successfully transferred the robot will move around. This symbolizes him transferring an object. It helps clarify the whole process, most people do not even know what a Bitcoin is, so it helps to visualize to as much of the process as possible.

Our hypothesis is that the cryptocurrency Bitcoin cannot be used on a Raspberry Pi, because if it's high resource requirements. We expect that we can use IOTA, however, because it advertises to IOT (Internet of Things) devices.

4 How Cryptocurrencies Work

4.1 Introduction

There are many cryptocurrencies, a few names are, Bitcoin[9], Ethereum[3], Ripple[10], and IOTA[4]. You might have heard of them in the news recently, but what is a cryptocurrency in the first place?

A cryptocurrency is a fully digital currency, that has no central place of authority. This means that, unlike in traditional banking, there is not a small group of people that are in charge of the money. Normally a bank has a large vault with a lot of money in it. We call this centralized banking. In the real world, people carry around cards that say how much money in the previously named vault is theirs. There are a couple of problems with this method. First, the bank could be robbed, then the people have right to money that doesn't exist. Next, the bank is regulated by people, and people make mistakes. So a bank can go bankrupt. And a bank is often influenced by a government that puts a tax on transactions and tracks them for suspicious behavior and fraud. But with *decentralized cryptocurrencies* we can avoid a central bank that is vulnerable to attacks and mistakes.

This may seem impossible, but it is made the opposite using a technology called blockchain. Before we can even talk about the blockchain, we need to talk about sha256, proof-of-work, and digital signatures. Which we will explain below.

4.2 sha256

$$sha256(\texttt{"hetmml"}) = \begin{array}{l}\texttt{1101011111000001111011011101111101}\\ \texttt{11001101100111101110100110110110}\\ \texttt{10111101111011110100110101011110}\\ \texttt{11101111101111010011010101110111}\\ \texttt{01101101011110011110011101001110}\\ \texttt{10110111011101111011011110011101}\\ \texttt{01111101111001111001101101101101}\\ \texttt{11100110101110000111101110101110}\\ \texttt{00110100011101111000111001011100}\\ \texttt{01101111101011010111100111010011}\\ \texttt{11011101100111101111111111011101}\\ \texttt{10111100110110111010111010011010}\end{array} \tag{1}$$

*sha*256 is a function that takes a number and returns another number. In formula 1 we use a piece of text as input and a binary encoded number as output, but keep in mind that as far as the *sha*256 function is concerned, these are both numbers. This function has three interesting properties that are very important for the blockchain to function.

1. Given the same input you will *always* get the same output. So if you run $sha256$(`"hetmml"`), you will *always* get `11010111...10011010`.

2. The output of the $sha256$ is seemingly random. If you change one bit of the input value, the output value will have completely changed. If we run $sha256$(`"hetmmk"`) for example, we will get `01110111...10111011`. This output is unpredictable, and the only way to know what the output will be is to run the function and see it for yourself.

3. Given an output value, it's almost impossible to calculate the input value without a lot of guessing. We call this a no-reversible calculation, meaning that you can do the calculation in one way. $a \rightarrow b$, *but not* $b \rightarrow a$. Old hash functions used to implement this by multiplying two prime numbers. If we use this simple multiplication function: $a * b = c$ and we know a and b we can easily calculate c. But because we are working with only primes for a or b, c's value can only have one solution. For example, $7 * 11 = 77$ you can only get the number 77 by multiplying 11 and 7, no other numbers. Now comes the use full part of the equation, if we only know c, how can we calculate a and b? It's simple we can't, at least not without a lot of trial and error. You might say "Well you can just try a couple million combiniations with a modern computer". Yes, indeed you can, but the larger the prime the longer it will take to try all the combinations, but if we use big enough numbers, it will be unfeasible to try all the combinations will take thousands of years a million's of dollars. In practice, this means that no one will even attempt to crack a single transaction encrypted with sha256.

This brings us to the next topic important to the blockchain.

4.3 Proof of work

What if you want someone to proof that their machine did a lot of work. One way to do this is with a so-called Proof of work, it works like this; You give them a number, let's say 947080, and you ask them to find a number that you can add to that number, so that the $sha256$ of that number starts with 8 zeros. We call this number a nonce. Remember, the only way to find that nonce is to try them all. You start at 0, and then you keep adding 1 to that number and checking the output of $sha256(number + nonce)$ to see if it starts with 8 zeros. This is a lot of work, but once you finally finish this task, I only have to run $sha256(number + nonce)$ *once* to see if the nonce is indeed correct.

I can easily, mathematically, verify that you did a lot of work.

4.4 Digital signatures

$$sign(message, sk) = signature \tag{2}$$

$$verify(message, signature, pk) = T/F \tag{3}$$

Digital signatures are a way to verify that someone (or something) actually said something. It is a way that allows you to be incredibly confident that no one tampered with that message and that the message actually came from the person that signed it.

It requires one to create a *public key* (pk), and a *private key*, also known as a *secret key* (sk). As the name suggests the private key is a digital key that you keep to yourself. We also have a public key, that you can give to everyone who asks for it.

Let say we have a piece of information that Alice wants to sign digitally, so you use function 2:

$$sign(\texttt{"Alice pays Bob €100,-"}, sk_{Alice}) = \begin{array}{l} \texttt{11101011010111101111110111110111} \\ \texttt{10010110111111010111001111011110} \\ \texttt{11011111011110111000011101110110} \\ \texttt{11101011110101111111100001110011} \\ \texttt{01100110111111000111011111011110} \\ \texttt{00011110111000111001111100011100} \\ \texttt{01111101111111110000111110111011 1} \\ \texttt{01010111100111001110011110010111} \\ \texttt{11011010111101011100111010011111} \\ \texttt{11011011110001101011100111010011} \\ \texttt{11001101011110011110001111001101} \\ \texttt{00011110110110111000111011110100} \end{array}$$

Alice has now digitally signed the message "`Alice pays Bob €100,-`". Keep in mind that this signature is entirely depended on the message. Even if a change *one bit* of the message, the correct signature would have to be completely different.

Now, let's say that Bob wants to verify if Alice actually signed this, and not someone who wanted to trick Bob. Let's assume at Alice already told Bob her public key in advance. Bob would now simple use the verify function (3):

$$sign(\texttt{"Alice pays Bob €100,-"}, \begin{array}{l} \texttt{11101011010111101111110111110111} \\ \texttt{10010110111111010111001111011110} \\ \texttt{11011111011110111000011101110110} \\ \texttt{11101011110101111111100001110011} \\ \texttt{01100110111111000111011111011110} \\ \texttt{00011110111000111001111100011100} \\ \texttt{01111101111111110000111110111011 1} \\ \texttt{01010111100111001110011110010111} \\ \texttt{11011010111101011100111010011111} \\ \texttt{11011011110001101011100111010011} \\ \texttt{11001101011110011110001111001101} \\ \texttt{00011110110110111000111011110100} \end{array}, pk_{Alice}) = T$$

As you can see, the function returned True. Meaning it was verified that the message "`Alice pays Bob €100,-`" was indeed signed by Alice. Now, even if we change the message by *one bit*, the output of function 3 will no longer be True, but False.

That giant binary number is the signature. The signature consists of 256 bits. That means there are 2^{256} combinations of possible signatures, only one of which is the correct one[1]. As Grant Sanderson said in his cryptocurrencies video[12]:

[1]We actually do not know with absolute certainty that there is only one signature, but as

"This is a stupidly large number, to call it astronomically large would be giving way too much credit to astronomy."

In the context of cryptocurrencies, this means that we can use digital signatures to allow anyone to proof that they actually want a transaction to be sent.

How function 1, 2, and 3 actually work is way beyond the scope of this essay and would require an entire essay dedicated to these to explain them, so we won't go into them here.

4.5 Ledger

Let's summarize the elements we just discussed:

sha256 A function that takes some data and returns some unique garbage generated from that data.

Proof of work A way that an unthrusted 3rd party can proof they did a certain amount of work.

Digital signatures A way to verify that someone signed something thru an entirely digital manner.

We will now take a look at how we could make our own unofficial currency. Let's say we have a publicly accessible list. Now let's say everyone is allowed to add data to the bottom of the list, but no one is allowed to change anything that is already on the list.

This ledger is used by three people, Alice, Bob, and Charlie. They all have €100,- to start with. If Alice wants to pay bob €25,- she adds to following to the list: `"1. Alice pays Bob €25,-"`. Then she signs it using a digital signature. Now, if Bob wants to know his current balance, he simply takes a piece of paper and writes in starting balance, €100, at the top. And then does the following for every transaction on the list that has Bob as the recipient.

1. He checks if the signature is valid.
2. If true, he checks if the sender had enough money at that moment of time.
3. If true, he writes the amount on his piece of paper.

Now he adds all of the money that is written down on the notepad, subtracts all of the money he has spent, and he knows his balance.

But then how does he check if the sender had enough money at that moment in time? Simple, he repeats the steps above for that person, but only counts the transaction below the transaction he was checking in the first place.

This may seem like a lot of work, but with a little optimization, a computer can it do stupid fast.

Putting this together, a ledger could look like this:

far as we know, no one has ever found another signature that would also cause the function to return true in this case.

`"1. Alice pays Bob €25,-"`	Valid signature	*Valid.*
`"1. Alice pays Bob €25,-"`	Valid signature	*Invalid.*
		(Transaction 1 is already used.)
`"2. Charlie pays Alice €200,-"`	Invalid signature	*Invalid.*
		(Signature is invalid.)
`"3. Bob pays Alice €125,-"`	Valid signature	*Valid.*
`"4. Alice pays Bob €250,-"`	Valid signature	*Invalid.*
		(Alice does not have the necessary funds.)

t

The impotent thing to note here is that anyone can put data on the ledger, invalid or valid. And everyone in continuously checking if the data that was added to the ledger is valid before accepting it.

4.6 The Blockchain

So here is the final idea that brings this all together into cryptocurrencies, which is one of the possible implementations of a blockchain.

The blockchain solves a very specific problem: *How do you make many unthrusted parties agree on a piece of data without a central authority?* From the context of cryptocurrencies: *How do you make everyone agree on the global list of transactions, without using a central authority?*

What we will do is to make everyone store it's own copy of the block-chain. This block-chain looks like figure 1. Take a look at it and see if you can make sense out of it.

Each block contains a couple of values. The id of this block, the data stored in the block (in our case a list of transactions), a nonce that we will come back to later, the $sha256$ hash of the previous block, and the $sha256$ hash of the current block.

The important thing to note here is that a lot of work was done to create this block-chain. You can verify this yourself because this block-chain has a proof-of-work embedded in it. The hash that can be seen at the bottom of each block is created by adding all of the values above it together and running the $sha256$ function on it. If you look at the hash you will a number that, when written down in binary, starts with 40 zeros. This is because, as described in section 4.3, I instructed my computer to try out different values of the nonce until the hash at the bottom started with 40 zeros. That took a lot of work. You can easily verify that I indeed did a lot of work by running $sha256(blockId + data + nonce + prevHash)$, and observing that the outcome indeed starts with 40 zeros.

The key part is, you did not do that work, you simply got a block from someone else you could easily verify that it indeed took a lot of work.

In the blockchain, everyone can create a block. When someone makes a transaction, they broadcast it to everyone in the network. Some people in the network are block creators, or more commonly called, miners. What they do is they listen for incoming transaction, check the validity, and create a block out of it. In order to create a block, they first need to do a lot of work to calculate

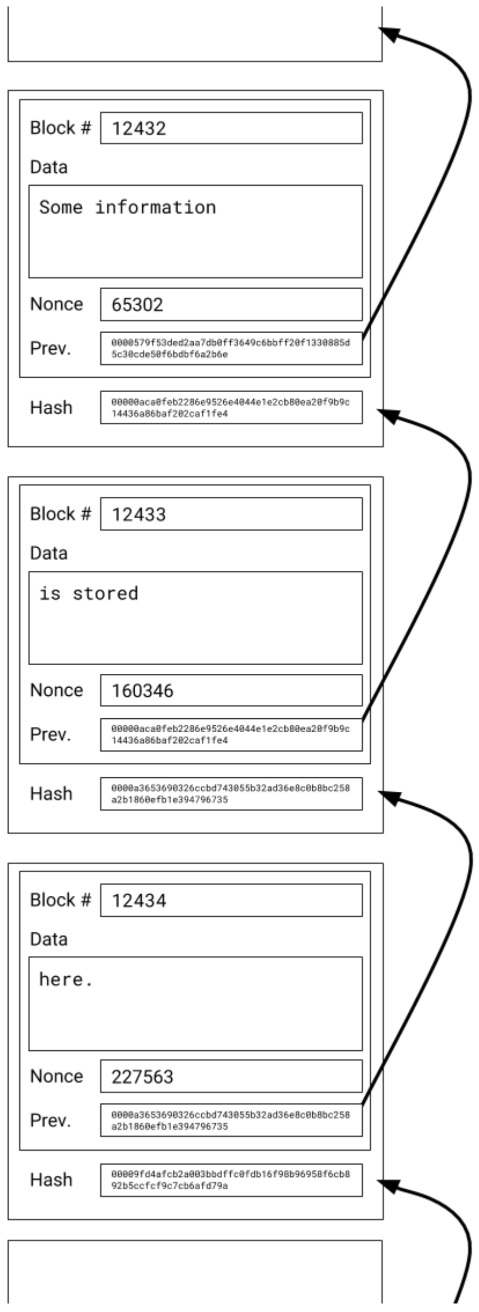
Block # 12432
Data
Some information
Nonce 65302
Prev. 0000579f53ded2aa7db0ff3649c6bbff20f1330885d5c30cde50f6bdbf6a2b6e
Hash 00000aca0feb2286e9526e4044e1e2cb80ea20f9b9c14436a86baf202caf1fe4
Block # 12433
Data
is stored
Nonce 160346
Prev. 00000aca0feb2286e9526e4044e1e2cb80ea20f9b9c14436a86baf202caf1fe4
Hash 0000a3653690326ccbd743055b32ad36e8c0b8bc258a2b1860efb1e394796735
Block # 12434
Data
here.
Nonce 227563
Prev. 0000a3653690326ccbd743055b32ad36e8c0b8bc258a2b1860efb1e394796735
Hash 00009fd4afcb2a003bbdffc0fdb16f98b96958f6cb892b5ccfcf9c7cb6afd79a

Figure 1: Blockchain

the nonce. When that work is done, they broadcast it to everyone, everyone adds it to their personal copy of the blockchain, and the process continues.

This means that from the miners perspective, it becomes a race. A race to be the first one who can create a valid block. But why would they participate in this race? In the case of cryptocurrencies, it's because the miner is allowed to add one special transaction to the top of the list. This transaction does not have a sender nor a signature but just states that the miner receives a predefined number of money in that currency (for example, 12 BTC).

But since this is a P2P network, we cain't just say that the first one wins, because not everyone receives the block at the same time. Then how do you handle conflicts? Let's say you have to follow blockchain (4), which block should to trust? The golden rule is to *trust the chain with the most work in it.* In this case, you would trust the chain containing block K. But let's say they are both the same length, what should you do? The answer is to wait. Wait until one is longer.

$$\cdots \leftarrow A \leftarrow B \leftarrow C \leftarrow D \leftarrow E \leftarrow F \leftarrow G \leftarrow H \leftarrow I \begin{array}{l} \leftarrow J_1 \\ \leftarrow J_2 \leftarrow K \end{array} \tag{4}$$

This does not only quickly resolve accidental conflicts, but also malisues conflicts. For this, we will give an example using Dogecoin[6]. Let's say Bob wants to fool Alice into thinking Bob send Alice 500 DOGE, without actually sending the 500 DOGE. First Bob needs to create a fake block that contains the transaction that didn't really happen before or around the same time that the real block creator did. Bob then sends the block to Alice and only Alice. In blockchain 4 that would be block J_1. Now, since the rest of the network doesn't know about the transaction, the network will continue to work on block J_2. Now one block creator in the network will have created block K, it will be sent to everyone including Alice. The change that Bob was faster than the rest of this network in generating K is already unbelievably small. And even if Bob would be able to generate a fake K, the amount of computing power Bob would need to successfully keep fooling Alice is at least 50% of the *entire* network. And this all was assuming that Alice did not just forward block J_1 to Alice's pears. If Alice did that, block J_1 would have a chance of becoming the real blockchain, meaning that Alice now actually owns the $500 DOGE$ that Bob wanted to fool Alice into owning.

Not that this means that branches in the blockchain are quite common, but will be resolved very quickly be the network if you just wait for the one with the most trust. This means that you should **not** trust a block as soon as you hear it, but instead to should wait for a few blocks to be appended to that block before you trust it.

There are two things to note, the first note is that the block creators are usually referred to as miners in mainstream media. The second note is that in a lot of cryptocurrency, new money is created when creating a block by placing one special transaction in each block that gives the block creator a certain amount of money out of thin air. This is not used in all cryptocurrencies.

This concludes the introduction to cryptocurrencies, in the next chapter, we will look the differences between mainstream cryptocurrencies.

5 Field Research: Comparison of Cryptocurrencies

We looked at 5 cryptocurrancies as potention cryptocurrancies for our aplication. Bitcoin[9], Ethereum[3], Ripple[10], Zcash[2], and IOTA[4].

5.1 General comparison

	Disk usage (block-chain size)			
Currency	**Full**	**Normal**	**Light**	**Min. Memory**
Bitcoin	145 GB[8]	145 GB[8]	5 GB[8]	256 MB[8]
Ethereum (geth)	385 GB[13]	25 GB[13]	0,005 GB[13]	$<$ 1 GB*
Ripple (rippled)	N/A	50GB SSD[11]	-	4 GB
Zcash	420 GB[7]	10 GB[1]	-	4 GB[7]
IOTA	*Unpublished, but advertised to IOT.*			

*: Personal testing

But these cryptocurrencies are popular for a reason, so let's see give a quick description of what makes them special.

Bitcoin is the first ever cryptocurrency.

Ethereum tries to do everything.

Ripple advertises to banks, not to consumers. Ripple can be used by a bank to send money abroad. Ripple does not use the blockchain system described in section 4

Zcash solves the problem that everything is public, and allows people to hide the amount, sender and recipient using a technology called zk-SNARKs.

IOTA focuses on allowing itself to be used by the IOT. It has features such as fee-less transactions, data transfer, and very little disk usage requirements.

On paper IOTA looks like the best use case for our project, so we went on with that.

5.2 IOTA

IOTA looked really promising, it was designed from the standpoint of allowing very cheap devices to communicate any pay each other using the blockchain. It does not use the blockchain but instead uses what it calls a Tangle[14]. The really attractive part of IOTA was that it doesn't have transfer fees. The system that is used instead requires everyone to verify two other transactions and create a proof of work for it. But when we finally had IOTA working, we found out that the technology at its current state just wasn't ready. In our testing, a transaction took at best 2 minutes and at worst more than 24 hours. The system was way too unreliable for our needs.

5.3 Ethereum

So, we went looking another cryptocurrency that could do the job. After some searching we found that that Ethereum had a special (but experimental) `--light` option[5]. Unlike IOTA, Ethereum also has a couple of test networks that allow us to test our product without using any real money. After we finally got that working we saw that it was very quick. When making a transaction from our Ethereum address to another Ethereum address, our code detected an the other Ethereum account received the funds in about 16 seconds.

6 Painpoints during the Process

6.1 Hardware

- The first problem was to attach the omniwheels to the servo's, we ended up with a pre-build kit that has special mounting hardware.
- We wanted to use old 18650 battery cells from an old laptop. But it turns out that soldering to these batteries is quite hard with an 80 Watt soldering iron. To solve this we used a 3.3 V LiPo battery from another laptop and increasing the voltage with a simple converter
- We wanted to attach the Arduino, servos, sensors and the raspberry to the same 5V rail, but we didn't realize that the L7805CV can only handle 0.5 amps. So we ended up using another power bank only for the raspberry. Unfortunately, it was too late and the L7805CV burned out. We replaced it and added a heat-sink for further protection.
- We had the problem that one of the three motors wasn't working we solved this by replacing the wire that came with the servo.
- We also found out that you can bend a wire only so many times. We ended up hot-gluing the wires in place.
- We also destroyed an Arduino Uno during the process, we don't know why. Luckily we cloud get a replacement.
- The voltage booster ended up not working as he was supposed to, we fixed this by replacing the conductor on the board. It is always a good idea to check for ripple on the output.

6.2 Software

We acquired a raspberry pi with a cheap Chinese screen. At first, we couldn't get the screen working. The website provided an install tool that required a very specific version of the Linux kernel installed. We had a hard time following the instructions because of the terrible English that they used. Eventually, we found out that one eBay seller figured it out themselves and provided raspberry pi images with the drivers pre-installed. We do not know if the images clean and not infected with malware, but it was basically our only option so we went with it.

At first, we attempted to get IOTA working. I'm not talking about the code here, at this stage we just tried to get IOTA working using the provided GUI. The first problem was actually getting some IOTA to spend. After some looking around we found out that they had a chat room on slack when people could ask questions. It even had a specific room for asking for ”donations”. We asked for a donation in the evening and the next day we saw that we had been given 0,1 IOTA or about €0,40. We tested a single transaction and it took about 8 minutes to complete. Slower than expected, but still acceptable.

Now we started writing some code that interacted with the IOTA 'blockchain'. The documentation was really hard to get thou, it was riddled with outdated information and invalid server URLs. But we finally got it working and saw that and did a few more test, making sure our program would print all the information we would probably need if it didn't work. What we basically did was repeatedly queering the local IOTA client that was connected to the IOTA network for our current balance. After a few tries, we finally got it working. But then something strange happened, we send money to the test account but the test account didn't relieve it. We waited 24 hours and the money was still not received.

So we started to look at Ethereum. The Ethereum GUI client was relatively easy to setup using the test network. We requested some money to test with using their online tool and started making some transactions. We found that this system was way more reliable so we continued to writing some code for it.

We quickly figured out that using their proved API we couldn't just query for a transaction history of a specific address. So we looked for an alternative. We settled on the following solution: Listing for newly created blocks and then checking if that block contains a transaction that has our address as the recipient. This works but the downside is that we can't check if a transaction happened after the fact.

Now we needed some GUI for the user to interact with. The robot has a screen that we can display information on, but we cannot use buttons since the touch screen is broken. We chose to display a QR code counting the address and some money that we want to have transferred. When we receive the transaction we change the display so it says thank you.

Now we needed to get the code working on the robot itself. Because Noah didn't want to go to Joppe's house every time he wants to make a small change we decided to add an SSH connection. SSH allows computers to send commands to other commuters over the internet. Noah already had experience in setting up secure SSH connections so it wasn't hard to get it working. We also wanted to see the screen of the robot so we set up a VNC. VNC allows us to interact with the GUI of the Raspberry Pi (or any computer that has VNC) over the network.

Now Noah can work on the Raspberry Pi from his home. First, we first needed to get Ethereum (geth) working. There are no official builds for the ARM processors that the raspberry pi used so we compiled it our self. In the process of compiling it, we needed to do a full OS upgrade. All went well and we copied to the Raspberry Pi and it worked.

Or so we thought. Later we discovered that in the process of upgrading the OS the drivers of the screen stopped working. First, we tried to downgrade the kernel, but we failed to figure out how to do that on a raspberry pi. Then we tried to overwrite the boot partition with the boot partition that we manually extracted from the image that had the drivers pre-installed, but without success. So eventually we overwrote the Raspberry Pi entirely with the image, meaning that we had to start over with setting up the Pi. We created an SSH connection and a VNC connection.

But now came the problem, we couldn't compile the code without a system upgrade. Eventually, we used a technology called Docker. In essence, docker allows you to have an OS inside another OS with minimal performance hits. So we used docker and ran into another problem, the 8 GB SD card in the raspberry pi was too small. So, I put the SD card in my laptop, created a backup image. Then I put a larger SD card in my laptop and restored the backup image. It worked, I was able to finish the installation of Ethereum and our code worked as well.

6.3 Further Information

All of our code is available at `https://github.com/nloomans/pws-robot`. This also includes an overview of how our robot works. Feel free to use it for your own projects.

7 Discussion

Our hypothesis was mostly right, but we didn't end up using IOTA. While IOTA did fulfill on its promise of low resource requirements, it simply was too unreliable for our needs. Ethereum unexpectedly became the one we ended up using and we can quite confidently say that, at least for our needs, it is ready to be used on low-end machines like the Raspberry Pi.

8 Conclusion

So in conclusion, this was an incredibly fun project. For Noah, it was his second larges project he ever worked on. We both learned a lot about how cryptocurrencies work.

So to answer our research question; **Yes, cryptocurrencies are ready to be interacted with on a light-weight computer.** To be specific Ethereum is ready. It should be noted that we only tested simple listening from transactions in the blockchain. But it seems likely that sending transactions would also be easy. Also note that this definitely does not include mining, because why it is technically possible, you are very unlikely to ever be the first one to create a block, if at all.

The is also a presentation that goes hand in hand with this essay, we would highly recommend attending.

References

NOTE: We do not use APA, instead we use default biblatex style. This is because it allows for some easy automation and makes the references in the document clickable.

[1] *1.0 User Guide*. URL: https://github.com/zcash/zcash/wiki/1.0-User-Guide.

[2] Zerocoin Electric Coin Company. *Official Zcash website*. URL: https://z.cash/.

[3] Ethereum Foundation. *Official Ethereum website*. URL: https://www.ethereum.org/.

[4] IOTA Foundation. *Offical IOTA Website*. URL: https://iota.org/.

[5] *Light Clients and Proof of Stake*. URL: https://blog.ethereum.org/2015/01/10/light-clients-proof-stake/.

[6] Jackson Palmer and Shibetoshi Nakamoto. *Dogecoin Official Website*. URL: http://dogecoin.com/.

[7] Paige Peterson. *User Expectations at Sprout Pt. 2: Software Usability and Hardware Requirements*. URL: https://z.cash/blog/software-usability-and-hardware-requirements.html.

[8] Bitcoin Project. *Bitcoin System Requirements*. URL: https://bitcoin.org/en/bitcoin-core/features/requirements.

[9] Bitcoin Project. *Official-ish Bitcoin website*. URL: https://bitcoin.org/.

[10] Ripple. *Official Ripple website*. URL: https://ripple.com/.

[11] Ripple. *Operating rippled Servers*. URL: https://ripple.com/build/rippled-setup/.

[12] Grant Sanderson. *Ever wonder how Bitcoin (and other cryptocurrencies) actually work?* URL: https://www.youtube.com/watch?v=bBC-nXj3Ng4.

[13] Afri Schoedon. *The Ethereum-blockchain size will not exceed 1TB anytime soon.* URL: https://dev.to/5chdn/the-ethereum-blockchain-size-will-not-exceed-1tb-anytime-soon-58a.

[14] *The Tangle*. URL: https://iota.org/IOTA_Whitepaper.pdf.