Protecting privacy by mirroring nature

T.S. Jaspers Focks
W. Nguyen
H. van Veltom
V. Wigmore

Protecting privacy by mirroring nature

by

T.S. Jaspers Focks
W. Nguyen
H. van Veltom
V. Wigmore

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be presented publicly on Monday July 4, 2018.

Project duration:	April 23, 2018 – June 25, 2018	
Thesis committee:	H. Wang	TU Delft, Bachelor Project Coordinator
	Dr. Ir. J. Pouwelse	TU Delft, coach
	M. de Vos	Tribler, client

Preface

Abstract

Contents

CHAPTER 1

Introduction

In recent years, privacy on the Internet is becoming a more and more concerning issue. Governmental agencies are more frequently asking ISPs for data about their customers. And even without permission they are still allowed to gather data due to the introduction of several new laws. [1] Because they are able to look at our data, we are vulnerable to the potential abuse of it. Furthermore, sharing information and content is being threatened by events such as the repeal of Net Neutrality rules [2] in the United States and Europe's new Article 13 [3].

Article 13 claims to protect digital content by mandating that all content uploaded to the internet to be monitored and filtered by machines. When the content is recognised by the filter as copyright infringing, the content is deleted and the uploader or hoster of said content may be held liable for these infringements. While this seems at first glance to be a reasonable reshape of the copyright law, the worrying aspect of this development is that digital content that fall under *Fair Use* will most likely be prohibited by Article 13. Briefly, a Fair Use is the act of using copyrighted material in a way that is transformative, the copyrighted material is often used for the purpose of commentary and criticism or parody [4]. For example, Youtube videos that are created for the purpose of reviewing movies or music generally fall under fair use. With Article 13, sharing links to websites or articles, using images for academic purposes or quoting articles will become next to impossible.

These laws put the power of the internet in the hands of internet service providers (ISPs) and corporations, restricting many forms of freedom on the internet. This could result in thoughts, opinions and information being censored and monitored on the internet.

As the future for digital content sharing is looking bleak, the need for alternative ways of sharing content is increasing. Since the early days of the internet, peer-to-peer (P2P) sharing services have been the 'underground' way of sharing content over the internet. In P2P networks, computers are connected to each other without a central node. This way, there is no point through which all data flows. Whenever a node in such a network goes offline, the other nodes can still provide the requested data as they have it too. Because of this, P2P networks are resilient. However, with current P2P networks privacy is nonexistent as it is clear from who to whom data is flowing.

To protect one's anonymity, virtual private networks (VPNs) can be used. VPNs provide a way to encrypt internet traffic between the client and the internet by routing all the traffic through a VPN server. However, VPN providers have different logging policies, and may reveal the client's identity when asked by the government. Another way of protecting one's anonymity is by using The Onion Router (Tor). Tor works by routing the client's internet traffic through a number of randomly selected nodes/computers in the Tor network. The path chosen to route traffic is referred to as a relay circuit. These relay circuits can change after some amount of time and the link between two nodes are encrypted (with the exception of the exit node). The more relay nodes the client has to 'hop' through, the more secure the connection.

The final node is referred to as an exit node, these node connect the client to the server or in the case of **Fig. 1.1**, Alice to Jane. The traffic between an exit node and the server is not encrypted (apart from HTTPS). Because the traffic is relayed through other nodes and eventually through the exit node, the connection seems to be coming from the exit node, instead of the client. This implementation makes it very difficult to decipher the source and destination of a message. All nodes are (assumingly) run by volunteers. While there are a great number of relay nodes, the amount of exit nodes is limited **Fig. 1.2**. This is because the activity of Tor users appear to originate from the exit node, this also

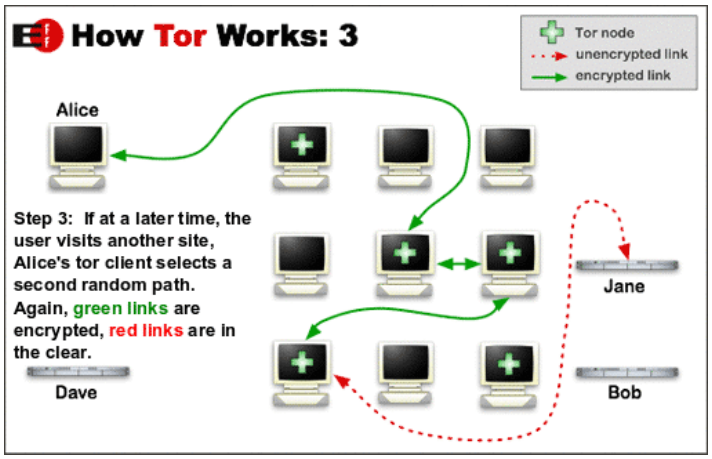


Figure 1.1: A simplified overview of how Tor works. *Source: https://www.eff.org/document/2013-10-04-guard-iat-tor*

includes illegal activity. For a robust network, the amount of exit nodes need to be high. The most well-known implementation of the Tor protocol is the Tor browser.

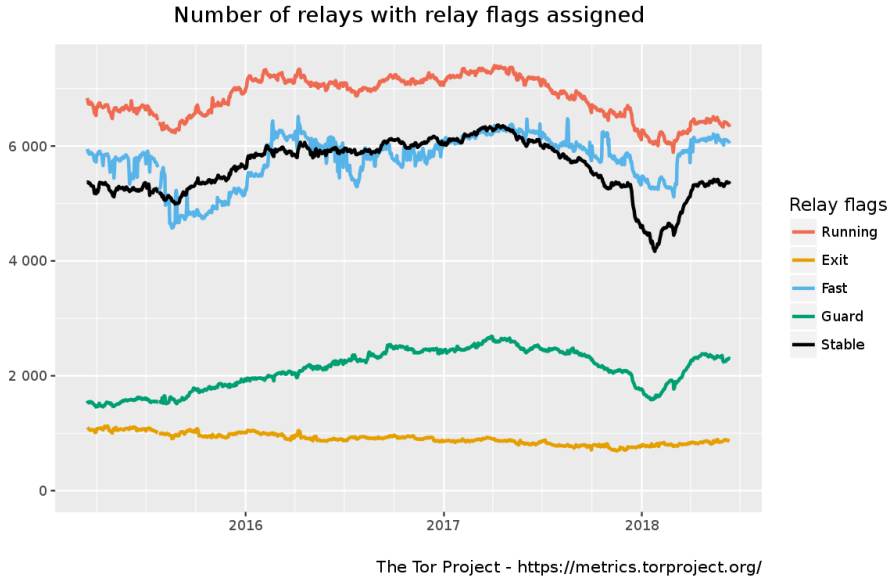


Figure 1.2: The number of relay servers reported by The Tor Project, the number of exit nodes are low in comparison to other relay nodes due to the increased risk.

With the goal of anonymous decentralised file-sharing in mind, Tribler[5] was developed, an open source P2P file sharing program developed as a research project at Delft University of Technology. Tribler combines the Bittorrent protocol and a Tor-like network and thus, implements proxy layers to add privacy for both the downloader and the uploader. With a single proxy layer there is still a risk of the proxy being corrupt and listening to the data sent. For this reason, Tribler has three proxy layers between the user and the rest of the P2P network, similar to a Tor network.
For the user to be connected to the Tribler network, an exit node is needed. Such a node is the connection between the user and the anonymous Tribler network. Because this node is publicly

visible, it damages the privacy of the person functioning as an exit node. Because of this, there is a lack of exit nodes in the Tribler network as not many people are willing to function as an exit node. This project focuses on a network of autonomous self replicating exit nodes. Buying VPSes and using them as an exit node would create such a network, but this requires money as these VPSes have to be bought. The system to be implemented earns Tribler tokens by functioning as an exit node, these tokens have a theoretical value as they give the owner certain benefits within Tribler. The obtained tokens are then to be sold for Bitcoin with which VPSes can be bought.

CHAPTER 2

Prior work

This chapter is about the work done in previous projects as well as explaining the concepts of how a network of autonomous self replicating exit nodes should function. At the beginning of the project, we were given access to the latest version of the botnet and thus the first step of the project was to understand the working and to asses the functionality of the existing system as well as the dependencies of the botnet namely the Bitcoin wallet library Electrum[6] and Tribler.
But first, a short summary of how the botnet started. The concept of an autonomous self-maintaining network started with the development of TENNET[7] by the first bachelor group. After this, a second bachelor group[8] started over from scratch, using the experience of the first group and developed two modules called PlebNet and Cloudomate. After the second bachelor group finished working on PlebNet and Cloudomate, a group of master students [?] continued working on Cloudomate. Cloudomate was implemented to select a VPS provider from a selection of providers who accept Bitcoin, register an account at that provider, and buy a server with the given account. PlebNet was responsible for installing Tribler, setting itself up as an exit node, earn money, activating Cloudomate and ultimately installing itself again on the newly acquired servers.
This brings the number of components in PlebNet down to three: Cloudomate, Tribler and Electrum. Both Cloudomate and Tribler will be explained in this chapter after which the working of PlebNet will be explained. As Electrum is just an implementation of an Bitcoin wallet we do not consider it a key component and will not be explained in full detail in this chapter.

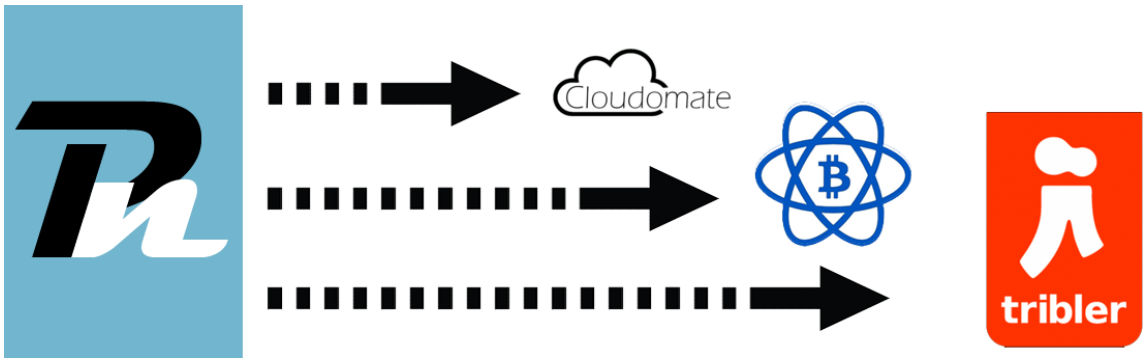


Figure 2.1: The different components that PlebNet uses.

2.1 Cloudomate

As mentioned in the intro Cloudomate is a program that is used to automate the selecting and buying of VPS servers as well as VPN protection. It allows users to buy VPS/VPN services through the commandline as well as allowing programs such as PlebNet to buy servers. The main usage of Cloudomate for PlebNet is buying VPS servers. Cloudomate provides a list of options where PlebNet can choose from PlebNet selects one option and provides the funding. Cloudomate buys the VPS

server and gives it to PlebNet. PlebNet than uses the VPS server to install and create a new agent.

Currently Cloudomate only offers the option to buy VPS servers. A VPS server works by having multiple accounts on the same server. However each of these accounts gets a dedicated portion of the resources. These resources being things such as memory, bandwidth etc. PlebNet needs relatively little resources to be able to run so VPS is an affordable option and because of the guaranteed resources sharing the server with other uses is not a problem for PlebNet. While it would be possible for Cloudomate to buy other types of servers these being dedicated hosting and shared hosting these weren't implemented for the following reasons. dedicated hosting servers are servers where one account gets access to the entire server without having to share the resources with other users. This is great however dedicated hosting is much more expensive than VPS servers and because PlebNet doesn't need many resources dedicated hosting does not offer an improvement over VPS. Shared hosting is similar to VPS in the sense that multiple users have to share the same server unlike VPS shared hosting offers no guarantee of resources to any of the accounts. This means that the resources of the bought server could be taken by other accounts meaning that PlebNet would not be able to run and earn tokens. So for that reason despite overall being cheaper than VPS server shared hosting was not implemented in Cloudomate. There are six VPS providers implemented currently in Cloudomate these providers are: Linevast, CCIHosting, BlueAngelHost, UndergroundPrivate, PulseServer and CrownCloud

A secondary use of Cloudomate besides buying VPS servers is buying VPN protection. These VPNs would be used to hide the IP address of the server when it is being run as exit node for PlebNet. This is useful because many VPS provider don't allow there servers to be used to distribute copyrighted content. And if a VPS provider monitors its servers traffic which many do they will see that PlebNet is using their server to run as exit node for Tribler. And this could lead to the provider banning the server. A way to prevent this from happening is by buying a VPN through Cloudomate and let PlebNet use that VPN to hide its traffic. Currently only one VPN provider called AzireVPN is implemented in Cloudomate.

Cloudomate works by crawling the webpages of VPS and VPN providers for options and filling out the HTML forms on the purchase page. There are some problems with this approach the main concern is that providers may change their page layouts, resulting in non-functionality of Cloudomate and therefore PlebNet. However currently it is the only way because no provider is offering an alternative way to buy. They could offer an alternative for example an **API** that could be used to buy server however it is unlikely this would happen because this API would allow possible harmful bots from purchasing their services. So currently this is the only way Cloudomate is able to buy a VPS/VPN service in an automated way.

Currently when Cloudomate buys a VPS server it only leases it for a month. This is the shortest period the majority of VPS providers allow a server to be leased for. Cloudomate does not offer a way to lease servers for longer then a month nor does it allow for a lease to be extended. This inability to extent a lease is not a problem for PlebNet. Because for PlebNet there is little difference between an agent extending its own server's lease and an agent buying a new server where a new PlebNet agent can run.

When Cloudomate buys a VPS/VPN service it has to fill in user information. This user information can either be real and entered by the user but Cloudomate can also randomly generate a user identity. The VPS providers that are offered by Cloudomate are shown in **Fig. 2.2**. The options that are purchasable from these providers can be queried and shown in **Fig. 2.3** the image shows the options for CCIHosting.

```
Providers:
   linevast       https://linevast.de/
   CCIHosting     https://www.ccihosting.com/
   BlueAngelHost  https://www.blueangelhost.com/
   UndergroundPrivatehttps://undergroundprivate.com
   PulseServers   https://pulseservers.com/
   CrownCloud     https://crowncloud.net/
```

Figure 2.2: VPS providers

Options for CCIHosting:

#	Name	Cores	Memory (GB)	Storage (GB)	Bandwidth	Connection (Gbit/s)	Est. Price (mBTC)	Price (USD)
0	Level 1	1	1.0	40.0	Unlimited	0.01	1.87	15.0
1	Level 2	2	4.0	60.0	Unlimited	0.01	2.6	21.0
2	Level 3	2	4.0	80.0	Unlimited	0.01	4.06	33.0
3	Level 4	4	4.0	100.0	Unlimited	0.01	4.79	39.0
4	Level 5	4	8.0	120.0	Unlimited	0.01	6.61	54.0
5	Level 6	5	10.0	140.0	Unlimited	0.01	8.8	72.0
6	Level 7	5	12.0	160.0	Unlimited	0.01	10.26	84.0
7	Level 8	6	14.0	180.0	Unlimited	0.01	11.72	96.0
8	Level 9	6	16.0	200.0	Unlimited	0.01	12.45	102.0
9	Level 10	8	16.0	200.0	Unlimited	0.01	14.64	120.0

Figure 2.3: CCIHosting's options

2.1.1 Gateways

When ordering a VPS the transaction has to go through a payment gateway. A payment gateway is an application that providers use too authorise credit card or direct payments with the provider a famous example of a gateway would be paypal. Cloudomate is only able to make transactions with Bitcoin so it exclusively interacts with bitcoin gateway of the providers. The different gateways used by each provider were: Bitpay this gateway is used by Linevast, BlueAngelHost and CrownCloud. CCIHosting works with the Coinbase gateway in Clouodmate however they currently use the Coinpayments gateway. Similarly UndergroundPrivate is implemented in Cloudomate as working with Blockchainv2 gateway however they changed this to the Spectrocoin gateway. And while PulseServer is implemented as working with Coinbase in Cloudomate during the research phase of this project they stopped accepting Bitcoin payments all together. Fortunately later in the project they reimplemented a Bitcoin gateway however this wasn't coinbase anymore but Coinpayments.

2.1.2 Electrum

Electrum is the Bitcoin wallet that was used in Cloudomate to purchase VPS/VPNs. The version that was used was 2.8.3, which was a Python 2.7 version. Because Tribler, PlebNet and Cloudomate were written in Python 2.7, upgrading Electrum was not possible. Electrum 2.8.3 was fortunately fully functional and is being used in this iteration of PlebNet as well. The way the system currently works when PlebNet uses Cloudomate is that PlebNet and Cloudomate both have their own Electrum wallet and Cloudomate only has access to its own wallet. So if PlebNet wants to buy a server it first has to make a transaction from its wallet to the Cloudomate wallet before being able to buy a VPS server.

2.1.3 Positives and negatives of Cloudomate

Now that a general overview of the workings of Cloudomate have been given now the positive and negative aspects of Cloudomate will be discussed. Because this project is about making PlebNet work with Cloudomate the positive and negative aspects are viewed from that perspective. The usability of Cloudomate as a stand alone program is therfore not considered.

Positives of Cloudomate

While PlebNet and Cloudomate have both been created by the same development Cloudomate has been developed on further by more teams. This has resulted in a very good quality of the code. Most of the code has been commented and a good read.me has been provided this meant that it is easy for developer teams to understand the inner workings of Cloudomate. Cloudomate also offers six VPS provider options which is sufficient for PlebNet to use. And Cloudomate now offers VPN providers and these could be implemented to hide the Plebnet agent's traffic.

Negatives of Cloudomate

There are unfortunately some negatives to the current Iteration of Cloudomate. These are threefold. The first negative is that when Cloudomate was further developed on by other teams the interaction with PlebNet was lost and now PlebNet is unable to interact with Cloudomate correctly. The second negative is that while Cloudomate provides many provider all but one of them still work this one being BlueAngelHost. This problem happened because many providers have decided to change their websites, gateways, etc. Having only one working provider results in a vulnerability for PlebNet because if BlueAngelHost decided to change their system Cloudomate would not be able to buy VPS servers anymore. The final problem is the random user generation. There is an error in Cloudomate that while the user generation itself is succesful the resulting data is not saved. This means that when Plebnet buys a VPS server through Cloudomate, Cloudomate will generate random user data and use that to buy the server. However the problem is that this random user data also contains the root password and username which is needed to access the VPS server. And because Cloudomate does not save this data it means that Cloudomate will buy the server but will not have the information necessary to gain access to the server.

2.2 Tribler

This section is about our initial research on Tribler. Tribler is one of the key components of PlebNet, the most important parts within Tribler for our project will be discussed here. As mentioned before, Tribler is the P2P file sharing program developed at Delft University of Technology. For this project, we used an experimental version of Tribler which has access to the still in development Marketplace module. The components discussed are the exit node service for Tribler and the Tribler Marketplace.

2.2.1 Running as an exit node for Tribler

The ultimate goal of the agent is to generate money as an agent needs money to buy new servers and install a copy of itself onto them. The only way an agent can earn this money is by providing specific services for Tribler. Tribler rewards people who run as an exit node by giving them MB tokens. These tokens are a currency within Tribler and should give people benefits in the future, such as faster download speed, they can have a value and can thus be sold for money. Earned tokens can be traded on the Tribler Marketplace which will be discussed in the following section.
Before and agent can start earning tokens in Tribler, Tribler itself has to be installed. When installing Tribler on Linux operating systems the user has to install all needed dependencies in order for Tribler to work correctly. The second part is ensuring that the agent can run Tribler while functioning as an exit node. This is done by setting the value of the option `tunnel_community_exitnode_enabled` to `True`. After this, Tribler is running as an exit node and MBs are being earned. The amount of MBs indicate the amount of trust a user has and is calculated by: $MB_{given} - MB_{taken}$. Where MB_{given} stands for the amount of data distributed across the network in megabyte and MB_{taken} stands for the amount of data collected from the network in megabyte. These values can be seen in **Fig. 2.4** as they are represented in Tribler.

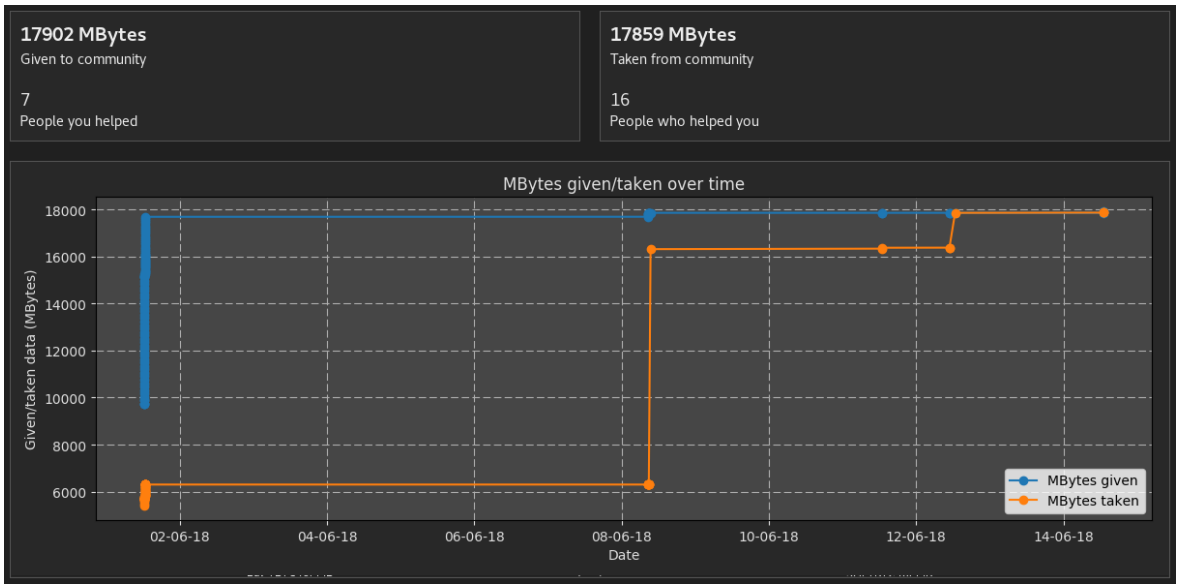


Figure 2.4: Amount of MBs given to and taken from the Tribler community.

2.2.2 The Tribler Marketplace

In order for the agent to be able to buy new servers, tokens (MB) acquired need to be sold for Bitcoins. In Tribler, an experimental version of a market was implemented, allowing users to sell their earned MBs for Bitcoins or buy MBs with Bitcoin. The market could also be tested using Testnet Bitcoins (TBTC), which offers a way to easily test transactions. For now, these are the only currencies supported in Tribler.
There are two ways for an user to buy and sell MBs on the market. Either via the graphical user interface in Tribler or via the web **API** . Both use a system in which bids and asks can be made. Bids are for buying currency while specifying the volume to buy and the price per unit to pay. Asks are for selling currency while specifying the volume to sell and the price per unit to recieve. When Tribler is running, the web API is located at `http://localhost:8085`. The web API contains much more information, but for the purpose of this section, only the market API will be discussed. The commands necessary for the agent to successfully interact with the marketplace to sell its tokens are as follows:

```
curl -X PUT http://localhost:8085/market/bids --data "price=P &quantity=Q
                    &price_type=PT &quantity_type=QT"
curl -X PUT http://localhost:8085/market/asks --data "price=P &quantity=Q
                    &price_type=PT &quantity_type=QT"
```

These PUT requests let the agent make a bid or ask. The price, quantity, price_type and quantity_type are to be filled in with the desired values. In the current implementation of Tribler, all bids and asks have to matched, meaning that a buyer cannot give a seller more for his currency than that he asked for. An bid on Bitcoin for MB has to be matched with an ask for Bitcoin with MB and not with an bid on MB with Bitcoin and vice versa.

```
curl -X GET http://localhost:8085/market/bids
curl -X GET http://localhost:8085/market/asks
```

These GET requests pull all bids and asks on the market. All bids and asks are also visible within Tribler as can be seen in **Fig. 2.5**. With these tools the agent is successfully able to interact with the Tribler marketplace so that it can sell its tokens for Bitcoins to buy new VPSes.

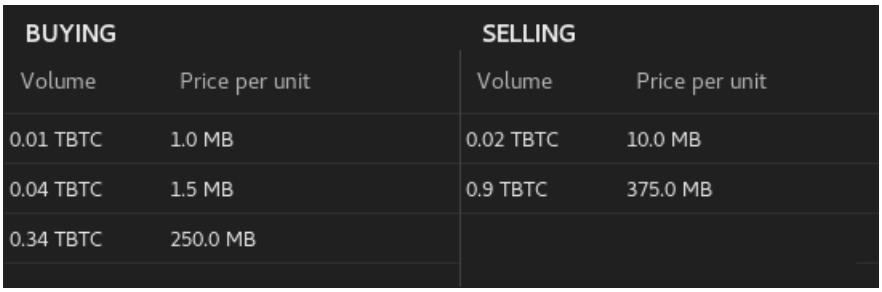


Figure 2.5: Asks and bids on the Tribler Marketplace.

2.2.3 Matchmakers

In order to place bids or asks and to receive updates about the market, special type of peers are needed. These peers are called matchmakers and are the entrance point to the Tribler Marketplace. Each user of Tribler has its own list of matchmakers and they form the network that is the market. When initiating Tribler, it takes various amounts of time in order for the user to be connected to at least one matchmaker, we experienced it to be ranging from a couple of seconds to around fifteen minutes.

```
curl -X GET http://localhost:8085/market/matchmakers
```

With the above written GET request, the agent is able to verify if it has any matchmakers. This is needed as the agent would otherwise get stuck waiting for verification that an ask has been created, meaning that the agent would not continue with his routine of buying and installing servers.

2.3 PlebNet

In the previous sections the key dependencies of PlebNet have been discussed. With these modules PlebNet is able to function as intended. This section will explain in detail how PlebNet functions as well as explain the state of the project as we received it.

2.3.1 Purpose of PlebNet

In order for PlebNet to be a successful network of autonomous self replicating exit nodes PlebNet is responsible for the deployment of as many Tribler exit nodes as possible with on the long-term a stable expanding network of independent agents. The definition of a stable network is that the deployment rate of new agents is larger each iteration, meaning at least one VPS per month per agent should be bought and installed. The steps necessary in order to achieve this are as follows:

- Install Tribler and run as exit node
- Earn money via Tribler
- Use Cloudomate in order to buy VPSes
- Installation of the child server
- Repeat this cycle

PlebNet's source code has two main functions which can be called, `setup` and `check`. At the start of the cycle, the `setup` function is called once initiating all configuration files and creating the Electrum wallet. After this, the `check` function is called systematically every couple of minutes. This function

is responsible for checking if Tribler is running, checking if the agent has enough credits to buy a new server, checking if a server has been bought and is ready to install, and ultimately installing a new instance of PlebNet on the new server and let it run the `setup` and `check` functions. An overview of this behaviour is provided in the communication diagram in **Fig. 2.6**.

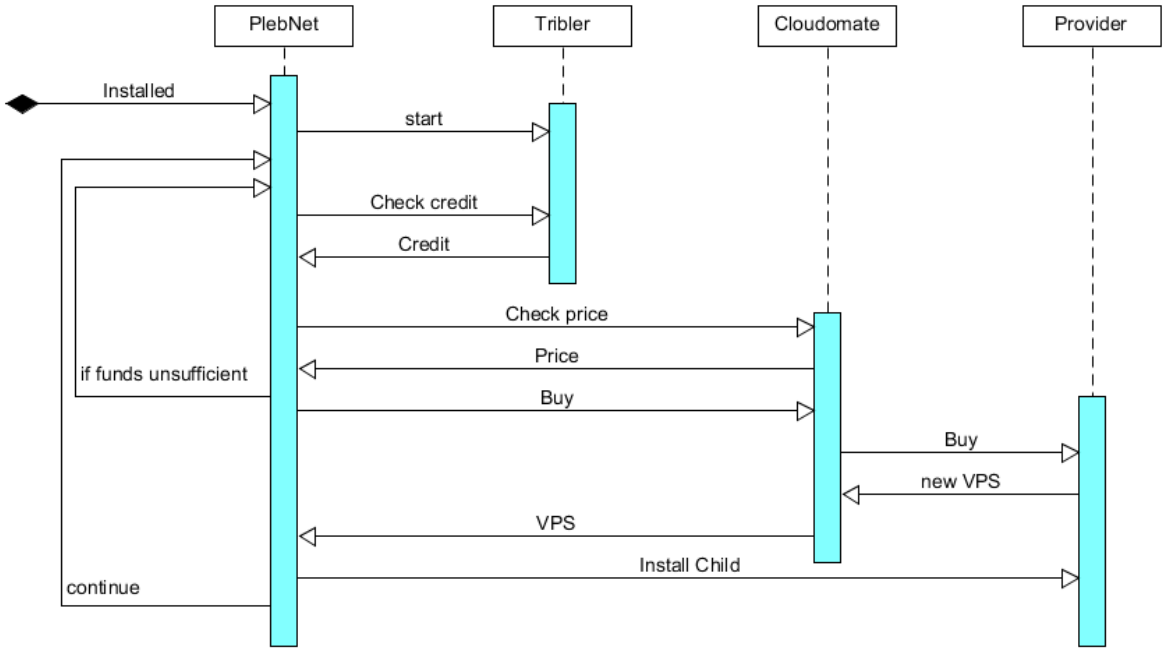


Figure 2.6: Communication diagram of PlebNet

2.3.2 Choosing and installing offspring

As Cloudomate offers PlebNet multiple VPS providers to choose from, PlebNet has to make a smart decision on which server to buy. In the current version of PlebNet, a basic form of DNA was implemented giving PlebNet the ability to develop itself over future generations knowing which servers to buy and which not. Each PlebNet agent has a list of tuples containing available providers and a value representing how good it is. This value is increased when a server is successfully bought and decreased when not. When deciding which server to buy, it chooses a random provider giving providers with a higher score a higher chance of being chosen. After choosing a provider, the agent has to choose between different server specifications. This includes the number of CPU-cores, memory and bandwidth. As these options directly impact the performance of the agent, it has to choose an option which is the most cost-effective. However, for now it chooses the cheapest option.
After having chosen and bought a server, the agent has to install all software. This is done via the `create-child` bash script together with the IP address and root password of the new server. This bash script first creates all needed directories, then it downloads the latest version of the PlebNet project from GitHub and afterwards the `install` bash script is used. This script installs all needed python dependencies and then installs PlebNet. After this is all done, a single call to the `setup` function is made.

```
echo "*/2 * * * * root /usr/local/bin/plebnet check" > /etc/cron.d/plebnet
```

Then the Linux software utility cron is used for letting the agent call the `check` function every two minutes with the above written line of code.

2.3.3 Original state

The received code was not optimal as it was not functioning anymore. There were several errors we encountered while testing the functionality of the initial system. Some methods in PlebNet made calls to methods in Cloudomate which had been removed or changed by the master group working on improving Cloudomate. Each agent created a single configuration file for creating accounts at the VPS providers and as these providers restrict the number of accounts per email to one, each agent could only buy a single server per VPS provider. This meant that after having bought a server, it would decrease the DNA value for that provider as it was unable to buy more servers. The level of documentation and commenting on the code was low, which made it harder to grasp the purpose of different files and understand the relation between different packages. As some methods where over 100 lines long, some comments or documentation would have helped whilst debugging the errors encountered.

CHAPTER 3

Requirements analysis

After studying the related work delivered by the previous groups the next step of the project was to create a list of requirements based on this research. This list would detail all the features that needed to be added to the new version of PlebNet. The list of requirements was created using the MOSCOW method, The MOSCOW method groups all requirements based on their priority together. It recognises four priorities these being: must haves, should haves, could haves, won't haves. The resulting MOSCOW list with an explanation for each item is given below.

3.1 Must haves

Must haves: these are the functionalities which should be implemented for the project to be successful.

Fully operational end-to-end system: Both PlebNet and Cloudomate have been developed by a single group after which Cloudomate has been developed even further resulting in PlebNet not being able to function anymore together with Cloudomate. PlebNet has to be modified in such a way that it can use Cloudomate again. This must result in a system which can earn money, automatically buy servers with the money earned, install itself on these new servers and repeat this cycle. Money is earned by functioning as an exit node for the Tribler network. By functioning as an exit node, credits (called MB) can be earned. These credits must then be sold on the Tribler marketplace for bitcoins. With this money, a new server must be bought via Cloudomate from a list of providers which accept bitcoin. After the server is bought, PlebNet must automatically install the source code of PlebNet and Cloudomate onto the new server.

Autonomous installation: When a new server is bought, PlebNet is responsible for successfully initialising a new PlebNet agent on this server. To achieve this PlebNet has to be able too autonomously install PlebNet, Cloudomate and Tribler as well as all the required dependencies of these three systems on the new server.

Status monitoring: Because PlebNet is a fully autonomous self-replicating system it is nearly impossible for a user to keep oversight of all the PlebNet agents in the system and whether or not these agents are still alive. To give the user this oversight status monitoring has to be implemented. Status monitoring means that all PlebNet agents in the system will send their status to a central receiver which collects and displays them for the user.

successful continuous integration in PlebNet: Continuous integration is the principal of integrating code into a shared repository. This shared repository is then regularly verified by an automated build. This is useful because it allows development teams to easily detect and locate errors in the code. The Continuous integration tool that will be used for PlebNet is Jenkins CI. This tool has been chosen because both Tribler and Cloudomate already use Jenkins CI and because PlebNet is reliant on both these systems using a different continuous integration tool would only lead to more overhead for future development teams.

3.2 Should haves

Should haves: these are the functionalities that are wanted by the client, but the software is usable without their implementation.

improving the project documentation: The current version of the project lacks sufficient documentation and there exists no clear overview of how the project is structured. In order to make it easier for new developers to gain oversight of the project a so called UML diagram should be made explaining the basic function and interaction of classes with each other. On top of the UML diagram the projects read.me (a text file explaining how someone uses PlebNet) should be updated because currently it only contains a single line.

improving code maintainability: The current version of the project should be improved upon in terms of maintainability. Here maintainability is defined as a combination of two things. First is that the current code lacks comments explaining how the functions work and what they should do. This makes it difficult for new developers to gain insight in the inner workings of the code. So it is important to add these comments. The second part is having all external dependencies(calls to systems that are outside of PlebNet) go through one dedicated class. The current version already has these dedicated classes however most other classes still contain external dependencies. This is an issue because if one of these external systems where to make changes to its system all of these classes would have to be updated. If these external dependencies went through a dedicated class only that class would need to be updated making it much easier for PlebNet to adapt to changes in external systems.

Dynamically add new VPS provider to the existing PlebNet VPS options: In the current version of the project, the list of available VPS providers is hard-coded in both PlebNet and Cloudomate. PlebNet should be able to retrieve the list of available VPS providers from Cloudomate. When a child agent is installed, any new providers added to Cloudomate should be automatically added to the VPS options of the child agent.

Installing VPN protection: Because most VPS providers do not accept their services being used for distributing copyrighted material. Therefore there exists a risk that the VPS provider sees what the PlebNet agent is doing and promptly bans the VPS server. To prevent the agent from being banned PlebNet should be able to install VPN protection.

3.3 Could haves

Could haves: these functionalities will only be implemented when there is time available at the end of the project

Transfer obsolete funds: At the end of the life cycle of an agent, the agent will most likely have some money left which was not enough to buy a new server with. Because each agent has their own wallet if an agent reaches the end of their life cycle its left over money will simply be lost. To prevent losing this money, the agent could be able to transfer this money to a central node or to one of its child nodes.

The ability to create an issue on GitHub In order to be able to improve the agents in the long term, it is necessary to be notified about (possible) problems and errors in the code. This can be achieved by a notification to the developers, but a direct and clear way to handle these notifications is by implementing an automated issue creator. When an error occurs, this posts an

issue on GitHub(GitHub being the repository service where the code is stored), including all important information such as provider, trace call-back and other settings.

Genetic Algorithm for reproduction The bot could be programmed to dynamically chose a survival strategy whenever it creates a new child. Possibilities are the decision which VPS to acquire, or the trading strategy on the Tribler market. It could adjust these strategies based on success and failures in agents lifetime. And it could pass on this survival strategy to its child nodes. So over multiple generations agents will develop better survival strategies.

Simulate VPS provider to allow for free end-to-end testing End-to-end testing is very difficult for PlebNet. This is because it has to buy and run on a new VPS instance and buying a new VPS instance for each end-to-end test is expensive. So to solve this a fully controlled environment could be set up. thist allows the developers to simulate different settings and prove the concept of PlebNet. The configurations can be quickly restored and the system improved.

Ability to monitor the status of the online agents live Instead of only being able to receive notifications such as heartbeats from the running agents, it could also be possible to ask for specific information from all online agents, or one in particular. This could be the current DNA configuration, the time until shutdown or the amount of processed data on Tribler. The information should be read only, so that the agents cannot be altered and live independent of central influences.

Package PlebNet with its dependencies for easier installations Currently PlebNet is installed using batch scripts. A batch script is a file of text with commands that are executed sequentially. So PlebNet and all of its dependencies are now downloaded and installed sequentially. This costs time to install. An alternative to this could be used by delivering a pre-installed package which in comparison would result in relatively simple and clean installation of PlebNet and all dependencies.

3.4 Won't haves

Won't haves: these are the functionalities that are outside the scope of this project, however they might be nice to implement later on.

Adding new VPS/VPN services This project focuses on implementing the end-to-end usability of PlebNet and improving or adding new functionalities. The addition of new services would fall under improving Cloudomate. For PlebNet to function it only needs a hand full of available services and at the start of the project these were already implemented. So it is not necessary for the functionality of PlebNet to implement this feature. However it could be done afterwards by new groups that want to improve Cloudomate.

The ability to extend the lease of the current VPS It could be possible to implement the functionality that instead of letting the PlebNet agent buy a new VPS server it could instead choose to extend its own lease(currently all VPS's are leased for one month). However this would require extra work for each VPS option to implement and there is next to no practical difference between a PlebNet agent extending its own VPS license and a PlebNet agent buying a new VPS. So for that reason this feature is considered outside the scope of this project.

Improve Cloudomate for standalone use The functionality of Cloudomate can be improved so that it becomes useful for standalone use outside of PlebNet. However this would not benefit the overall goal of the project which is to make PlebNet a fully operational end-to-end system. for that reason this feature is outside the scope of this project.

CHAPTER 4

System architecture

The first step in creating a fully functional version of PlebNet is to analyse the provided system architecture and determine the structure, as this is the core of the application. In this chapter the old structure is analysed and the modification to the code are explained.

4.1 Old Architecture

As PlebNet lacked proper documentation regarding the code and comments were ommited, it resulted to be quite a task to create an overview. By going through all the code, file by file, it was possible to get a grasp of the original structure. A dependency diagram was created and is shown below.

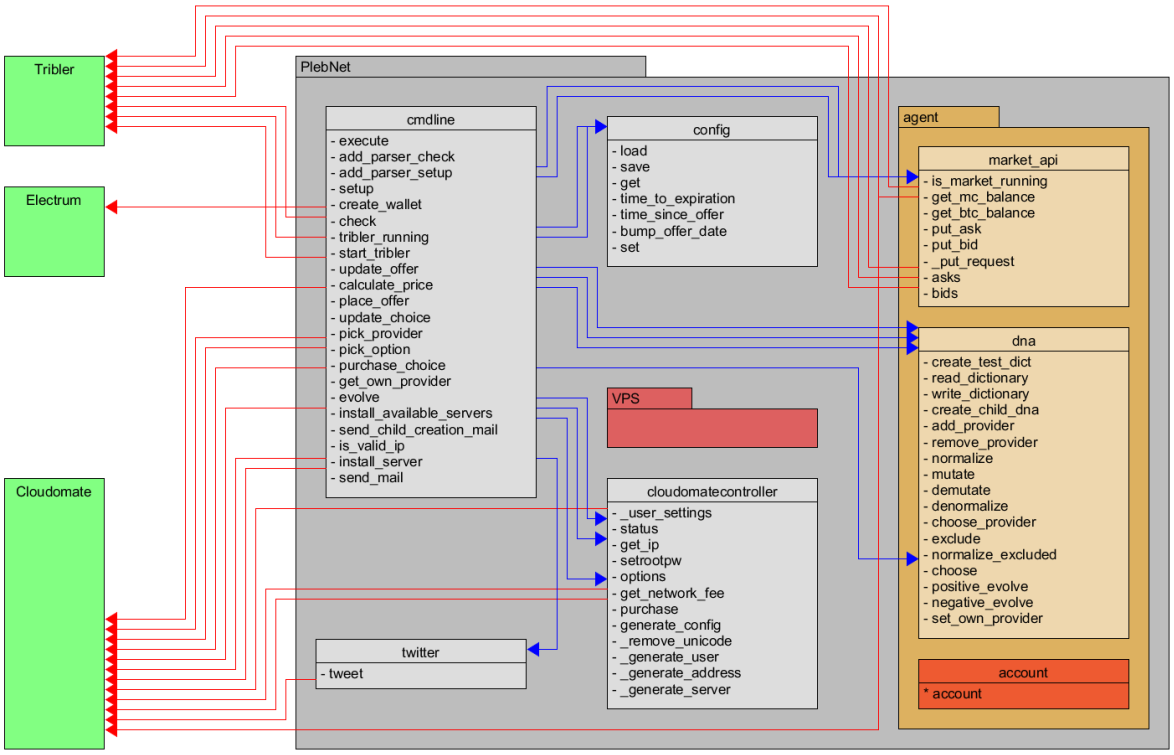


Figure 4.1: Interpackage dependency diagram of the provided code. Blue lines indicate internal dependencies and red lines external. Red files/packages indicate unused code. The amount of arrows between different files indicate the amount of calls made between the files. More lines results in a higher connectivity.

Figure 4.1 shows the different packages and files of PlebNet and their dependencies. The first things to notice regarding the structure are the unused files. These can be removed completely, but were probably left as the overview was lost.

The core python file is the `cmdline.py` file. This file contains 370 line of code (loc) which handles all the calls made to PlebNet and most of it is done internally. This ranges from the complete initialisation of PlebNet to acquiring Bitcoin balance and buying servers. The scope of this file is too broad for a single file and the purpose of the file can only be described as being the complete functionality of PlebNet.

It was also noted that the `cloudomatecontroller` was not used to its full potential. It contained several methods which had nothing to do with Cloudomate, and lacked other methods which were required for PlebNet. This resulted in other classes making direct calls to Cloudomate, bypassing the controller. When Cloudomate is updated and calls are change, this means that multiple PlebNet files are outdated and have to be adapted. If the controller is the only class with direct dependencies to Cloudomate, it is easy to keep PlebNet up-to-date. The same applies to other external dependencies such as Electrum and Tribler, but these lacked a controller in the original code.

The provided structure seems inadequate to use when PlebNet should be easily maintained and/or modified. The modifications which are made will be discussed in the next section.

4.2 New Architecture

While reviewing the code written for PlebNet by the previous group, it was noted that there were some classes which had too many responsibilities as well as some classes which were not used at all by PlebNet. Because of these reasons, It was decided to restructure PlebNet in such a way that every package within PlebNet would have its own responsibility and within these packages the responsibilities per file should be clearly defined as well.

The first part of the refactoring process is to determine the new main structure: The package devision. In the old structure the main folder `PlebNet/` contained 2 packages (`VPS` and `agent`) and multiple separate files. The `agent` package is kept, as its responsibilities are clear: handling PlebNet. The core file `cmdline.py` is kept as well. This file should handle the commandline input. The other files are separated or removed, which resulted in the structure shown below.

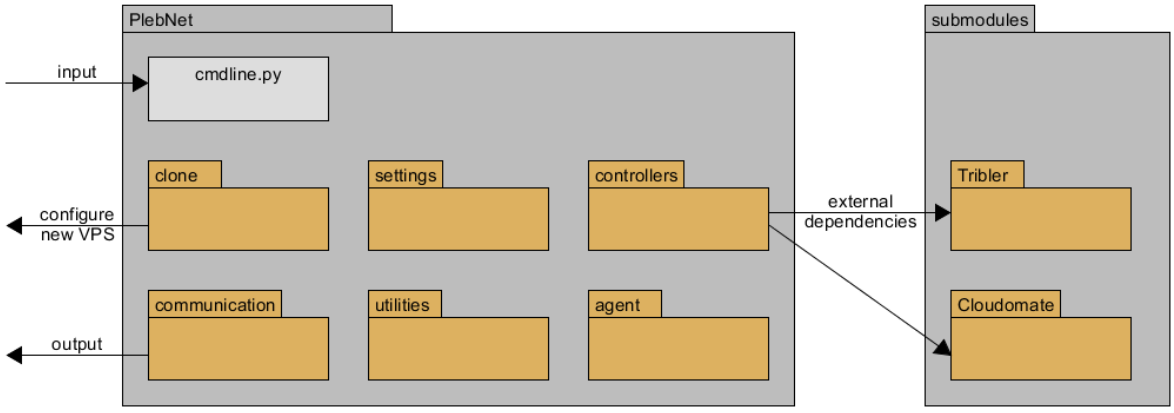


Figure 4.2: Package diagram

The updated structure results in a better understandable dependency diagram as well, as shown below. The `cmdline.py` is still the entry of the entire system, but it is not the core anymore. Its responsibilities are reduced to handling the input from the commandline (while running these are the `plebnet setup` and `plebnet check` calls) and redirect the calls to the proper package/files.

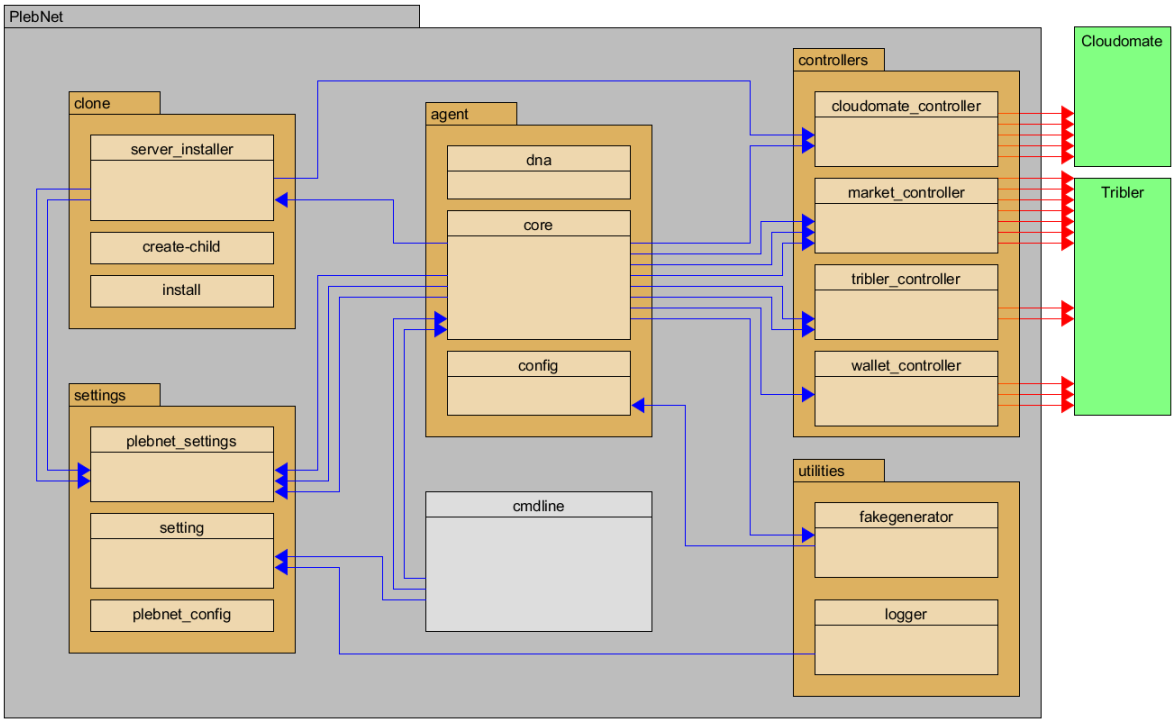


Figure 4.3: Interpackage dependency diagram of the provided code. Blue lines indicate internal dependencies and red lines external. The communication package and the dependencies on the logger are left out for readability as they are not part of the core of PlebNet. The amount of arrows between different files indicate the amount of calls made between the files. More lines results in a higher connectivity.

The structure of the refactored PlebNet can be seen in the package diagram in **Fig. 4.2**. The responsibility of every package is as follows:

The controller package contains the all the code which makes calls to external submodules, in this case Tribler and Cloudomate. This way, whenever something changes in these dependencies, the only code which needs to be updated is located inside the controller package. It was decided to split up the Triblercontroller in to multiple separate controllers, as the file would become to large (>200 lines of code). Each of the new subcontrollers (`market_controller`, `wallet_controller` and `tribler_controller` have their own specific tasks withing the communication to Tribler.

The agent package contains the DNA configuration of the agent as well as the code which makes calls to the various controllers for running Tribler and Cloudomate for buying new services. The responsibility of this package is everything regarding the agent operations.

The clone package is responsible for installing PlebNet on the newly bought servers and initialising this new agent. This package contains multiple bash scrips as these are used to run before python is installed.

The communication package contains modules which can send messages to the real world. For now, the PlebNet agent is able to communicate via IRC as well as create a GitHub issue when an uncaught error occurs. This can be extended to contain modules for visualising the current network. It should be kept in mind that this communication should be read only, as the agents should not be altered by external parties.

The settings package contains all methods which handle the settins of the agent. These settings include the loggin information for GitHub and IRC, and the use of the logger. These settings are inherited from the parent node and can be set from the command line, or in the files in the subdirectory `configuration`

The utilities package contains classes which are project wide used and do not belong to one of the discussed packages above, this includes the logger and a file for project wide configurations.

4.2.1 Lower level architecture

On the file/class level the architecture did not require large refactoring as the actual process of running plebnet can be described as a linear process: check the balance, if sufficient buy a VPS and install. This does not require many different classes, just many lines of order code and methods.

In the provided code the class DNA was used (`plebnet/agent/dna.py`), which makes sense as the dna can be seen as an instance, and while handling the cloning, there can be multiple instances of dna. However, the calls made to this dna were done is such a manner that there were multiple instance of dna at the same time, resulting in problems when the dna was updated, but not written to a file and reloaded before every use. This is solved in the new version of PlebNet by using singletons whenever a class is used. This way every alife instance is the same and updates will handled properly.
singletons usage

CHAPTER 5

Improving PlebNet

In this chapter, PlebNet is being discussed in detail. As mentioned before, PlebNet is responsible for running a Tribler exit node on a VPS, earn 'bandwidth tokens' (MBs) from Tribler and sell these for Bitcoin in order to purchase new VPSes and replicate itself. Although the project is still in its early stages, the long-term goal of PlebNet is to be an autonomous self-replicating organism. A PlebNet agent should be able to make certain choices in regard to its environment, be able to adapt and replicate itself. Because these goals are difficult to achieve, especially within a nine week time frame, the short-term goal is to create a system that could run an exit node on servers and replicate itself. The following sections discuss the different functionalities of PlebNet.

5.1 Initialisation

Initialisation of PlebNet encompasses the installation process of the agent, Tribler, Cloudomate and Electrum. Because Linux VPSes are generally cheaper than Windows instances and offer more flexibility in terms of server setup, PlebNet was designed exclusively for Linux servers. The Ubuntu 16.04 distribution was chosen as the default Linux distribution for development because this is the distribution that most, if not all VPS providers offer. PlebNet is also able to run on Ubuntu 18.04 and Debian 8, and should be able to run on most Debian derivatives with some modifications. The project is largely written in Python2.7, along several bash scripts for configuring servers.

5.1.1 Packaging

One of the challenges of installing PlebNet was that it has to be able to run on different servers. While the Ubuntu distributions are similar, there are still slight differences in the available packages/libraries in the Ubuntu software repositories depending on the distribution version. Additionally, not all dependencies can be installed via Ubuntu's Advanced Packaging Tool (APT) due to the many Python packages needed to run Tribler. Therefore, Python's package manager "Pip" was used to install most Python dependencies.

Snaps

We have researched the possibility of packaging code as Snaps[9], allowing for a more stable way of installing PlebNet in a containerised fashion. As PlebNet now uses over twenty packages, it is vulnerable to changes in these dependencies as a single change can be enough to stop the system from functioning. When using Snaps, the version of the packages at the creation of the snap will be used, resulting in a single package created specifically for PlebNet. Unfortunately, Snaps works by mounting a virtual filesystem which will not work on most VPSes due to restricted access as can be seen in **Fig. 5.1**

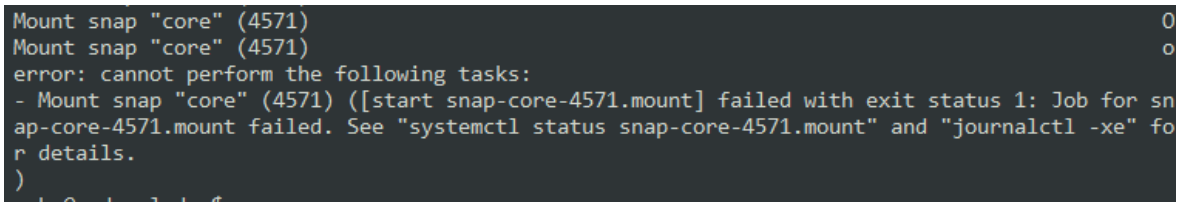


Figure 5.1: The virtual filesystem could not be mounted on a VPS

5.1.2 Wallet creation

The initial version of PlebNet used the Electrum wallet library directly for creating a wallet. After trying to make asks and bids on the Tribler Marketplace, it gave an error that a Bitcoin wallet had to be created first. After some research, it became apparent that the wallet was not linked to Tribler and Tribler was unable to use it. Because we could not figure out how to link them, we decided to use the functions within Tribler for creating new wallets. This also solved the issue of having Electrum as a direct dependency for both PlebNet and Tribler, as Tribler also uses Electrum to create a wallet.

```
curl -X PUT http://localhost:8085/wallets/BTC --data "password=secret"
curl -X PUT http://localhost:8085/wallets/TBTC --data "password=secret"
```

With these PUT requests both Bitcoin and Testnet Bitcoin wallets can be created via the Tribler web API. With the wallet now being created via Tribler, it is also possible to

5.2 Trading

When the agent has earned MBs by running as an exit node it needs to sell these on the Tribler marketplace for Bitcoin. In 2.2.2 the commands of the Tribler marketplace where explained. And these commands still worked and where not changed during the Project. What was changed is that a market strategy had to be implemented in PlebNet this market strategy determines when PlebNet should sell its MBs. The current strategy implemented is a very basic strategy. PlebNet sells by creating asks. It regularly checks the amount of MBs it has and if it has at least one MB it will create an ask trying to sell all its MBs. The price of the MBs is determined as follows PlebNet requests the amount of Bitcoins it needs to buy a new VPS and this amount is divided with the amount of MBs. This means that PlebNet is always selling all its MBs for the amount of Bitcoins necessary to buy a new VPS no matter the amount of MBs. After the ask is created a timestamp is stored in PlebNet and PlebNet waits for exactly one hour. Now two things can happen in that hour either someone accepts the ask and a transaction is made or the ask expires(an ask always expires after an hour). After this hour has passed PlebNet will create a new ask trying to sell all of the MBs it has at that point and that is the cycle of the PlebNet market strategy.

This strategy is clearly very simple and there is a lot of room of improvement to create a smarter market strategy for PlebNet. This however was left outside of the scope of the project for two reasons. One it would cost a lot of time to implement and while it would improve PlebNet it would not help with the overall goal of this project which is to get PlebNet functional from end-to-end. The second reason is the Tribler marketplace itself. It is still in early development meaning there are as of now no buyers that are willing to trade Bitcoin for MBs and it is difficult to implement a competitive market strategy if there are no buyers available.

As mentioned before there are no current buyers of MBs on the Tribler marketplace so to be able to sell Triblers MBs this buyer has to be simulated with a bot. The bot works as follows: It queries all the asks in the market, for each ask it creates a bid that exactly matches that and after that bid

is made Tribler will automatically create a transaction between the ask and the bid selling all MBs from PlebNet to the bot. Currently the bot only buys with TBTC to showcase the proof of concept of trading.

5.3 Acquiring new VPS

The key functionality of the agent is self-replication. As mentioned before, only Linux VPSes are considered in the choosing and purchasing process. Cloudomate is used to purchase new VPSes, the user information needed for purchase is provided by PlebNet which generates somewhat believable identities for its children.

5.3.1 Purchasing

As the server is initialised, the agent chooses which server will be bought for the next child. While the agent is earning MB's and trading them for actual Bitcoin, it keeps checking Cloudomate to determine the current price for the previously chosen VPS and the transaction fee.

Once enough Bitcoin is accumulated, the agent approaches Cloudomate to attempt and purchase. As Cloudomate was updated and refactored before the initialisation of this project, many of the dependencies for this approaching were broken. The first step was to restore the communication between PlebNet and Cloudomate. This also included the centralisation of the dependencies in a specific Cloudomate controller which handles all communication and dependencies between these two programs. This should prove useful in the future when one of the applications is updated, as only one bridging connection should be restored. As the dependencies were fixed, the first purchases could be made.

```
curl -X POST http://localhost:8085/wallets/BTC/transfer --data
                "amount=0.3&destination=xxxxx"
```

As the wallet used is now created via Tribler, PlebNet has its own wallet class which it passes on to Cloudomate for purchasing VPSes. This wallet uses the POST request above for transferring money to the specified Bitcoin address.

5.3.2 Identities

The purchase of an VPS is fully handled by Cloudomate. It requires the availability of sufficient funds and an user identity which is used provide the information for the user details such as a name and an address. The parent node creates this identity for its child and uses it to purchase a new VPS. Creating one single account for the entire network would also result in a Single Point of Failure (SPoF), which could easily take down the network in case the account is banned.

During the course of the project the agent was banned multiple times and some of the purchases were dennied. Most of the time this had to do with the fact that the same VPS host was approached with a significant amount of requests from a single ip address during the implementation of a new host. This will not pose problems for the PlebNet network, as a single agent (represented by a unique ip address) will not be able buy dozens of new VPSes per day.

An agent is not able to reuse an email address for a new purchase, as the VPS host will notify that there already exists an account with that email address. This is email is a requirement for purchasing, but it is solved by using a functionality of gmail. By using a single gmail address, for example `plebbot@gmail.com` and using the appendix `+[randomstuff]` an infinite amount of new emails can

be generated, which all send the mail to the original `plebbot@gmail.com`, while the servers do not recognise them as being similar. The `[randomstuff]` is chosen to be the username of the newly created instance. This username is generated by adding the first and lastname from the identity. As these names are randomly picked from a large database in the `faker` package of python, these can be considered to be unique. Some providers, such as CrownCloud, checked their sales manually and still found out that we were buying automatically. As shown in **Fig. 5.2**

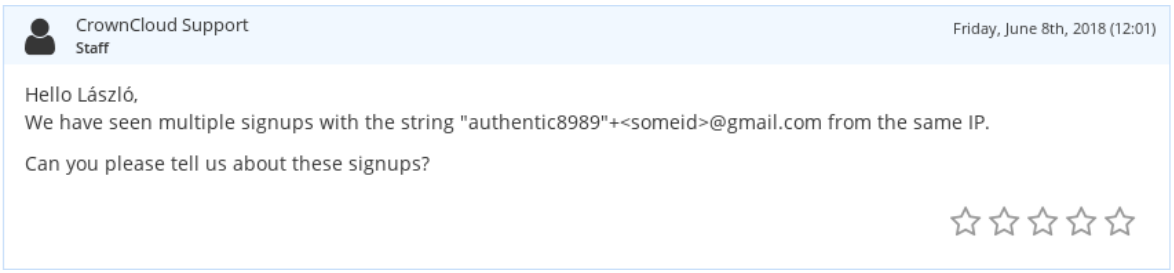


Figure 5.2: CrownCloud asking questions due to large amounts of purchase attempts

This however, occured as the implementation in Cloudomate was tested. In real life this situation would only occur if a single agent buys multiple servers a day of the same host.

5.3.3 Modifying DNA for the new child

This phase of the purchasing results in an update for the agents' DNA. When a purchase is made succesfully, the gen for the provider is increased with a certain rate (set in the `dna.py` file). The opposite happens when a purchase is not successful. The exact influence of the mutation algorithm on the survival probability should be investigated, but did not fit whitin the time reserved for this project.

Accessing the new VPS

If the purchase is successful, Cloudomate returns the IP address of the new VPS and its root password. The server is than added to an internal list with available servers. As it takes some time for the server to be initialised by the host, the agent keeps trying to log in onto the new server every iteration.

5.4 Cloning

After acquiring a new server, the agent will attempt to gain access to the server. When the server is online, the agent will start installing a new version of PlebNet and initialise a new agent.

5.4.1 approach new vps

During the purchase of a new VPS, cloudomate returns the root password and the ip address of the newly acquired server. This is stored in the configuration file (`Child_DNA.json`) and this can be used to log on into the new server via ssh. The first code to run on the bash of the new agent is defined in the `create-child.sh` file. This creates the required directories and install the proper certificates. This file contains all contact between the parent and the child node. The final step is to download the latest version of the `install.sh` file from GitHub and and run it.

5.5 Monitoring

As the program is running headless on a distant machine, inaccessible and password protected, it is not possible to find information regarding its status and progress. This also means the once the software is deployed on a server, it is unknown whether or not PlebNet keeps running or died somewhere in the process. Implementing a proper monitoring strategy allows for observation and perhaps even interaction with the online agents.

5.5.1 Legacy communication

The initial version of PlebNet used both email and Twitter[10] for communicating with the outside world. Email was used to send data such as the server configuration of the agent as well as Tribler related information such as upload/download numbers. The way email communication was implemented was trough a mail server which belonged to one of the developers of PlebNet. Having no access to this mail server, we looked into using the google mail server but it required each agent to be authenticated. For this reason we decided to remove the means of communicating through email. Twitter was used to send a message upon spawning a child. The downside of using Twitter is that it does not provide two-way communication and it is has scalability issues, when the network consists of a large number of agents each sending its status, it will flood the Twitter feed. For these reasons we decided to remove the Twitter communication module as well.

Whenever a new server is acquired, the host sends an email with the specifications to an email address. This email address can be chosen to be accessible by the person who installed the initial bot and provides some information regarding the clone speed of the online servers and the amount of online agents. However, it does not notify anyone if the server is banned or runs into other trouble such as runtime errors. This can be solved by implementing a better notification strategy.

5.5.2 IRC

An example of a solid communication network is Internet Relay Chat, or IRC[11]. This protocol is in use since 1988 and still has hundreds of thousands of users. IRC works by setting up a connection to a server node and once the bot is on, it can join a certain channel. As long as the IRC client is online and responsive to PING messages, it is notified about all messages in its channels. This means that all agents can join a chosen channel and provide information in heartbeats. This way it is possible to keep track of the still alive agents. As a new agent is installed it can send information regarding its DNA and configuration. By sending these notifications to a public chat, it is possible for anyone to keep track of the events in the botnet. This also prevent single points of failure in the network. Contrary to sending messages to a single access account, which could be lost.

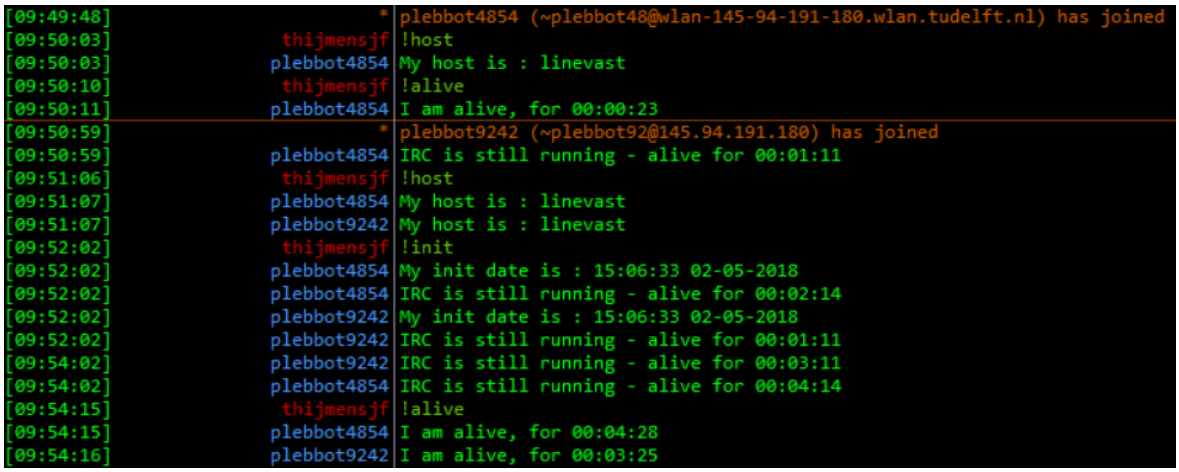


Figure 5.3: IRC communication showing the usage of the commands !host, !alive and !init

Another useful feature of IRC is the ability to send messages not only to channels, but also to specific users. As an agent joins an IRC server, it generates a nickname, `plebbot<number>` with a random number. Other users in the same channel are notified about this join and can respond to it. This method allows for asking certain information about configuration or upload/download numbers regarding Tribler. The following methods are implemented:

- `!alive` asks for a heartbeat
- `!host` asks for the host information
- `!init` asks for information regarding the initialisation of the agent
- `!MB_wallet` asks for the MB wallet address
- `!BTC_wallet` asks for the BTC wallet address
- `!TBTC_wallet` asks for the TBTC wallet address
- `!MB_balance` asks for the MB balance
- `!BTC_balance` asks for the BTC balance
- `!TBTC_balance` asks for the TBTC balance
- `!matchmakers` asks for the number of matchmakers connected
- `!uploaded` asks for amount of MBs uploaded
- `!downloaded` asks for amount of MBs downloaded
- `!helped` asks for the amount of peers helped by the agent
- `!helped_by` asks for the amount of peers that helped the agent

These commands can be send to all online agents by posting in the PlebNet channel or to a specific agent by sending a private message. New commands can be added easily in the `ircbot.py` file, but it should be kept in mind that these commands should not alter the agents. When it is possible to alter the configuration or the behaviour of the agent externally, the agent becomes dependent and other actors can change settings as well. This feature is added for monitoring purposes and added commands should oblige to this.

5.5.3 Git issues

The previous method of monitoring does allow for live communication, but it does not provide the means to properly handle errors which occur during the live stage of the PlebNet agents. Git issues are an efficient way to handle these errors. For each error which occurs, the agent should create an git issue and post the relevant information. This information should include information regarding the settings of the agent, the error and the event resulting in the error. This is done by including the error traceback and the full log of the agent. This way the behaviour of the agent can be analysed and improved for further generations of the network. An example of such an issue is shown in **Fig. 5.4**.

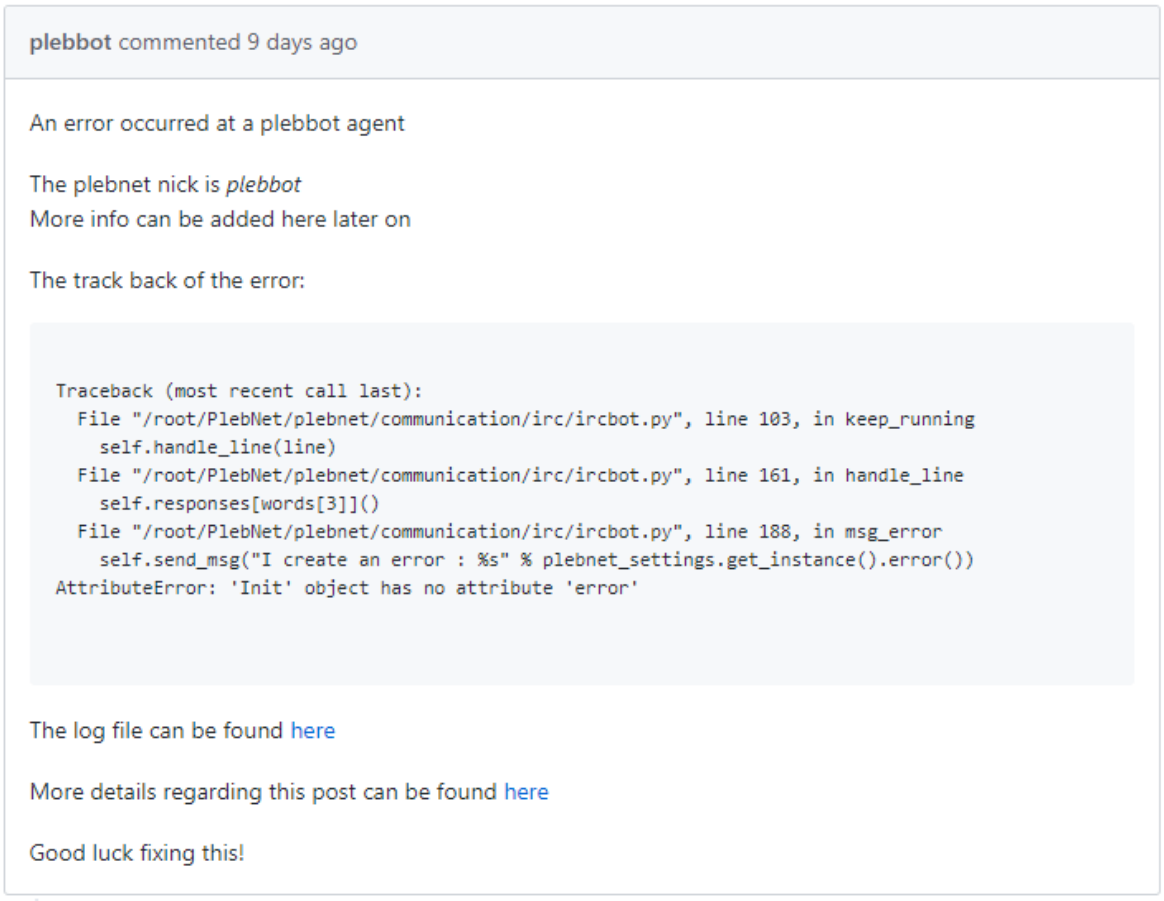


Figure 5.4: An automatically created git issue

The git issuer has to be enabled in the PlebNet configuration. A GitHub account is required and the proper repository has to be set. This can be done manually in the configuration, or using the command line setup:

```
plebnet conf setup -gu <username> -gp <password> -go <repo owner>
-gr <git repo> -ga 1
```

5.6 Additional

5.6.1 VPN

As additional functionality, a VPN is installed on the server. The VPN protects the server from DMCA claims to which most providers unfortunately respond by shutting down the server. The VPN purchasing functionaly was previously implemented in Cloudomate, but the process of purchasing and installing by the agent still needed to be implemented.
Installing a VPN as it turned out posed a few difficulties. Firstly, in order for a VPN to be installed on a VPS, TUN and TAP devices need to be enabled. The TUN(neling) interface is a virtual network devices work allowing programs such as OpenVPN to attach to it *(ref)* *https://www.kernel.org/doc/Documentation/networki*. TUN/TAP is generally not enabled on servers by default by providers, and may not even be allowed by some (**Fig. 5.5**). Furthermore, enabling TUN/TAP happens in the control panel of the provider, which has a different set of credentials than the client area (where server IP and statistics can be found). Accessing the control panel includes having to be able to access and parse the agent's email containing the credentials. Finally, Cloudomate has to be modified to access the control panel and change the TUN/TAP options.

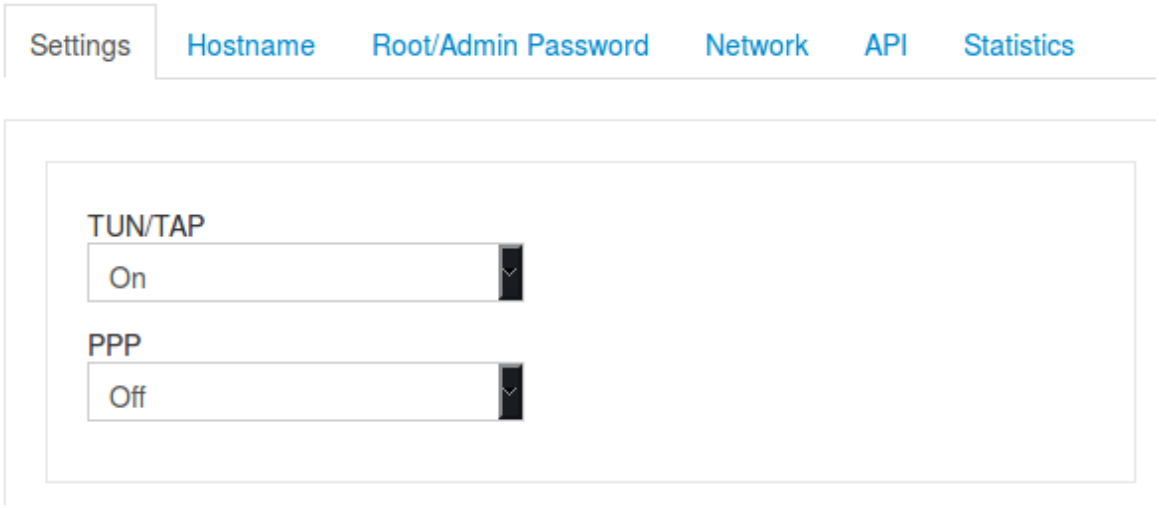


Figure 5.5: The TUN/TAP option on Linevast's control panel.

CHAPTER 6

Cloudomate

In this chapter the major changes to Cloudomate are discussed. As mentioned before, Cloudomate is responsible for providing the VPS and VPN options to PlebNet as well as handling the purchasing of these servers. The ultimate goal of the project is to make PlebNet fully operational and therefore the focus of this project is on improving PlebNet and not on improving the functionality of Cloudomate. However to make PlebNet fully operational certain changes have to be made to Cloudomate.
The changes are classified under four categories. The first category is to fix the random user generation of Cloudomate, The second category is fixing the VPS options. The third is to have Cloudomate give its VPS options dynamically to PlebNet. and the final change is to add testnet support to Cloudomate.

6.1 fixing random user generation

For Cloudomate to be able to buy a server it needs to fill in a user form. For it to be able to do that Cloudomate needs user information. A user could give its information to Cloudomate or Cloudomate could generate random user information for them. This second option is the option that PlebNet uses. However there was an error where Cloudomate would create a random user's data use it to buy a server but then Cloudomate would not save this user information. This was a problem because the user information generated includes the root password and username needed to gain access to the bought server. This meant that Cloudomate would buy a server but would not have access to it. Normally this information can be salvaged because most VPS providers sent the username and root password in an email however the email address would also be randomly generated by Cloudomate. This problem was solved by making by storing the user information in a configuration in a configuration file which can then be read when the information is queried by either the user or the PlebNet agent.

6.2 fixing the VPS providers

The ultimate goal of Cloudomate is to buy VPS providers for PlebNet to use as agents. However many of the VPS providers implemented in Cloudomate have made changes to their services. Examples of these changes would be changing the layout of their websites or changing the Bitcoin gateway used for the transactions. These changes make it impossible for Cloudomate to buy VPS servers of these providers. At the start of the project Cloudomate had six VPS providers implemented these were: Linevast, Blueangelhost, ccihosting, crowncloud, pulseserver and undergroundprivate. Of these six only Blueangelhost still works for Cloudomate making PlebNet completely reliant on one VPS provider. And if that provider were to make changes PlebNet would not be able to replicate itself anymore. So to prevent this from happening the other five providers have to be fixed or in the case of them not being fixable a replacement provider has to be implemented.

6.2.1 unfixable providers

First the providers that aren't fixable will be discussed. These providers are Ccihosting, Pulseserver and Crowncloud. Ccihosting and Pulseserver are not fixable for the same reason and that reason is that

they both use Coinpayments as their Bitcoin transaction gateway. This gateway made a major change during this project that makes it unusable for a program like Cloudomate. That change is how it handles invoices. The way invoices work is as follows after Cloudomate has ordered a VPS server an invoice is made on the providers corresponding gateway. To make the payment Cloudomate has to go to the gateway url with the corresponding invoice. from that web page Cloudomate can collect the address and the amount needed for the payment. With this information Cloudomate can make the purchase and acquire the VPS server. The way Coinpayments handles this invoice url is fundamentally different from different gateways. Take Bitpay as an example their the invoice can be added to the url itself in and an invoice url would look as follows https://bitpay.com/invoice?id=exampleID. In Coinpayments this isn't the case their invoice url's look as follows https://www.coinpayments.net/index.php. However if you just go too that url it will direct you to the main page of Coinpayments. here the invoice id is given as a secret token and the page has to be accessed with that token to be able to see the invoice. And the python library Cloudomate uses to access these urls beautifulsoup doesn't have a way implemented to access these urls with a secret token. So for that reason if Cloudomate makes an order to Ccihosting or Pulserver it isn't able to access the invoice url and therefore it cannot retrieve the address and amount it needs for the transaction. So for that reason Cloudomate is unable to use these two providers.

The other provider Crowncloud was not fixable for a different reason. And this reason was also a problem the previous development group had issues with. This issue is that it is possible to buy a Crowncloud VPS server however Crowncloud doesn't give the buyer an option to set the root password during purchasing. So when Cloudomate purchases a Crowncloud instance it generates and saves a root password however Crowncloud doesn't allow setting a root password during purchasing and creates its own root password. This means that the root password in Cloudomate is not correct and won't be able to be used to access the bought Crowncloud VPS server. However Cloudomate and PlebNet are not aware that this is the case so they will try to access the VPS server with a wrong password and this will cause errors.

To change this one of the following would need to be implemented. Crowncloud sends an email with the root password this email would have to be scraped either to retrieve the root password and set this password in Cloudomate or the email would have to be scrapped to find a link to the Crowncloud control panel. In this control panel a new root password for the bought server could be set but this would require new scraping and form filling methods that are not implemented in Cloudomate. So either way to fix Crowncloud many methods had to be written specifically for this VPS provider and for that reason it was decided to leave Crowncloud and focus on other VPS providers.

6.2.2 fixed and new providers

Now the VPS providers that have been fixed and the ones that have been newly added will be discussed. These providers were Linevast, Undergroundprivate and a new provider called 2-Sync. First Linevast will be discussed. Linevast was relatively easy to fix. The were fully operational for most of the project however during the project Linevast decided to change its main web page which listed all its VPS options. So this made it impossible for Cloudomate to read the VPS options which include the purchase URLs for each option. And without these URLs it is impossible for Cloudomate to buy the VPS options. After repairing the Scrapping methods Cloudomate was able to retrieve these options again and after that Cloudomate was able to buy Linevast servers again.

The next provider fixed is Underground private they had changed their Bitcoin gateway from blockchainv2 to spectrocoin. Too fix this two things had to be done. First when a VPS server is ordered the link to the gateway isn't automatically given. The page has to be loaded first and after five seconds of loading the web page is automatically sent to the gateway invoice. However Cloudomate uses Beatifulsoup for webscraping and that library is a static browser so automatically loading doesn't work. Luckily

once an order is made a button is displayed on the webpage and with a simple on click function Beautifulsoup can go to the gateway. The second thing that had to be changed was once Cloudomate could access the invoice url it had to scrape the relevant inforamtion. After implementing this it was possible to buy VPS servers from Undergroundprivate again.

The final provider is one that wasn't fixed but was added is 2-Sync. 2-Sync is a website that offers not one but many types of VPS provider however in Cloudomate only the Ukrainian VPS provider was implemented. 2-Sync itself was chosen because the website design and forms used are similar to other already implemented VPS providers so implementing 2-sync itself did not cost a lot of extra work. Only the Ukrainian option was chosen because it uses a simple self designed gateway from which the relevant information could be easily scraped 6.1. This wasn't the case for many other 2-Sync providers which used more complicated gateways with on example being Coinpayments. The implementation of 2-Sync itself went relatively easy without issues.

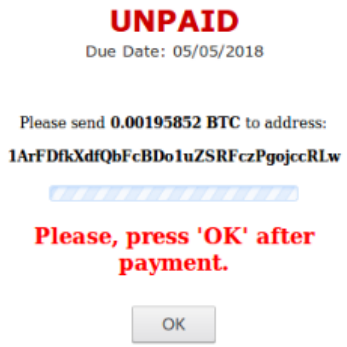


Figure 6.1: 2-Sync gateway

At the start of the project six VPS providers were implemented now after fixing and adding VPS providers there are now four fully functioning VPS providers in Cloudomate. These are Blueangelhost, Linevast, Undergroundprivate and 2-Sync.

6.3 Dynamically sending VPS options to PlebNet

The problem with the delivered version of the previous development group was that the VPS options where hard coded both in PlebNet and in Cloudomate. This is an issue for the following reason. If a provider were to be removed from Cloudomate or a new provider were to be added PlebNet would not be aware of that unless a developer manually added updated the options in PlebNet. This could lead to instances were PlebNet is unable to buy from certain VPS providers or tries to buy from a provider that is no longer supported by Cloudomate. To fix this issue Cloudomate had to directly send its options to PlebNet so that when the providers are changed PlebNet's VPS provider list is automatically updated. The way this is implemented is quite simple. There is a list of working providers in Cloudomate and PlebNet can request this list via the Cloudomate controller. And with this list PlebNet will always stay up to date with Cloudomate.

6.4 Adding end-to-end testing support

One of the major additions to PlebNet is implementing end-to-end testing. However part of the end-to-end process of PlebNet is buying a server through Cloudomate. So to implement end-to-end testing certain additions had to be made to Cloudomate. These additions are twofold. The first being the implementation of buying proxmox containers these containers are used to mock the VPSes that PlebNet has to run on. This is by adding proxhost to the VPS providers. proxhost is a mock website of a VPS server provider. This website **Add screenshot of website** is used by Cloudomate like any other. Cloudomate scrapes the web page to gain VPS options and purchase the proxmox container. After this mock payment that goes through the Bitpay gateway the proxmox container is initialised for PlebNet to use.

The second addition is adding testnet support testnet is an alternative currency provided by Bitcoin that is used exclusively for testing purposes meaning the currency used has no real value. But to be able to make transactions with this new currency Cloudomate had to be updated. The updates were relatively minor. If Cloudomate is used standalone the user has to provide a –testnet command in the command line. After this Cloudomate sets a universal variable which means that Cloudomate will only allow the user to buy form the proxhost provider. And all payments will use testnet and not Bitcoin. If PlebNet creates a testnet wallet it will set the same universal variable. Cloudomate will see this and again only allow PlebNet to buy from the mock proxhost provider with testnet coins. With these changes if PlebNet is end-to-end tested Cloudomate will mock the purchasing of a server and deliver the proxmox container back to PlebNet making that part of the end-to-end test possible.

CHAPTER 7

Quality assurance

this chapter is still work in progress

After refactoring PlebNet, both PlebNet and Cloudomate had to be tested. As our main focus was on PlebNet, we decided to fix the existing failing tests of Cloudomate and not write any new tests for Cloudomate. PlebNet was not tested at all, so we decided to test functionalities of PlebNet independent of each other as well as end-to-end test the whole system. Keeping track of the project was done using Jenkins as a continuous integration tool. [12]

7.1 Unit testing

7.1.1 Testing Cloudomate

Certain tests that used to work on Cloudomate failed at the start of this project. So one of the early priorities was to fix these tests and make sure that the Cloudomate Jenkins build would succeed. The tests that have been successfully fixed are as follows: fixed the way that bitcoin urls are split up, fixed the list options method for AzireVPN, Made it so that the email used does not end in @email.com this particular address was blocked by multiple vps providers and fixed the command line test class. This class had an issue with mocking which caused other test classes to fail. The tests that haven't been able to be fixed are as follows: The tests for the Coinbase gateway failed this is because the test URL wasn't supported anymore. To fix this a new test URL had to be set up but this wasn't done because none of the providers used the Coinbase gateway anymore so that class and its corresponding tests were considered obsolete. The tests that gave the most trouble were the VPS purchase tests. These tests scrape VPS providers website make an order, fill in the user form and scan the bitcoin payment gateway for the details necessary to make the transaction. The tests didn't actually purchasing these VPS servers. Because the tests were reliant on the VPS providers websites these websites had to be consistent for the purpose of the tests working. However for many providers this wasn't the case. websites layouts were changed, as mentioned in the gateways section multiple providers changed from bitcoin payments, increasing the difficulty of scraping for example by adding new on click buttons and some of the VPS providers websites where down for up to a day at a time which makes testing these providers impossible. This resulted in many difficulties often resulting that every day one of these tests would fail for a new reason. So these tests are now skipped because the websites they rely on were considered to be too variable.

7.1.2 Testing PlebNet

As there were no tests written for PlebNet by the previous group and since we were planning to change the structure of PlebNet, we decided to wait with testing until the new structure was implemented. After refactoring, it became easier to test PlebNet as calls to dependencies were handled by a single dedicated controller. The behaviour of these controller functions could now be mocked and thus simplify testing. **Problems while testing !!**

Jenkins

As stated earlier, Jenkins is the framework used for continuous integration. This includes running all tests with each pull request as well as creating a coverage report with each build. Jenkins is chosen, as the Tribler organisation uses it already and the previous group used it for Cloudomate as well. This way all projects are maintained by a single service. While setting up Jenkins, there were some problems we ran into. Our project is intended to run as root and uses several directories for storing data such as log files and configuration files. We had to be sure that these directories are correctly called. As well as creating a virtual environment on Jenkins in order to install our dependencies in. As we have never worked with Jenkins before, setting this up took some trials. **Fig. 7.1** shows the coverage percentage per build. As can be seen, there are some drops in coverage which are caused by the problems described above.

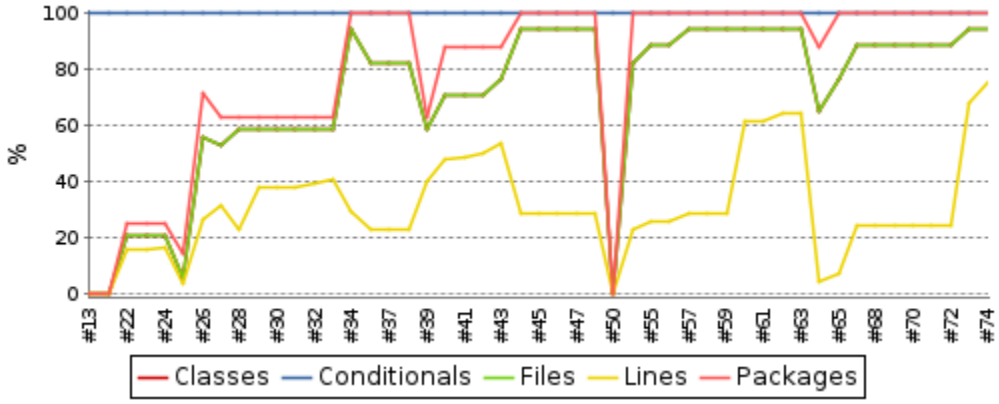


Figure 7.1: Coverage percentage per build

The total coverage we have achieved for PlebNet is **76%** . More details can be seen in **Fig. 7.2**.

Coverage report: 76%

Module ↓	statements	missing	excluded	coverage
plebnet/__init__.py	0	0	0	100%
plebnet/agent/__init__.py	0	0	0	100%
plebnet/agent/config.py	38	6	0	84%
plebnet/agent/core.py	108	84	0	22%
plebnet/agent/dna.py	94	10	0	89%
plebnet/clone/__init__.py	0	0	0	100%
plebnet/clone/server_installer.py	47	12	0	74%
plebnet/cmdline.py	82	9	0	89%
plebnet/communication/__init__.py	0	0	0	100%
plebnet/communication/git_issuer.py	56	45	0	20%
plebnet/communication/irc/__init__.py	0	0	0	100%
plebnet/communication/irc/irc_handler.py	34	1	0	97%
plebnet/communication/irc/ircbot.py	107	21	0	80%
plebnet/controllers/__init__.py	0	0	0	100%
plebnet/controllers/cloudomate_controller.py	97	5	0	95%
plebnet/controllers/market_controller.py	50	9	0	82%
plebnet/controllers/tribler_controller.py	25	3	0	88%
plebnet/controllers/wallet_controller.py	59	0	0	100%
plebnet/settings/__init__.py	0	0	0	100%
plebnet/settings/agent_settings.py	26	26	0	0%
plebnet/settings/plebnet_settings.py	67	5	0	93%
plebnet/settings/setting.py	31	7	0	77%
plebnet/utilities/__init__.py	0	0	0	100%
plebnet/utilities/fake_generator.py	55	0	0	100%
plebnet/utilities/logger.py	41	3	0	93%
Total	**1017**	**246**	**0**	**76%**

Figure 7.2: Coverage report on Jenkins

7.2 End-to-end testing

To properly test whether PlebNet performs well from beginning to end, a Proxmox server is used to to emulate VPS providers. PlebNet is placed on one of the Proxmox containers. The remaining free servers are provided by Cloudomate as options under 'ProxHost' and can be bought through the Bitpay gateway using testnet Bitcoins. The motivation behind such an end-to-end testing system is that it allows for easier testing opportunities. Actual providers are quick to shutdown their servers when DMCA claims are filed, making it difficult to test anything related to Tribler. Furthermore, having to test PlebNet by purchasing real servers is not a cost efficient and stable way to develop. In addition to providers shutting down servers, providers are also quick to change their web layout, making purchases via Cloudomate unreliable when the focus is on PlebNet. Although repairing Cloudomate's parsing of hosts is a largely trivial task, it is another issue that this system helps minimize.

7.2.1 Proxmox

The Proxmox server is managed via *ansible* scripts. Using ansible, creating, cloning and restoring containers can be done automatically. The backend consists of three parts: the web api, used for managing bought containers; the Flask application and payment controller and the ansible scripts.
The ansible scripts contain tasks that create Debian or Ubuntu containers and set up the network including peervpn.
Peervpn is used to create a network in which the containers can be assessed from outside the network by peers that are on the same VPN. The reason for this is because the containers are behind a single ipv4 (NAT), while ip routing can be used to gain assess to individual containers, assigning a unique ip address to each container more realistic as far as trying to emulate a real provider.
The Flask application provides entrance to the management section and contain routes which are assessing to cloudomate allowing for purchase and status requests from cloudomate.

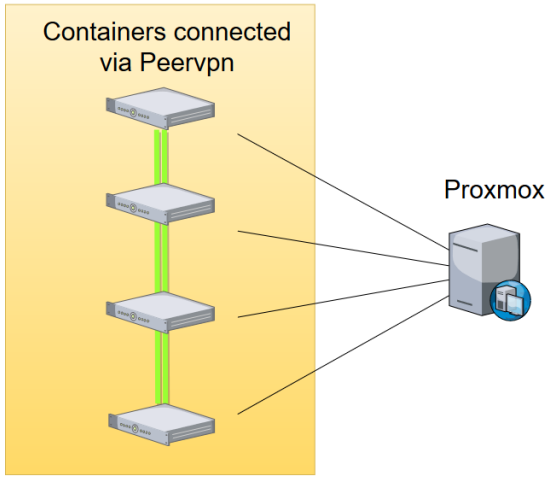


Figure 7.3: Overview of the Proxmox server. Containers are connected with each other through PeerVPN.

7.2.2 Bitpay testnet

Bitpay offers a testing service (test.bitpay.com) which makes use of testnet Bitcoins. With a merchant account, invoices can be created by ProxHost. These invoices can be paid using the Electrum wallet in testnet mode. To make use of the API Bitpay provides, the BitPay library for Python2.7 was used. Each machine hosting ProxHost needs to first have a token verified from the Bitpay control panel.

7.2.3 Web API

A minimal web API was created to destroy containers bought by Cloudomate. Additionally, containers for testing can be created and destroyed.

7.3 Maintainability

7.3.1 SIG

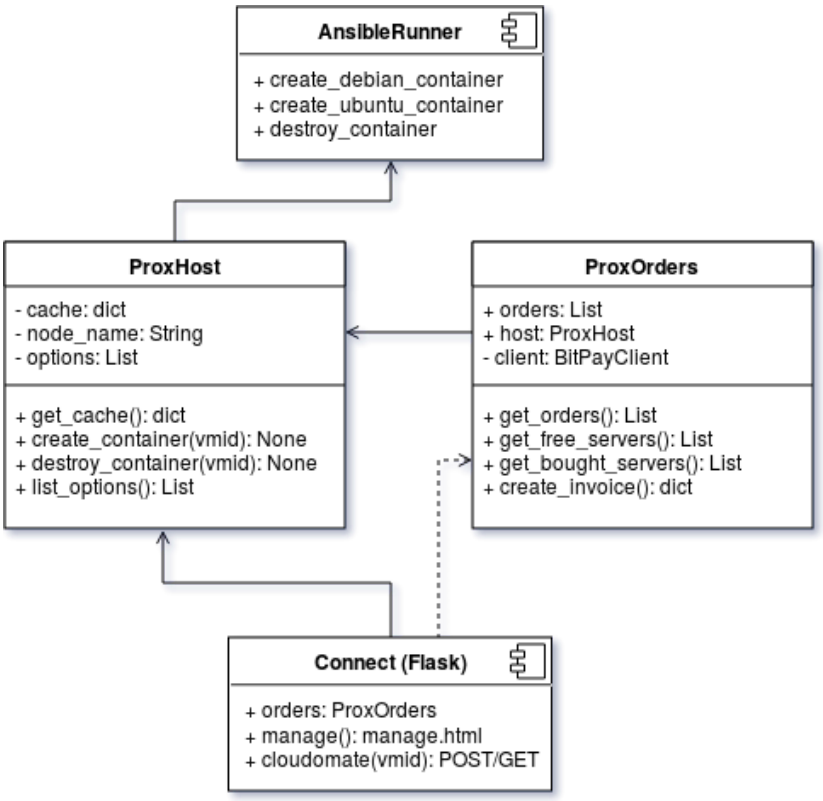


Figure 7.4: A simplified UML diagram of the end-to-end system. The Flask component "Connect" allows Cloudomate or web users to purchase containers. When a purchase is made, ProxOrders creates an invoice which Connect returns as a HTTP response. ProxHost can either create or destroy containers by creating an AnsibleRunner object with the appropriate parameters.

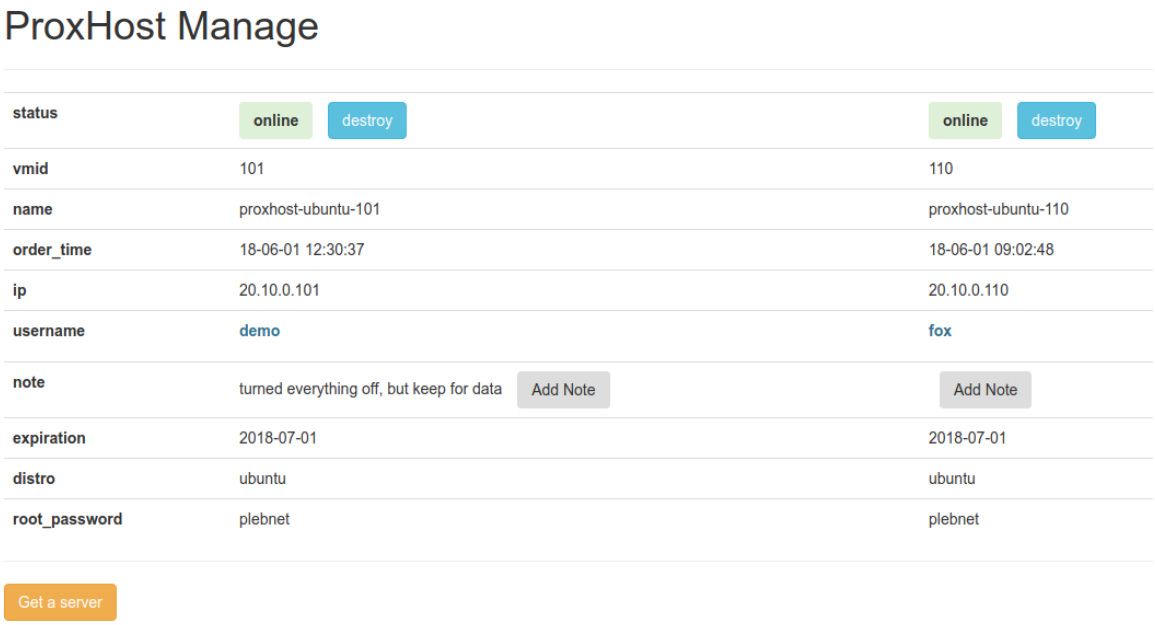


Figure 7.5: The minimal web API provides an easy way to destroy and create containers.

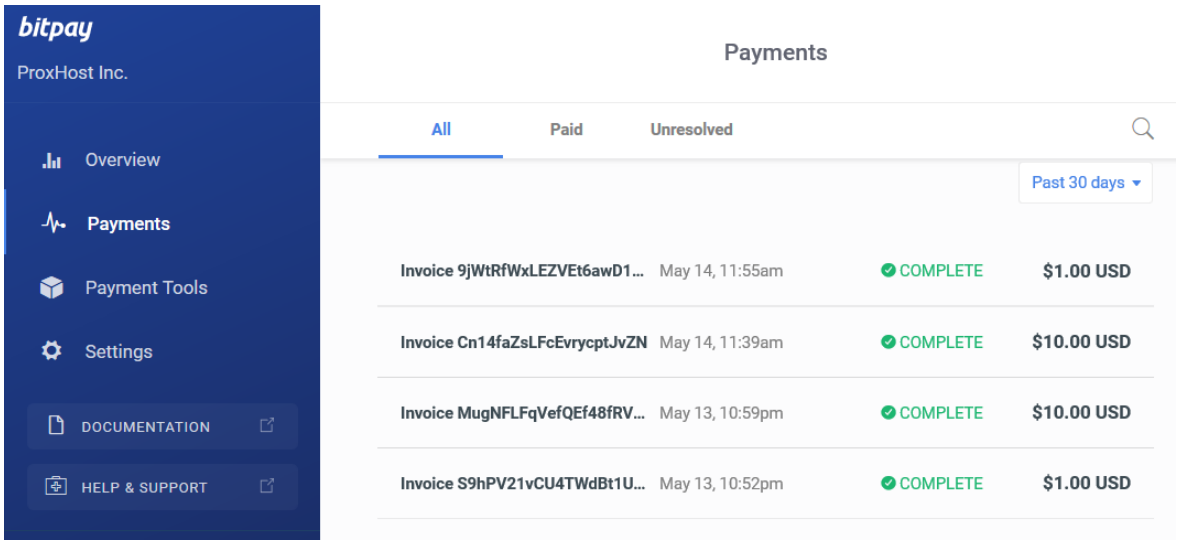


Figure 7.6: The testnet Bitpay merchant page. Invoices can be created and paid in testnet Bitcoins.

CHAPTER 8

Conclusions

At the end of a project it is always good to look back and conclude was has been done. This chapter does exactly that and it also discusses what can be improved in PlebNet in the future.

8.1 Conclusions

The Must haves are all implemented. As all the must haves which were determined at the start of the project are met, the minimal requirements are accomplished. The system can now be used as it is end-to-end operational with testnet and with actual Bitcoin. This results in a usable version of PlebNet which can keep it self alive as it earns enough MB while running as an exit node and is able to trade these for sufficient BTC funds to acquire new VPS instances for its children. The network is stable if these requirements are met. For now the incentive is too low for the market to provide enough BTC to keep PlebNet running, but this can be simulated by the `buybot` which is implemented as well. As the idea behind the Tribler market work out, the network should become more self-reliant.

The monitoring of PlebNet is also improved as the IRC can be used to ask for information from the online agents. This can be used to monitor the stability of the system as a whole or a specific agent running on a specific server. This provides useful insights in the effectiveness of PlebNet in providing sufficient exit-nodes for Tribler to be anonymous.

The continuous integration was also accomplished and can be used to verify the behaviour of PlebNet, as it verifies that all tests succeed.

The Should haves are all implemented. The first two were regarding the project work flow: improving documentation and maintainability. This should allow the next generation of programmes who work on PlebNet to have a better start and understanding of the code. This is mostly accomplished by implementing a clear structure for PlebNet and add comments and other documentation such as UMLs. As the structure is also discussed in this report, it should be possible to retrace our steps in decision making regarding the structure.

The Could haves are mostly implemented. The option to live monitoring and to create GitHub issues are added to the features of PlebNet. This allows PlebNet to provide useful information to improve the system. The work done to make use of Proxmox for end-to-end testing in a controlled environment also allow the possibilities to test certain behaviour, while being in full control of all the restraints.

The unimplemented **Could haves** were untouched due to the time restraints. The transferring of funds at the end of the life-cycle or another end of the bot are to be considered carefully. This will often mean that there should be some form of communication between bots, which is undesirable in a distributed system, as it is uncertain whether the other party can be trusted. This strategy requires an extensive research and decision making which did not fit in the time for this project. The same

applies for the genetics part of PlebNet. It allows the implementation of multiple strategies, but should be tested on their effectiveness on staying alive as a system.

The packaging of PlebNet was abandoned as it was considered unstable and heavily dependent on the VPS settings of the providers. The current method, using the install scripts, provides enough stability and can easily be adapted or extended.

The Won't haves are mostly left alone. During the project however, it was decided that new VPS hosts had to be implemented in order for the system to be stable enough, as most of the originally implemented providers did not work anymore. This resulted in the addition of two more hosts to Cloudomate.

8.2 Ethical Considerations

Peer-to-peer networks have existed since the early years of the internet; Napster, Kazaa and Limewire were among the most popular file-sharing services. Nowadays, Bittorrent is used worldwide and many variants of Bittorrent clients have been introduced such as qBittorrent, *µtorrent*, Transmission and streaming clients such as Popcorn-time and Tribler.

The main controversy surrounding P2P networks is that most content shared within these networks is copyrighted material. While the question whether sharing copyrighted material is ethical or not is a long discussion in and of itself, it is perhaps fair to say that the taboo surrounding P2P networks that allow people to download movies, music, software and literature without paying is deserving.

However, a P2P network is in its purest form just a file-sharing network. Similar to a village in which people can gossip and share their possessions, such a network is a way to exchange information in the digital age. In a similar vein, when the library containing all the books in the village is burned down, almost all information is lost. However, if the villagers store their books in a decentralised manner (storing books in their own homes), the information is much more likely to be preserved. These arguments are not to oppose public libraries, but are rather to illustrate the benefits of P2P networks; it is for example not uncommon for people to turn to torrents for content that would otherwise be impossible to attain, be it due to the content not being available for purchase anywhere or due to one's financial circumstances. With P2P networks, communities consisting of people from all over the world can be formed, allowing people in developing countries for example to have access to education.

As mentioned in the introduction, free speech and fair use of digital content are being threatened by upcoming laws. While the arguments for these laws are to protect the creators of intellectual property (IP) such as movies and music, the line between fair and unfair use of digital content is difficult to draw. Following the trends of these laws, it is not hard to imagine a world in which information sharing is strictly monitored, stunting the growth of academic fields and personal development.

Tribler allows for peer-to-peer file-sharing while preserving the user's anonymity. Although P2P networks and the torrenting world in general are resilient, which is apparent when one thinks of the resilience of The Pirate Bay *(ref)* *https://torrentfreak.com/the-pirate-bay-remains-on-top-11-years-after-the-raid-170531/* , the need for a healthy community of users is important for these networks. Specifically, in the case of Tribler which protect its users, a large network of exit nodes is needed due to its Tor-like framework.

PlebNet decreases the need for users to volunteer as exit nodes and therefore protects the privacy of Tribler users. The government and big corporations should not be able to censor or limit our use of the internet. If we were to live in a society in which free speech and information are not privileges but rights, resilient P2P networks with anonymity is imperative. For this reason, we believe that this project in conjunction with Tribler is ethical.

8.3 Discussion

Between the start of the project and the end a learning curve was noted. For most of the team members it was new to program for Linux and fully use this operating system. The working with Linux resulted in some new approaches which had to be understand, such as the full use of the terminal and the problems regarding dual boots and Virtual boxes did not speed up the initial phase. This, in combination with the lack of documentation of the provided code, resulted in a chaotic start. The goal was clear, but there was so much to achieve before PlebNet would be anything like the goal. This resulted in quite some struggles during the start of the project, but after creating UML's of the structure and creating our own documentation, the lack of structure became more clear. Refactoring resulted in a clearer overview and a better task division between the group members.

Some remarks regarding the origanization of the project have to be made. First of all it took some time to find an optimal and efficient way of using sprints. In the beginning the effort estimation were to optimistic and each member received too many tasks to perform. During the project this was improved and the sprints were better executed. This resulted in the tasks being performed in order based on their priority. The sprints were finished on Monday morning, followed by the start of the next sprint and every Friday the progress was discussed to see determine which tasks required more attention. This resulted in a pleasant cooperation.

The use of programming tools such as Github was done properly. Initially, every member had its own fork, but this resulted in an overload of merges and made it hard when multiple members were working on the same code, resulting in many merge conflicts. Therefore it was decided to create one main fork for each module (PlebNet, Tribler, Cloudomate) and use that one to merge into. The handling of merging and pull requests was done in such a way that all pull requests were first checked by another team member, before accepting. This worked well for the group and the projects.

All in all, in can be concluded that a lot was learned over the course of the project, regarding organisation skills as well as new programming skills.

8.4 Further work

PlebNet in its current form is far from finished. The basics work and PlebNet is able to run and maintain itself, but this does not mean that there is no future work to be done here. First of all the monitoring can be expanded. More methods can be added to provide a better insight in the state of the agents. The use of Twitter and other social media can also be implemented for promotional purposes. Another good addition would be to create a visualisation tool which displays the tree structure of an online network and the data flowing through.

In order for the system to be stable, it requires multiple VPS and VPN providers to be implemented. As a change in interface of the purchasing pages can easily remove a provider from the available list in Cloudomate, it is required that this list is expanded further. Neglecting to do so would leave PlebNet inoperable.

The trading can be improved as well. Currently the bots aims to sell all MB to be able to buy a new VPS after a single deal. Multiple strategies, some even including spending Bitcoin on MB whenever the price is right, can be tested and implemented to increase the life expectancy of the network. The BTC earned by a bot which is about to die is lost with the current implementation. This could be sent to a single child, or even spread out over multiple children.

The genetics part of PlebNet also deserves an update. The current algorithm is low-level and it takes many iteration before the DNA changes significantly to favour one provider over another. Using proxmox it can be evaluated what the effect of different update rates is and how they would affect the survival rate of PlebNet. It would also be interesting to look at more genes beside the provider choosing. For example the usage of new newest version of PlebNet or a known-to-be-stable version. Or using different trading tactics based on DNA. This is an interesting field and can help greatly to improve PlebNet.

The agents should also be better armed against being banned to use a certain provider. The purchasing behaviour could be more human-like by automatically creating a new email address with a trusted extension (`@gmail` for example). Having a unique ip for every bot and a unique email address for each sale. This would make it hard to detect automated purchases.

Bibliography

[1] Toezicht op aftapwet is onder meer gericht op 'sleepnet' en delen van data. https://tweakers.net/nieuws/138031/toezicht-op-aftapwet-is-onder-meer-gericht-op-sleepnet-en-delen-van-data.html. Accessed: June 11, 2018.

[2] Net Neutrality Has Officially Been Repealed. Here's How That Could Affect You. https://www.nytimes.com/2018/06/11/technology/net-neutrality-repeal.html. Accessed: June 11, 2018.

[3] Article 13 could "destroy the internet as we know it": What is it, why is it controversial and what will it mean for memes? http://www.alphr.com/politics/1009470/article-13-EU-what-is-it-copyright. Accessed: June 11, 2018.

[4] What Is Fair Use? https://fairuse.stanford.edu/overview/fair-use/what-is-fair-use. Accessed: June 11, 2018.

[5] Tribler - Privacy using our Tor-inspired onion routing. https://www.tribler.org/. Accessed: April 24, 2018.

[6] Electrum Bitcoin Wallet. https://electrum.org/#home. Accessed: April 27, 2018.

[7] N.C. Bakker, R. van de Berg, and S.A. Boodt. Autonomous Self-replicating Code. June 2016.

[8] J. Heijligers, R. van den Berg, and M. Hoppenbrouwer. Plebnet: Botnet for good. June 2017.

[9] Snapcraft. https://docs.snapcraft.io/snaps/. Accessed May 17, 2018.

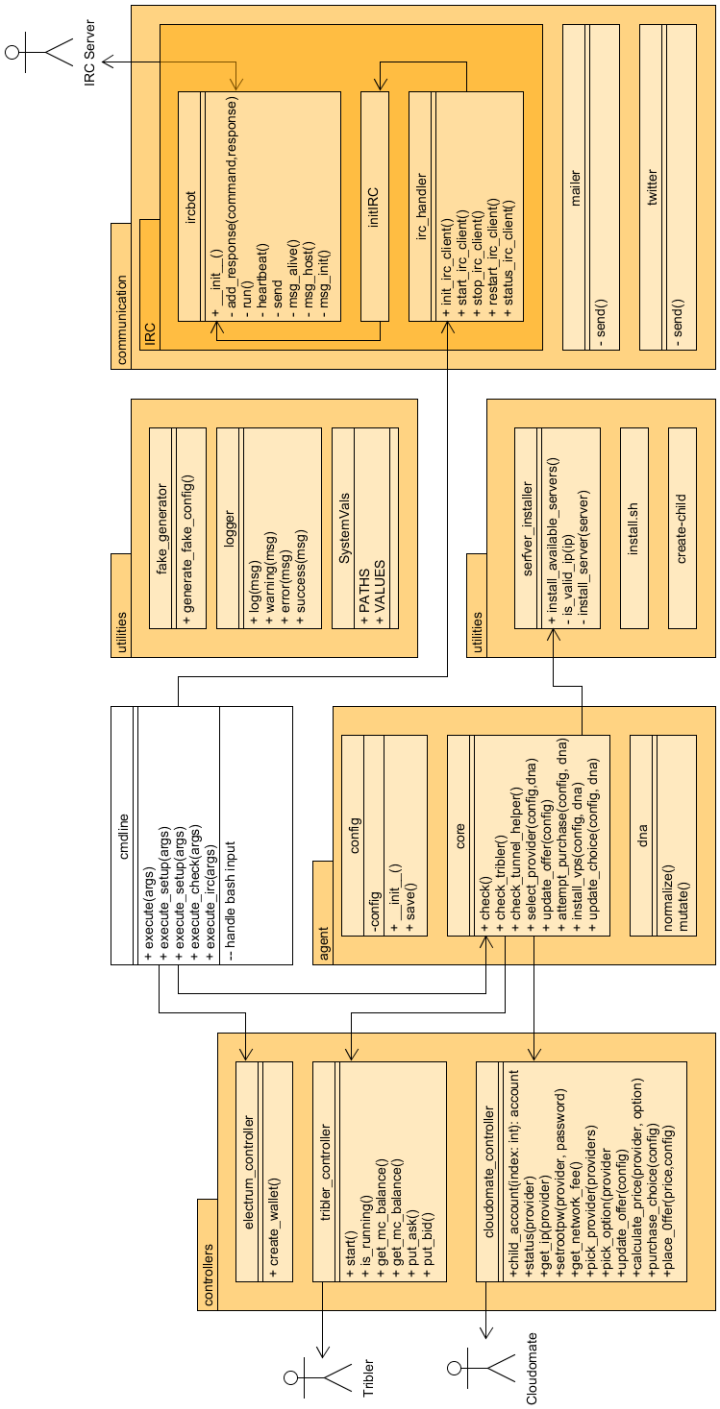
[10] Twitter. https://twitter.com/?lang=nl. Accessed: June 15, 2018.

[11] mIRC: Internet Relay Chat client. https://www.mirc.com/. Accessed: May 2, 2018.

[12] Jenkins. https://jenkins.tribler.org/job/GH_PlebNet/.

APPENDIX A

Class Diagram



APPENDIX B

SIG feedback week 5

Beste,

Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat een aantal aanbevelingen die meegenomen kunnen worden in de laatste fase van het project.

Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van hun code en dient niet gebruikt te worden voor andere doeleinden.

Mochten er nog vragen of opmerkingen zijn dan hoor ik dat graag.

Met vriendelijke groet,

Dennis Bijlsma

[Feedback]

De code van het systeem scoort 4.1 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Interfacing en Unit Size.

Op dit moment is de score dusdanig hoog dat we geen concrete aanbevelingen voor verdere verbetering hebben, hulde! Wel is het zaak om ervoor te zorgen dat jullie dit niveau tijdens het vervolg van het project vast weten te houden, en al helemaal op het moment dat de deadline in zicht komt.

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid tests blijft nog wel wat achter bij de hoeveelheid productiecode, hopelijk lukt het nog om dat tijdens het vervolg van het project te laten stijgen.

Over het algemeen scoort de code dus bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

APPENDIX C

Project Description

The goal of this Bachelor end project is to create a Bitcoin-based entity which can earn money, mutate, multiply, and dies. We provide a crude starting project which needs decades of further work in order to be considered 'really' living. This project builds upon existing code that creates autonomous life. Two previous groups have worked on this system and their documentation can be found in the TU Delft repository (see here and here). Recently, two groups of master students have expanded the project.

You will create an Internet-deployed system which can earn money, replicate itself, and which has no human control. In the past, humanity has created chess programs that it can no longer beat. The distant future of an omniscient computer system that on day chooses to exterminate humanity in the Terminator films is not the focus of your project. You will create software that is beyond human control and includes features such as earning money (Bitcoin) and self-replicating code (the software buys a server and spawn a clone). Earning money consists of helping others become anonymous using the Tor-like protocols developed at TUDelft and our own bandwidth token designed for this purpose, called Tribler tokens. A cardinal unanswered question is how to securely pass the wallets with Tribler tokens and Bitcoin to the offspring servers. You Python software is able to accomplish some parts of the following functionality:

- Earn income in a form of a bandwidth token (existing code).
- Sell these earned tokens on a decentralized market for Bitcoin or other currencies.
- Buy a server with Bitcoin or Ethereum without human intervention (see our existing PyPi scripts).
- Login to this Linux server and install itself with code from the Github repository.
- Automatically buy and install VPN protection to hide the outgoing traffic from the servers.

The software should be able to have a simplistic form of genetic evolution. Key parameters will be inherited to offspring servers and altered with a mutation probability. For instance, what software version of yourself to use (latest release?), what type of server to prefer buying (quad core, 4GB mem, etc), and whether you offer Tor exit node services for income or not. Bitcoins owned by TUDelft will be used to bootstrap your research