

PlebNet

A self-replicating botnet for privacy protection

T.S. Jaspers Focks BSc.

W. Nguyen

H. van Velton

V. Wigmore



PlebNet

A self-replicating botnet for privacy protection

by

T.S. Jaspers Focks BSc.
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 Wednesday July 4, 2018.

Project duration:	April 23, 2018 – July 4, 2018	
Thesis committee:	H. Wang	TU Delft, Bachelor Project Coordinator
	Dr. Ir. J. A. Pouwelse	TU Delft, coach
	Ir. M. A. de Vos	Delft Blockchain Lab, client

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

Preface

This report is written as part of the Bachelor End Project. It describes the development of PlebNet for the Tribler organisation. Tribler is a peer-to-peer file sharing program developed at Delft University of Technology. PlebNet is an autonomous self-maintaining network of exit nodes for Tribler. Meaning, it tries to create a stable network of servers which all function as an entry point for the Tribler network. For achieving this, PlebNet has to earn money to purchase these servers.

This report provides information on the requirements, design decisions, implementations and our final results as well as recommendations. We would like to thank dr. ir. Johan Pouwelse and ir. Martijn de Vos for their guidance and inspiration during the Bachelor End Project.

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

Summary

As the amount of digital content being shared is increasing, the need for a privacy protecting and resilient method of file sharing grows. New copyright law proposals are suggesting that governments and corporations are attempting to claim control over the use of the internet. As these new laws may lead to restrictions in digital content sharing, as well as endangering the privacy of users, alternative methods of sharing files may be able to solve these issues. Peer-to-peer networks for file sharing, such as BitTorrent, are resilient networks due to their decentralised nature. Tribler combines the BitTorrent protocol with a Tor-like anonymity system to protect its users against governments and spooks. Due to its Tor-like onion routing protocol, exit nodes are important for a robust network. This project focuses on PlebNet, an autonomous self-replicating network of Tribler exit nodes. Using Cloudomate, PlebNet is able to purchase servers with Bitcoins acquired from selling Tribler's bandwidth tokens and replicate itself to expand its network of exit nodes.

Contents

1	Introduction	1
2	Related work	5
2.1	Cloudomate	5
2.1.1	Gateways	7
2.1.2	Electrum	7
2.1.3	Functionality review of Cloudomate	7
2.2	Tribler	8
2.2.1	Running a Tribler exit node	8
2.2.2	The Tribler marketplace	9
2.2.3	Matchmakers	10
2.3	PlebNet	10
2.3.1	Purpose of PlebNet	10
2.3.2	Choosing and installing offspring	11
2.3.3	Original state	12
3	Requirements analysis	13
3.1	Must haves	13
3.2	Should haves	13
3.3	Could haves	14
3.4	Won't haves	15
4	Improving PlebNet	17
4.1	System architecture	17
4.1.1	Old Architecture	17
4.1.2	Improved Architecture	19
4.2	Initialisation of PlebNet	21
4.2.1	Packaging	21
4.2.2	Bash scripts	22
4.2.3	Wallet creation	23
4.3	Trading	23
4.4	Acquiring new VPS	24
4.4.1	Purchasing	24
4.4.2	Identities	24
4.4.3	Modifying DNA for the new child	25
4.5	Cloning	25
4.6	Monitoring	26
4.6.1	Legacy communication	26
4.6.2	IRC	26
4.6.3	Git issues	29
4.7	VPN	31
4.7.1	Gaining access to the control panel	31
4.7.2	Handling VPN from PlebNet	31

5	Improving Cloudomate	33
5.1	Completing the random user generator	33
5.2	Adapting Cloudomate to changing website layouts	33
5.2.1	Incompatible providers	34
5.2.2	Updating and adding providers	34
5.3	Dynamically sending VPS options to PlebNet	36
5.4	Adding end-to-end testing support	36
6	Quality assurance	37
6.1	Unit testing	37
6.1.1	Testing Cloudomate	37
6.1.2	Testing PlebNet	38
6.2	End-to-end testing	39
6.2.1	Proxmox	39
6.2.2	Bitpay testnet	41
6.2.3	Web API	42
6.3	Maintainability	42
6.3.1	Maintainability metric explanation	42
6.3.2	Increasing code readability	43
6.3.3	Minimising external dependencies	44
6.4	SIG feedback	44
7	Conclusions	45
7.1	Conclusions	45
7.2	Ethical Considerations	46
7.3	Reflection	47
7.4	Recommendations	47
	Appendix A Project Description	51
	Appendix B Infosheet	53
	Appendix C SIG feedback week 5	55
	Appendix D SIG feedback week 8	57

CHAPTER 1

Introduction

In recent years, privacy on the internet is becoming a more and more concerning issue. Governmental agencies are increasingly asking Internet Service Providers (ISPs) for data about their customers [1]. And even without permission they are still allowed to gather data due to the introduction of several new laws [2]. 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 in the United States [3] and Europe's new Article 13 [4].

Article 13 claims to protect digital content by mandating that all content uploaded to the internet to be monitored and filtered by machines. When content is recognised by the filter as copyright infringing, it 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 [5]. For example, videos on *YouTube* [6] 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 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. This monitoring could happen with the usage of content filters. There is too much content uploaded for manual reviews to be viable [7]. A potential alternative to manual reviews would be automated content filter. The reality is that content filter technologies are not equipped to perform context-sensitive interpretations of what does and doesn't fall under fair use [8].

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 (nodes) are connected to each other without a central server 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 more resilient. However, most P2P networks do not have measures implemented that protect their users anonymity. For example, the *BitTorrent* file sharing system [9] exposes the Internet Protocol (IP) address of nodes to other nodes in the same *BitTorrent* swarm. Such a swarm is a group of nodes that share one specific file. With these IP addresses it becomes possible to determine the identity of the nodes.

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 requested by the government.

Another way of protecting one's anonymity is by using The Onion Router (Tor) [10]. Tor works by encrypting a client's internet traffic and then routing this 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. The more relay nodes the client has to 'hop' through, the more secure the connection.

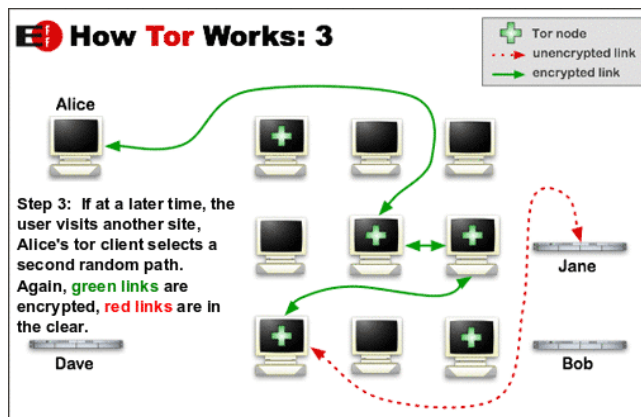


Figure 1.1: A simplified overview of Tor [11].

The last node in a relay circuit is called the exit node. This node connects the client to the server or as shown in **Fig. 1.1**, Alice to Jane. The traffic between an exit node and the server is not encrypted, unless the server uses HTTPS. HTTPS is a protocol used for secure communication over a computer network, if a server uses HTTPS the communication between server and client (if using Tor the exit node) will be encrypted. Because the traffic is relayed through other nodes and eventually through the exit node, the connection seems to be originating from the exit node, instead of the client. This implementation makes it difficult to find the source and destination of a message. All nodes are (assumably) run by volunteers. While there are a great number of relay nodes, the amount of exit nodes is limited, as can be seen in **Fig. 1.2**. This is because the activity of Tor users appear to originate from the exit node. As a result if a Tor user would perform illegal activities through a relay network it would seem that those illegal activities originate from the exit node. Because of this exit nodes are always at risk of being taken down. For a Tor network to be robust it needs multiple exit nodes, so if one exit node gets taken down a new relay circuit with a different exit node can be created. The more exit nodes in the network the more robust the network becomes. The most well-known implementation of the Tor protocol is the Tor browser [12].

With the goal of establishing a resilient privacy protecting file sharing network, Tribler [13] was developed. Tribler is an open source P2P file sharing platform developed as a research project at Delft University of Technology. Tribler combines the Bittorrent protocol [14] and a Tor-like network and thus, implements proxy layers (a proxy layer is a set of computers that relay messages between the user and the Bittorrent swarm) 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 the 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. To

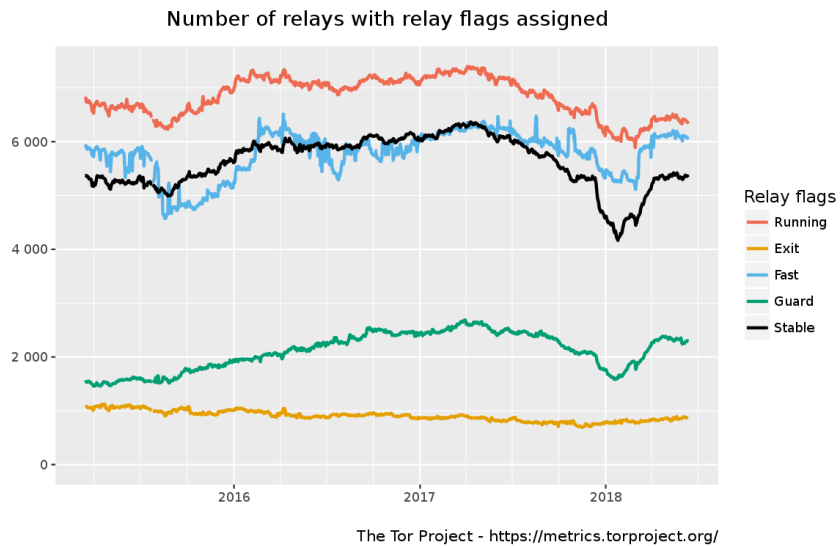


Figure 1.2: The number of relay servers as reported by The Tor Project. Note that the number of exit nodes are low in comparison to other relay nodes due to the increased risk.

solve this Tribler is implementing an incentive for people to function as exit nodes. Tribler does this by implementing a marketplace and by letting exit nodes earn Tribler tokens. These tokens have a theoretical value as they give the owner certain benefits within Tribler. These tokens can then be bought and sold for Bitcoin on the marketplace. With this it becomes possible to earn money by running as an exit node for Tribler.

The goal of this project is using this principle to create a self sustaining network of Tribler exit nodes. This network is then used to provide reliable and save anonymous usage of Tribler for those who do not want to function as an exit node. The network works by buying a private server, letting that server earn Tribler tokens by running as an exit node. These tokens are then traded for Bitcoins on the Tribler marketplace. And these Bitcoins can then be used to acquire new instances of private servers. This results in the creation of a network of autonomous self replicating exit nodes. This network is called PlebNet.

CHAPTER 2

Related work

This chapter explores the work done in previous projects. We also explain the concepts of how a network of autonomous self replicating exit nodes should function. The project did not start from scratch, as the latest version of the PlebNet botnet was provided. Therefore, the first step was to understand the working of the existing system and its assess its functionality.

First, a short summary on the history of PlebNet. The concept of an autonomous self-maintaining network started with the development of TENNET [15] by a group of students. After this, a second bachelor group [16] started over from scratch, using the experience of the first group and developed two modules called PlebNet and Clodomate. PlebNet was responsible for installing Tribler, letting Tribler run as an exit node, earn bandwidth tokens, buying a server using Clodomate and ultimately installing itself again on the newly acquired servers. After the second bachelor group finished working on PlebNet and Clodomate, a group of master students [17] continued working on Clodomate. Clodomate was implemented to select a virtual private server (VPS) provider from a selection of providers who accept Bitcoin, register an account at that provider, and buy a server with the given account. For managing Bitcoins, the Bitcoin wallet library Electrum [18] is used.

PlebNet consists of three main components: Clodomate, Tribler and Electrum. Both Clodomate and Tribler will be explained in this chapter. After which, the working of PlebNet will be explained. Electrum is not discussed in full detail as it is not part of Tribler.

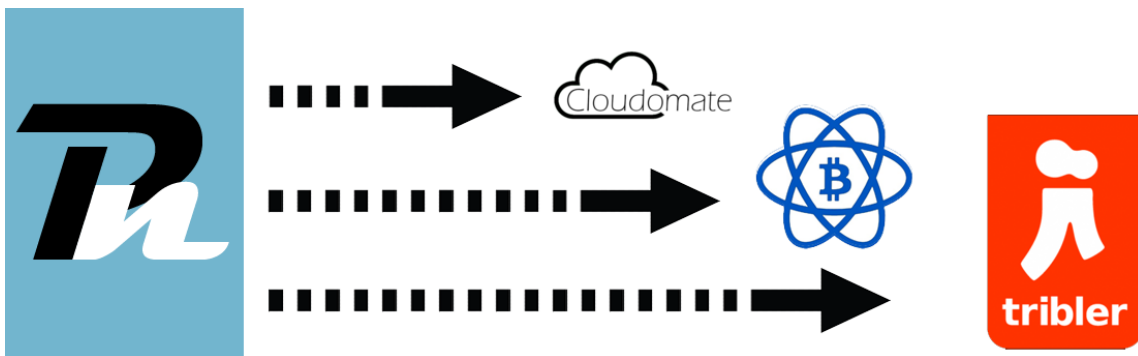


Figure 2.1: The different components that PlebNet uses.

2.1 Clodomate

Clodomate is the module that is used to facilitate the purchasing of VPSes as well as VPN protection by PlebNet. Additionally, Clodomate can be used as a stand-alone application, allowing users to purchase VPS/VPN services through a command-line interface. Clodomate gives a list of available providers and options where PlebNet can choose from. PlebNet selects an option from a provider and

gives the funding. Cloudomate purchases the chosen server and gives its login credentials to PlebNet. PlebNet then uses this server to install a new member of the botnet network on.

In general, VPS providers offer multiple options: Shared Hosting, VPS Hosting and Dedicated Hosting. Currently Cloudomate only offers the option of VPS Hosting. VPS hosting is the principle of having multiple users share the resources of a physical server. These resources being things such as memory, bandwidth, internet speed, etc. PlebNet needs relatively little resources to be able to run, so VPS Hosting is an affordable option. As the amount of resources are fixed, sharing the server with other users is not an issue for PlebNet. While it would be possible to extend Cloudomate to purchase other types of servers, such as dedicated servers, the feature is not necessary for the purpose of this project and thus not implemented. Dedicated hosting servers are servers where one account gets access to the entire physical server without having to share the resources with other users. These are more expensive than VPS hosting servers and as PlebNet does not need many resources, dedicated hosting does not offer an improvement over VPS hosting. Shared hosting is similar to VPS hosting in the sense that multiple users have to share the same server. Unlike VPS hosting, shared hosting offers no fixed amount of resources to users. This means that the resources of the bought server can be taken by other users such that PlebNet will not be able to run optimally and earn tokens. So for these reasons, VPS Hosting was the only option implemented in Cloudomate. 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 are shown for *CCIHosting* in **Fig. 2.3**.

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

Figure 2.2: VPS providers in Cloudomate.

```
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 in Cloudomate.

A secondary use of Cloudomate besides buying VPses is buying VPN protection. These VPNs can be used to hide the IP address of the server when it is running as an exit node for Tribler. This is useful because many VPS provider don't allow there servers to be used to distribute copyrighted content, this would be the case when Tribler is running as an exit node. If a VPS provider monitors the traffic over its servers, which many do, they will see that PlebNet is using their server to run as exit node. 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* [19] is supported in Cloudomate.

Cloudomate works by crawling the web pages 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 the non-functionality of Cloudomate and therefore PlebNet. However, this is the only way because no provider is offering an alternative way to purchase servers.

When Cloudomate purchases a VPS/VPN service, the user information is filled in on the registration form. This user information can either be real and entered by the user, or Cloudomate can randomly generate a user identity. Currently, when Cloudomate buys a VPS server, it leases the server 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 than a month nor does it allow for a lease to be extended. This inability to extend 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. Where an agent is an instance of PlebNet running on a server.

2.1.1 Gateways

When ordering a VPS, the transaction has to go through a payment gateway. A payment gateway is a service that providers use to authorise credit cards or direct payments. A well-known example of a gateway would be *Paypal* [20]. Cloudomate is only able to make Bitcoin transactions, so it exclusively interacts with the Bitcoin gateway of the providers. The different gateways used by each provider in the originally provided version of Cloudomate are:

- *Bitpay* [21]: this gateway is used by *Linevast*, *BlueAngelHost* and *CrownCloud*.
- *Coinbase* [22]: this gateway is used by *CCIHosting* and *PulseServer*. During the research phase of this project *PulseServer* stopped accepting Bitcoin payments all together. And while later on they reimplemented a Bitcoin gateway both providers changed to *Coinpayments* [23].
- *Blockchainv2*: this gateway was used by *UndergroundPrivate*, but they changed to the *SpectroCoin* [24] gateway.

2.1.2 Electrum

Electrum is the Bitcoin wallet library that was used in Cloudomate to purchase VPS/VPNs. The version that was used was 2.8.3, which is a Python 2.7 version. Because Tribler, PlebNet and Cloudomate are written in Python 2.7, upgrading Electrum is not possible within the allocated project period. Electrum 2.8.3 is still 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 Electrum's default wallet. If PlebNet want to buy a server, it has to pass along its own Electrum wallet to Cloudomate.

2.1.3 Functionality review of Cloudomate

Now that a general overview of the workings of Cloudomate have been given, both 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 therefore not taken into account.

Advantages

While PlebNet and Cloudomate have both been created by the same development team, Cloudomate has been developed further by multiple teams. This has resulted in a good code quality. Most of the code has been commented and a `read.me` has been provided. This means that it is easy for developer

teams to understand the inner workings of Clodomate. Clodomate offers six VPS provider options which is sufficient for PlebNet to use. And recently, Clodomate added to possibility to install VPN protection as well.

Disadvantages

There are some disadvantages with the current iteration of Clodomate. These are threefold:

- When Clodomate was further developed by other teams the compatibility with PlebNet was lost and PlebNet is unable to interact with Clodomate correctly.
- While Clodomate lists many providers all but one of them is still work. This one being *BlueAngelHost*. This problem occurs 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 PlebNet would not be able to purchase VPS servers anymore.
- There is an error in Clodomate that while the user generation itself is successful the resulting data is not saved. This means that when PlebNet buys a VPS server through Clodomate, Clodomate will generate random user data and use that to buy the server. The problem with this is that the random user data also contains the login credentials which are needed to access the VPS server. So Clodomate will buy a server but will not have the information necessary to give 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 components of Tribler for our project will be discussed here. As mentioned before, Tribler is a 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 a Tribler exit node

The goal of the agent is create as many child nodes running as Tribler exit nodes as possible. In order to do so, the agent has to generate tokens and sell them for Bitcoin, as this is required to buy new servers and install a copy of itself onto them. The only way an agent can earn these tokens is by providing specific services for Tribler. Tribler rewards people who run as an exit node or as a relay node by giving them bandwidth (MB) tokens. These tokens are a currency within Tribler and should give people benefits in the future, such as preferential treatment resulting in faster download speeds, they can have a value and can thus be sold for money. Earned tokens can be traded on the Tribler marketplace.

Before an 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 MB tokens are being earned. The amount of MB tokens 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.

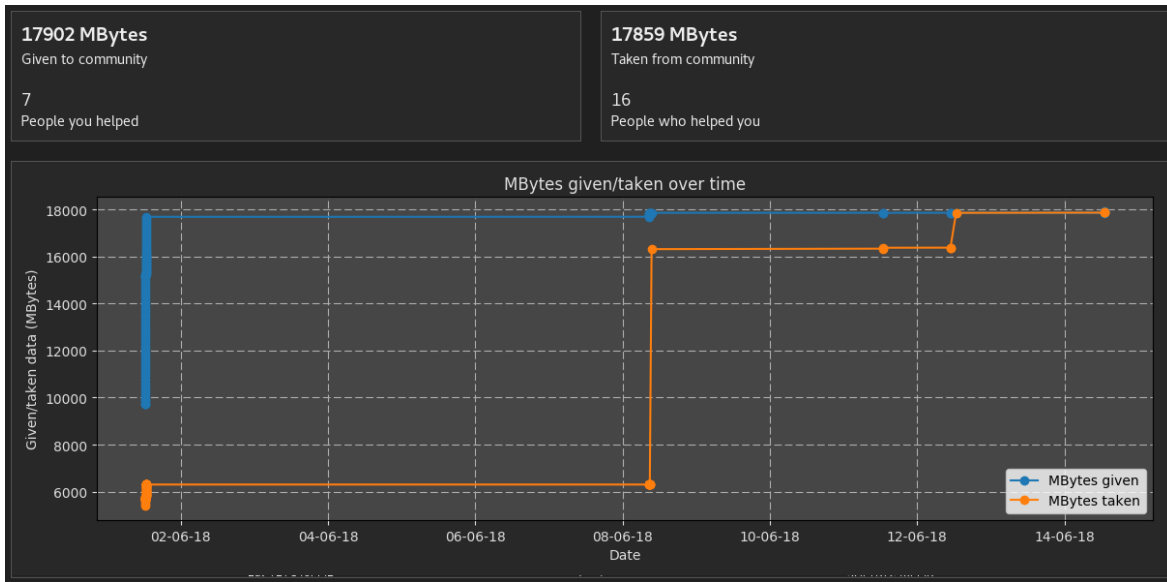


Figure 2.4: Amount of MBs given to and taken from the Tribler community.

2.2.2 The Tribler marketplace

In order for the agent to be able to buy new servers, the acquired tokens need to be sold for Bitcoins. In Tribler, an experimental version of a decentralised market was implemented, allowing users to sell their earned MB tokens for Bitcoins or buy MB tokens with Bitcoin. The market could also be tested using Testnet Bitcoins, which offers a way to easily test transactions. For now, these are the only currencies supported in the Tribler marketplace.

There are two ways for an user to buy and sell MB tokens on the market. Either via the graphical user interface in Tribler or via the http 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 receive. This works for both MB tokens and Bitcoins/Testnet Bitcoins (Testnet Bitcoins are an alternative to the Bitcoin blockchain used for testing). When Tribler is running, the http API is located at `http://localhost:8085`. The http 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, where P is the price per unit, Q the total quantity to sell, PT is the currency in which to be paid and QT is the currency which to sell:

```
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 allow the agent to make a bid or an ask. The variables are to be filled in with the desired values. In the current implementation of Tribler, all bids and asks have to matched exactly, 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 able to interact with the Tribler marketplace so that it can sell its tokens for Bitcoins to buy new VPSes.

BUYING		SELLING	
Volume	Price per unit	Volume	Price per unit
0.01 TBTC	1.0 MB	0.02 TBTC	10.0 MB
0.04 TBTC	1.5 MB	0.9 TBTC	375.0 MB
0.34 TBTC	250.0 MB		

Figure 2.5: Asks and bids on the Tribler marketplace.

2.2.3 Matchmakers

In order to place bids or asks on the market, the user needs to be connected to a special type of peer. These peers are called matchmakers and they try to match all incoming asks and bids. By default, every user can be a matchmaker. Each user has a local order book containing asks and bids of the peers it is connected to. When creating an ask or a bid, the created offer is sent to all matchmakers the user is connected to. They try to match the new ask or bid within their local order book. After a matchmaker has matched an ask or a bid, both parties are notified and they can begin the trade.

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. With the GET request below, the agent is able to verify if it has any matchmakers. When an ask or a bid is made while the agent has no matchmakers, the agent gets stuck waiting for verification that an ask/bid has been created, meaning that the agent would not continue with his routine of buying and installing servers.

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

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 it was received.

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 at least equal to the previous iteration, meaning at least one VPS per month per agent should be bought and installed. The steps necessary to achieve this are as follows:

1. Install Tribler and run as exit node.
2. Earn money via Tribler.

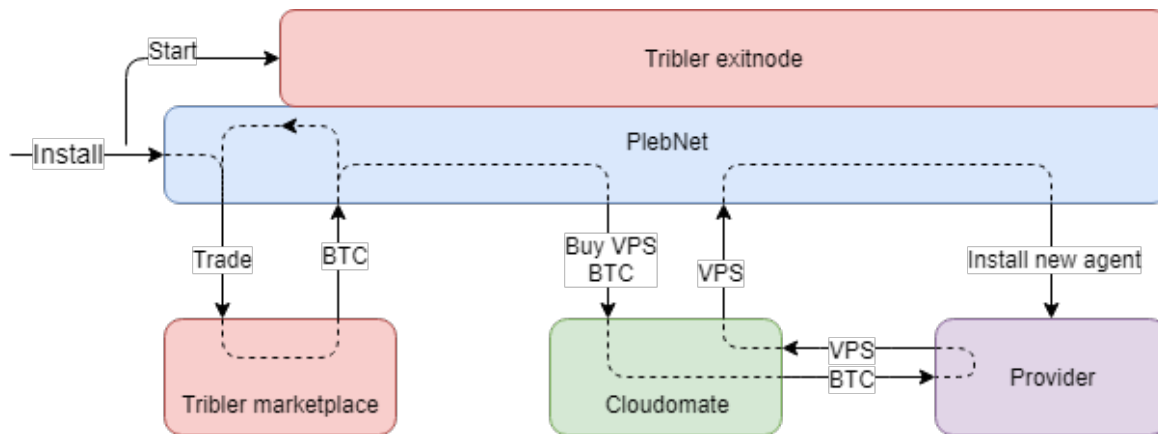


Figure 2.6: Flow diagram of PlebNet.

3. Use Cloudomate in order to buy VPSes.
4. Installation of the child server.
5. Repeat this cycle from step 2.

These steps are shown in the flow diagram in **Fig. 2.6**. After the Tribler marketplace is approached for trading, PlebNet checks whether or not it has sufficient funds to purchase a new VPN. This is shown as the split arrow in the figure.

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

- Check if Tribler is running.
- Check if the agent has enough Bitcoin to buy a new server.
- Check if a server has been bought and is ready to be configured.
- Installing new instances of PlebNet on newly bought servers.
- Letting it run the `setup` and `check` functions.

An overview of this behaviour is provided in the communication diagram shown in **Fig. 2.7**.

2.3.2 Choosing and installing offspring

As Cloudomate offers PlebNet multiple VPS providers to choose from, PlebNet has to make an informed decision on which server to buy. In the current version of PlebNet, a basic form of DNA was implemented using an basic evolutionary algorithm giving PlebNet the ability to evolve itself over future generations. Each PlebNet agent has a list of pairs 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 probability 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 as an elaborate genetic algorithm has no priority.

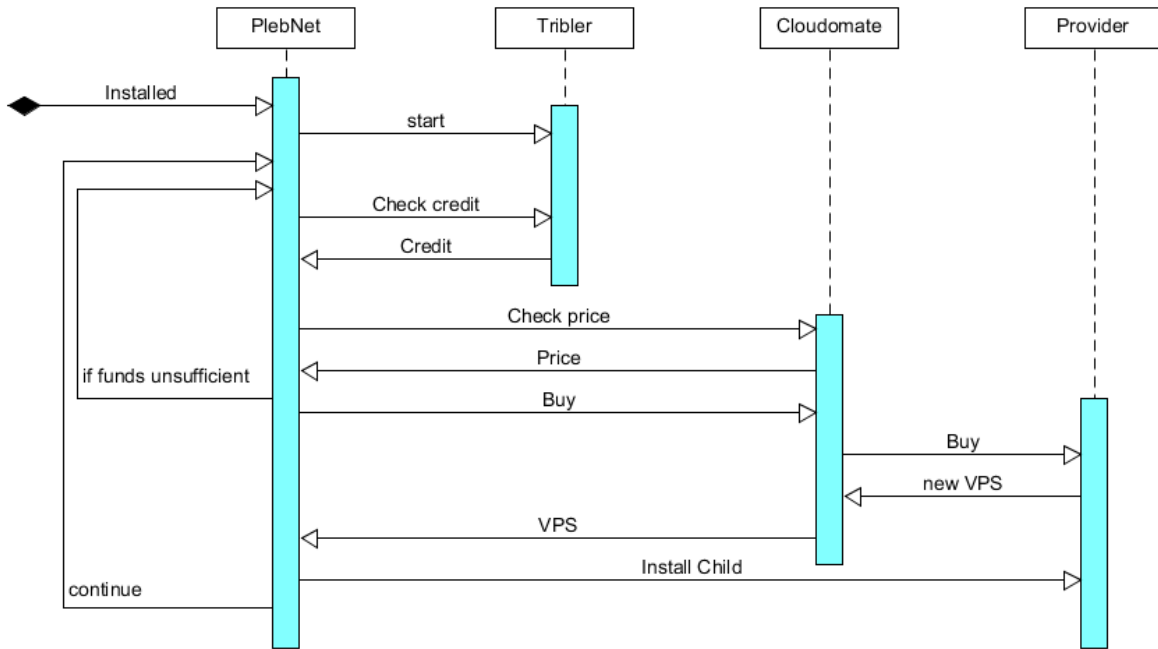


Figure 2.7: Communication diagram of PlebNet.

After having purchased a server, the agent has to install all required software. This is done via the `create-child.sh` 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 install script from GitHub and then the `install.sh` bash script is used. This script installs all needed python dependencies, downloads the complete PlebNet repository from GitHub, including submodules, and then installs PlebNet, Tribler and Clouddomate. After this is all done, a single call to the `setup` function is made. Then the Linux software utility `cron` is used for letting the agent call the `check` function every two minutes with the beneath written line of code.

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

2.3.3 Original state

The received code was not functioning anymore as written in the report. We encountered several errors while testing the functionality of the initial system. Some methods in PlebNet made calls to methods in Clouddomate which had been removed or changed by the master group working on improving Clouddomate. 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 were over 100 lines long, some comments or documentation would have helped whilst debugging the errors encountered. All in all it can be said that it was hard to get a grasp of the code's internal workings.

CHAPTER 3

Requirements analysis

After studying the related work delivered by the previous groups, the next step of the project was to compile a list of requirements for the deliverables of this project. This list would detail all features that need to be added to 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, being: *must have*s, *should have*s, *could have*s and *won't have*s. The resulting MOSCOW list with an explanation for each item is given below.

3.1 Must have

*Must have*s: these are the features which should be implemented for the project to be successful.

Fully operational end-to-end system: Both PlebNet and Clodomate have been developed by a single group after which Clodomate has been developed even further. This resulted in PlebNet not being compatible anymore with Clodomate. PlebNet has to be modified in such a way that it can use Clodomate again. This must result in a system that can earn bandwidth tokens by functioning as an exit node for the Tribler network, sell these tokens on the Tribler market for Bitcoin, automatically buy servers with the money earned via Clodomate, install itself on these new servers and repeat this cycle. To achieve this PlebNet has to be able to autonomously install PlebNet, Clodomate and Tribler as well as all the required dependencies of these three modules on the new server.

Status monitoring: Since 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 to the user in a structured manner.

Successful continuous integration in PlebNet: Continuous integration is the principal where developers integrate their code regularly into a shared repository where each integration is verified by an automated build [25]. This is useful because it allows development teams to easily detect and locate errors in the code. The tool for continuous integration that must be used for PlebNet is Jenkins CI [26]. This tool has been chosen because both Tribler and Clodomate already use Jenkins CI. Since PlebNet is reliant on both these modules, using a different continuous integration tool would only lead to more overhead for future development teams.

3.2 Should have

*Should have*s: these are the features that are desired 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 lacks essential information.

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, the current code lacks comments explaining how 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 a 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 were 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 Cloumate. PlebNet should be able to retrieve the list of available VPS providers from Cloumate. When a child agent is installed, any new providers added to Cloumate should be automatically added to the VPS options of the child agent.

Installing VPN protection: As most VPS providers do not accept their services being used for distributing copyrighted material, 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. Cloumate is already able to buy VPN protection, but PlebNet is currently unable to use it for its agents

3.3 Could haves

Could haves: these are the features that 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. As each agent has its own private wallet, these funds will be simply lost. To prevent losing this money, the agent could be able to transfer this money to a wallet managed another agent.

The ability to automatically 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, call traceback and environment information.

Genetic Algorithm for reproduction: The agent could be programmed to dynamically choose 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 the agent's lifetime and even pass this survival strategy on 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 difficult for PlebNet. This, because an agent has to purchase and run on a new VPS instance for every test and purchasing a new VPS instance for each end-to-end test is expensive. So to solve this a fully controlled environment could be set up. This allows the developers to simulate different settings and prove the concept of PlebNet. The configurations can be quickly adapted in order to improve the system.

Ability to monitor the status of the online agents real-time: 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 include the current DNA configuration, the time until VPS expiration or the amount of processed data on Tribler. This functionality should be read only, so that the agents cannot be altered or influenced by others.

Package PlebNet with its dependencies for easier installations: Currently PlebNet is installed using bash scripts. So PlebNet and all of its dependencies are now downloaded and installed sequentially. This costs time to install. An alternative to this, is delivering a pre-installed package containing all dependencies. In comparison, this would result in a relatively simple and clean installation of PlebNet and all dependencies.

3.4 Won't haves

Won't haves: these are the features that are outside the scope of this project; however, they might be considered to be implemented later on.

Adding new VPS/VPN services: This project focuses on implementing the end-to-end usability of PlebNet and improving or adding new functions. The addition of new services would fall under improving Cloudomate. For PlebNet to function it only needs a single available service, more services only improve the stability. At the start of the project these were already implemented. So it is not necessary for the functionality of PlebNet to implement this feature.

The ability to extend the lease of the current VPS: It could be possible to implement the functionality of instead of letting the PlebNet agent purchase a new VPS server, to extend its own lease (currently all VPSes 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 purchasing 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

Improving PlebNet

In this chapter, PlebNet is discussed in detail. As mentioned before, PlebNet is responsible for running a Tribler exit node on a VPS, earn bandwidth tokens (MBs) from Tribler, sell these for Bitcoin in order to purchase new VPSes and finally 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 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 can run an exit node on servers and replicate itself.

The following sections discuss the different features of PlebNet with respect to the improvements applied during the project. First, the general architectural updates are discussed, followed by sections regarding the different steps as mentioned in section 2.3.1. These steps are shown in the flow diagram in Fig. 4.1. The final sections in the chapter discuss the newly implemented features of monitoring and the implementation of VPN connections.

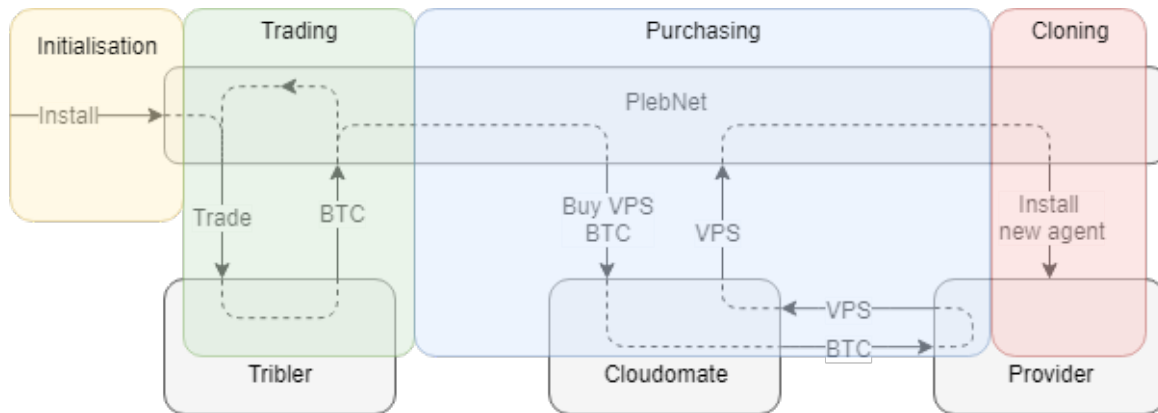


Figure 4.1: Flow diagram of PlebNet. Initialisation will be discussed in section 4.2, trading in section 4.3, purchasing in section 4.4 and cloning in section 4.5.

4.1 System architecture

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

4.1.1 Old Architecture

As the initial version of PlebNet lacked proper documentation, 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 coupling diagram was created and is shown in **Fig. 4.2**. The first things to notice regarding the structure are the unused files. These can be removed completely, without altering the functionality of PlebNet, but were probably left as the overview was lost.

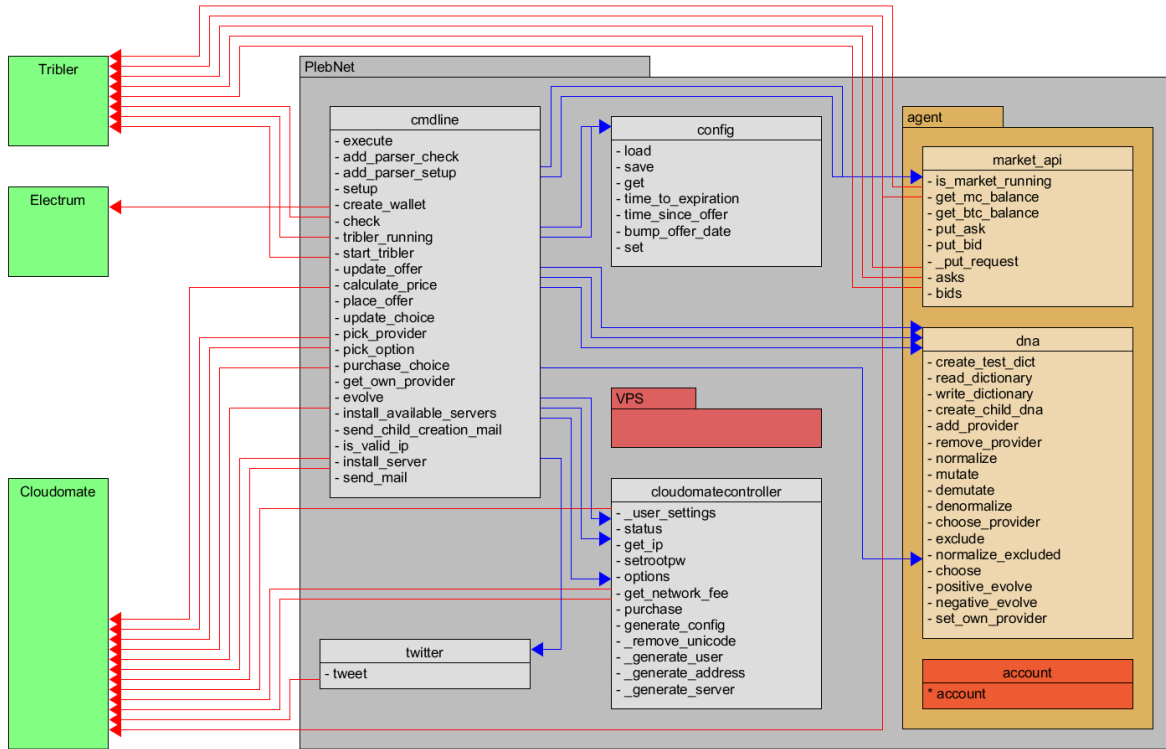


Figure 4.2: Coupling diagram of the provided code. Blue lines indicate internal and red lines external dependencies. Red files/packages indicate unused code. The amount of arrows between different files indicate the amount of calls made between the files. More lines results in a higher dependency.

The core Python file is the `cmdline.py` file. This file contains 370 lines of code 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 the Bitcoin balance and purchasing 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.

We also note that the `cloudomatecontroller.py` is not used to its full potential. It contains several methods which have nothing to do with Cloudomate and lacks other methods which are required for PlebNet. This results in multiple classes making direct calls to Cloudomate, bypassing the dedicated controller. Whenever Cloudomate is updated and calls are changed, multiple PlebNet files will be outdated and have to be adapted. If the controller is the only class with direct dependencies to Cloudomate, it is easier to keep PlebNet up-to-date. The same applies to other external dependencies such as Electrum and Tribler, but these lack a controller in the original code.

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

4.1.2 Improved Architecture

While reviewing the code written for PlebNet by the previous group, it was noted that there was no separation of concerns, some classes 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 structure: the package division. In the old structure the main folder `PlebNet/` contained two packages (`VPS` and `agent`) and multiple separate files. The `agent` package is kept, as its responsibilities are clear: handling core functions of PlebNet. The file `cmdline.py` is kept as well. This file handles the command-line input. The other files are separated or removed, which resulted in the structure shown in **Fig. 4.3**.

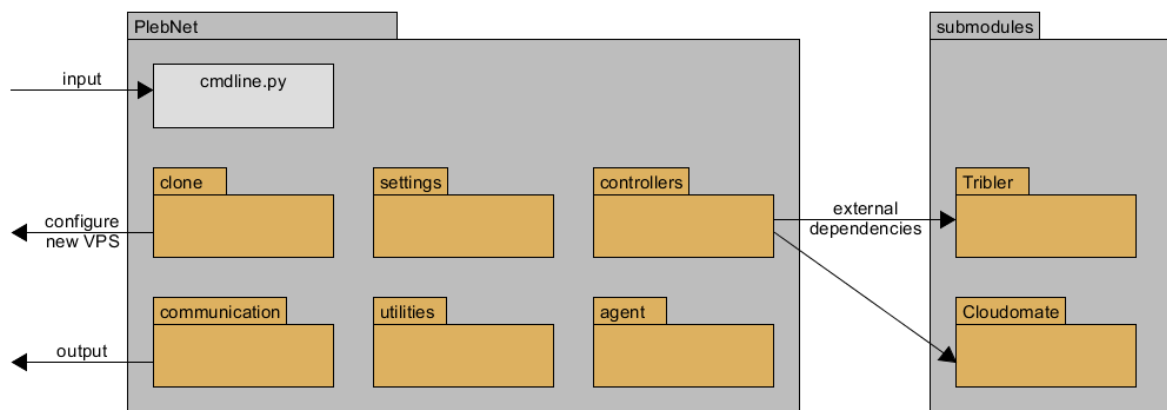


Figure 4.3: Package diagram of PlebNet.

The responsibility of every package is as follows:

- **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 Clouddomate for buying new services. The responsibility of this package is everything regarding the agent operations. This package can be considered as the core of PlebNet.
- **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 install the required Python dependencies on the newly bought server.
- **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 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 agents should not be altered by external parties.
- **The controller package** contains all code which makes calls to external submodules, in this case Tribler and Clouddomate. This way, whenever something changes in these dependencies, the only code which needs to be updated is located inside the controller package. Each of the new subcontrollers (`market_controller`, `wallet_controller` and `tribler_controller` have their own specific tasks with respect to Tribler. It was decided to split up the `tribler_controller` into multiple separate controllers, as the file would become too large (>200 lines of code).

- **The settings package** contains all methods which handle the settings of the agent. These settings include the login information for GitHub and IRC, and the use of the logger. These settings are inherited from the parent agent 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.

The updated structure results in a better understandable coupling diagram as well, as shown in **Fig. 4.4**. The `cmdline.py` is still the entry point of the entire system, but it is not the core anymore. Its responsibilities are reduced to handling the input from the command-line (while running, these are the `plebnet setup` and `plebnet check` calls) and redirect the calls to the proper packages and files.

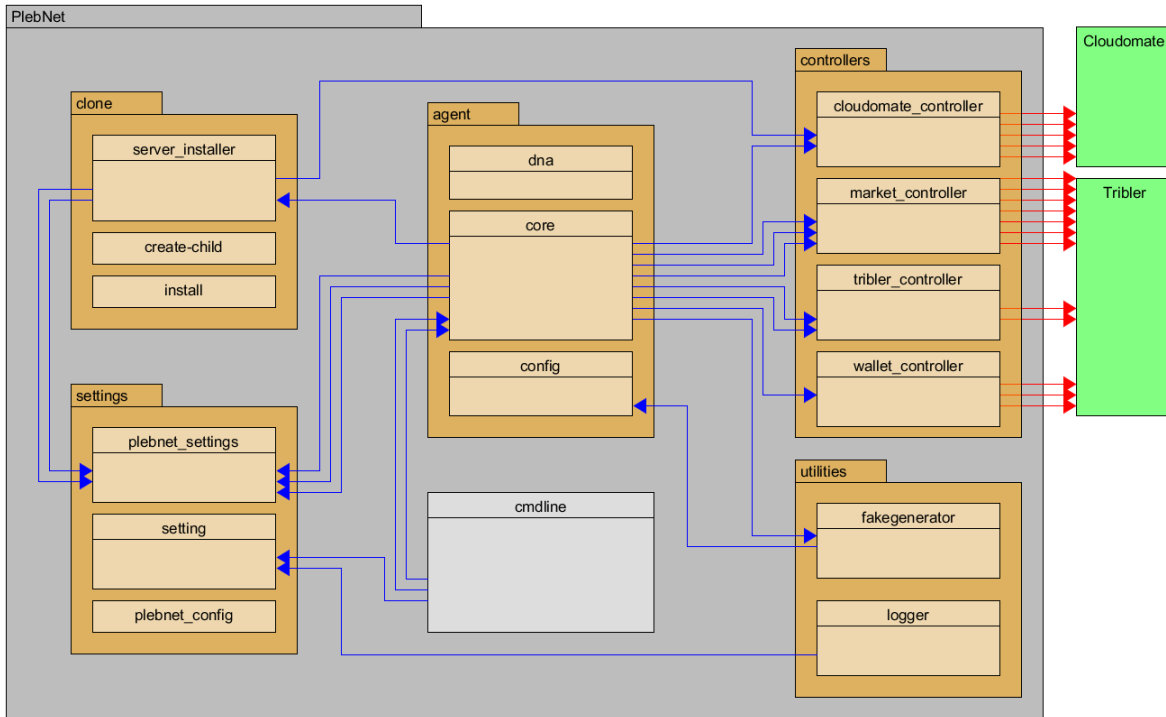


Figure 4.4: Coupling diagram of the provided code. Blue lines indicate internal and red lines external dependencies. The communication package and the dependencies on the logger are left out for readability as they are not part of the core of PlebNet. The amount of arrows between different files indicates the amount of calls made between the files. More lines results in a higher dependency.

Lower level architecture

On the file/class level the architecture did not require major refactoring efforts as the actual process of running PlebNet can be described as a linear process: check if the agent has enough funds for purchasing a VPS, make the purchase and install a new instance of PlebNet. This does not require many different classes, just many lines of code and methods.

In the provided code, the class DNA was created (`plebnet/agent/dna.py`). This makes sense as the DNA of an agent can be seen as an instance, and while handling the cloning, there can be multiple instances of DNA. However, calls made to the DNA were done in such a manner that there

were multiple instance of DNA at the same time. This resulted 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. Now, every instance is the same and updates are handled properly.

4.2 Initialisation of PlebNet

Initialisation of PlebNet encompasses the installation process of the agent, Tribler, Clodomate 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 compatible with Ubuntu 18.04 and Debian Jessie, and should be compatible with most Debian derivatives with some modifications. The project is largely written in Python2.7, along with several bash scripts for configuring servers.

4.2.1 Packaging

One of the challenges of installing PlebNet was that it has to be able to run on different operating systems. 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

The possibility of packaging code as Snaps was researched as this would allow a more stable way of installing PlebNet in a containerised fashion [27]. Unlike in a Windows operating system, where installing a program usually means executing an `.exe` file, making sure that programs such as PlebNet (including Tribler and Clodomate) install correctly on different Linux distributions is difficult to achieve. One possibility is to detect the distribution including its version number so that for each distribution a specific set of dependencies can be fetched. However, this approach is not be feasible due to the many variations of Linux distributions and would need constant management to keep the dependencies updated.

Similar to the idea of using a Docker [28] (which the 2017 BEP group attempted to use for deploying PlebNet), a Snap package, along with its dependencies is confined in a virtual filesystem as shown in **Fig. 4.5**. The management environment `snapsd` handles the installing and updating of applications.

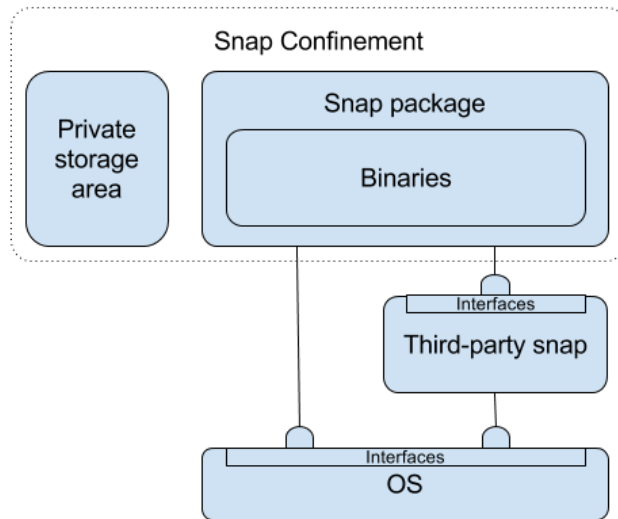


Figure 4.5: An overview of the snapd system that allows for confined Snap applications.

Because PlebNet now uses over 20 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 are fixed, resulting in a single package containing the binaries and dependencies needed to execute PlebNet. Unfortunately, Snaps works by mounting a virtual file system which will not work on most VPSes due to restricted access. This can be seen in the example in **Fig. 4.6**. Because of the limitations of VPSes, PlebNet is deployed using `bash` scripts. These scripts install the Python requirements as well as the lower-level dependencies (such as `keyring`) needed to run PlebNet and Tribler.

```
Mount snap "core" (4571) 0
Mount snap "core" (4571) 0
error: cannot perform the following tasks:
- Mount snap "core" (4571) ([start snap-core-4571.mount] failed with exit status 1: Job for sn
ap-core-4571.mount failed. See "systemctl status snap-core-4571.mount" and "journalctl -xe" fo
r details.
)
```

Figure 4.6: The virtual file system can not be mounted on a VPS.

4.2.2 Bash scripts

As mentioned before, bash scripts are responsible for installing the required dependencies and Python modules. The script `create-child.sh` is called to install a new child on a server. Several arguments can be provided when running the script:

```
Usage: ./create-child.sh <parameter> <value>
-h --help           Shows this help message
-i --ip             Ip address of the server to run install on
-p --password       Root password of the server
-t --testnet        Install agent in testnet mode (default 0)
-conf --config      (optional) VPN configuration file (.ovpn)
-cred --credentials (optional) VPN credentials file (.conf)
-b --branch         (optional) Branch of code to install from
```

When a branch is given as an argument to `create-child.sh`, the default branch in the script is updated and passed to `install.sh`. This makes sure that the agent and its children are running the code from the desired branch. `install.sh` first updates the system and installs the newest Python 2 and Pip versions. A large list of Python packages is then installed with Pip, most of these are Tribler dependencies. Finally, the script installs PlebNet, Clouddomate and Electrum and creates a `cron` job that calls `plebnet check` periodically.

4.2.3 Wallet creation

The initial version of PlebNet uses the Electrum wallet library directly for creating a wallet. After trying to make asks and bids on the Tribler marketplace, it gives an error that a Bitcoin wallet has to be created first. After some research, it became apparent that the wallet was not linked to Tribler causing Tribler to be unable to use the wallet. It was decided to use the functions within Tribler for creating new wallets, as this would result in a better integration of the wallet in Tribler. This also solves the issue of having Electrum as a direct dependency for both PlebNet and Tribler, as Tribler itself also uses Electrum to create a wallet.

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

With these PUT requests, both Bitcoin and Testnet Bitcoin wallets can be created via the Tribler http API, where `PW` is the used password.

4.3 Trading

When the agent has earned bandwidth tokens (MB) by running as an exit node it needs to sell these on the Tribler marketplace for Bitcoin. In section **2.2.2** the commands of the Tribler marketplace are explained. These commands still worked and were not changed during the project. Neither was the market strategy implemented in PlebNet changed. The 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 by 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 (asks made by the PlebNet agent expire after one hour). After this hour has passed PlebNet will create a new ask, trying to sell all of the MBs again.

This strategy is simplistic and there is a lot of room for creating a more sophisticated market strategy for PlebNet. This, however, was left as a lower priority feature to implement for two reasons. First, 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 development, meaning there are as of now no buyers that are willing to trade Bitcoin for MBs and it is not useful to implement competitive market strategies when there are no buyers available.

As mentioned before, there are currently no buyers of MBs on the Tribler marketplace, so in order to be able to sell them, buyers have to be simulated with a bot. This is accomplished with the `buybot` (located at `PlebNet/tools`). The bot works as follows:

- It queries all asks in the market.
- For each ask, it creates a bid that exactly matches.
- Tribler will create a transaction between the ask and the bid, selling all MBs from the PlebNet agent to the **buybot**.

As an agent created an ask in such a way that it should be able to buy a new server, a new VPS will be purchased as soon as an agent earns it first MB. Therefore, some limitations to the buying behaviour are implemented (time between bids and amount). The bot is able to place bids in either TBTC as well as BTC, depending on whether PlebNet is running in Testnet mode.

4.4 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 and the user information needed for purchasing is provided by PlebNet which generates acceptable identities for its children.

4.4.1 Purchasing

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

Once enough Bitcoin is accumulated, the agent invokes Cloudomate to attempt a purchase. As Cloudomate was updated and refactored before the start of this project, many of dependencies were lost. The first step was to restore the communication between PlebNet and Cloudomate. As these functions were repaired, the first purchase could be made. Since 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 below for transferring Bitcoin to the specified Bitcoin address, where A is the amount to be transferred and WA the target wallet address.

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

4.4.2 Identities

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

During the course of the project, the agent was banned multiple times and some of the purchases were denied. 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. This occurred during the implementation of a new host, as the exact approach had to be tuned for the specific provider. This is not expected to 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 email is a requirement for purchasing, but it is solved by using a functionality of gmail [29]. By using a single gmail address, for example `plebbot@gmail.com` and using the appendix `+[random_ID]`, 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 `[random_ID]` is chosen to be the username of the newly created instance. This username is generated by adding the first and last name from the identity. As the first- and last 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*, check their sales manually and still found out that we were buying automatically as shown in **Fig. 4.7**

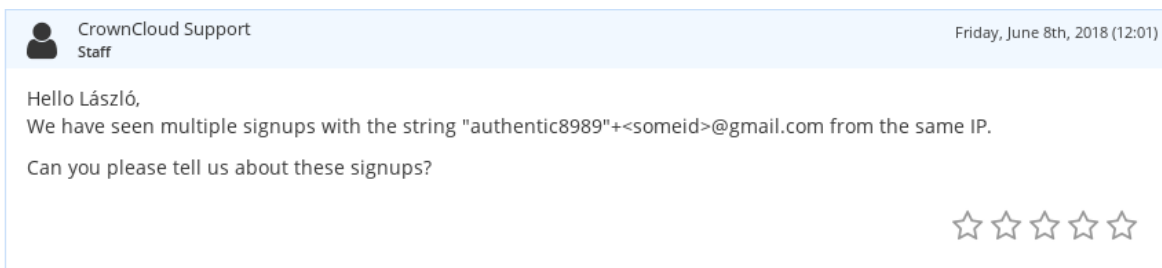


Figure 4.7: *CrownCloud* support refusing to grant access to the server because of multiple purchase attempts.

This, however, occurred as the implementation in *Clodomate* was tested. While running live, this situation would only occur, when a single agent purchases multiple servers a day at the same host.

4.4.3 Modifying DNA for the new child

This phase in the purchasing process results in an update of the agent's DNA. When a purchase is made successfully, the value for the agent's own provider is increased with a certain rate (set in the `dna.py` file). As it apparently made enough money to purchase new servers. When a purchase is not successful, the value of the provider chosen for its child is decreased as this may be due to errors from the provider changing its web page or the provider recognising bots. The exact influence of the mutation algorithm on the survival probability should be investigated, but did not fit within the time reserved for this project.

4.5 Cloning

If the purchase is successful, *Clodomate* returns the IP address of the new VPS and its root password. The server is 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 time the `plebnet check` command is called (every 2 minutes). The root password and IP address can be used to log in onto the new server via `ssh`. The first code to run on the new agent is defined in the `create-child.sh` file. This creates the required directories and install the proper certificates. The final step is to download the latest version of the `install.sh` file from the repository and execute it on the child.

4.6 Monitoring

As the program is running headless on a distant machine, inaccessible and password protected, and the current implementation of PlebNet does not support finding information regarding an agent's status and progress. This also means that 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.

4.6.1 Legacy communication

The initial version of PlebNet used both email and Twitter [30] 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 through a mail server which belonged to one of the developers of PlebNet. As the credentials for this mail server were not handed over by the last group, new options were explored. Google mail required each agent to be authenticated and was therefore not usable. Other mail servers included equal securities. Another aspect of mail communication is that it does not notify anyone if the server is banned or runs into other trouble such as runtime errors as the agent is taken down by the provider. This can be solved by implementing a better notification strategy. For these reasons it was 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 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 it was decided to remove the Twitter communication module as well.

4.6.2 IRC

An example of a solid communication network is Internet Relay Chat, or IRC [31]. 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 agent is connected, he 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 to those who are listening to the channel. This way it is possible to keep track of the agents which are still alive. 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 prevents single points of failure in the network. Contrary to sending messages to a single access account, which could be lost.

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.

```

[09:49:48] * plebbot4854 (~plebbot48@wlan-145-94-191-180.wlan.tudelft.nl) has joined
[09:50:03] thijmensjf !host
[09:50:03] plebbot4854 My host is : linevast
[09:50:10] thijmensjf !alive
[09:50:11] plebbot4854 I am alive, for 00:00:23
[09:50:59] * plebbot9242 (~plebbot92@145.94.191.180) has joined
[09:50:59] plebbot4854 IRC is still running - alive for 00:01:11
[09:51:06] thijmensjf !host
[09:51:07] plebbot4854 My host is : linevast
[09:51:07] plebbot9242 My host is : linevast
[09:52:02] thijmensjf !init
[09:52:02] plebbot4854 My init date is : 15:06:33 02-05-2018
[09:52:02] plebbot4854 IRC is still running - alive for 00:02:14
[09:52:02] plebbot9242 My init date is : 15:06:33 02-05-2018
[09:52:02] plebbot9242 IRC is still running - alive for 00:01:11
[09:54:02] plebbot9242 IRC is still running - alive for 00:03:11
[09:54:02] plebbot4854 IRC is still running - alive for 00:04:14
[09:54:15] thijmensjf !alive
[09:54:15] plebbot4854 I am alive, for 00:04:28
[09:54:16] plebbot9242 I am alive, for 00:03:25

```

Figure 4.8: IRC communication showing the usage of the commands `!host`, `!alive` and `!init`.

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.

The IRC data retrieved can be stored and interpreted to gain useful insights in the effectiveness of PlebNet. This is implemented as a `tracker_bot`. This Python script consists of two threads when running: one for listening to the IRC channel and storing the received data; the other for actively asking for data on the IRC channel after a defined interval.

The data collected by the `tracker_bot` is written to a file which contains four columns:

- A timestamp:
This stamp is set to the trackers time of reading. This timestamp can differ from the agent, as the agent can be in a different timezone. As the time from the tracker is used, it is possible to compare the timing of different agents.
- The nickname of the agent:
This can be used to identify the source. This nickname has to be unique, as is obliged by the IRC server. It is possible that a nickname is recycled, as they are randomly chosen, but this can be noted as each agent has a maximal life span of one month.
- The data type:
This is used as all types of data are stored in a single file. The data type can be one of the following: matchmakers, MB_balance, BTC_balance, TBTC_balance, uploaded, downloaded.
- The value:
The actual value of the data.

This information can be used to create an overview of the network, observe the data throughput and the tokens earned by all bots. The graphs in **Fig. 4.9** were automatically created using the `tracker_bot`. They currently only contain data from a 24 hour span, but this increases as the tracker is continuously collecting more data.

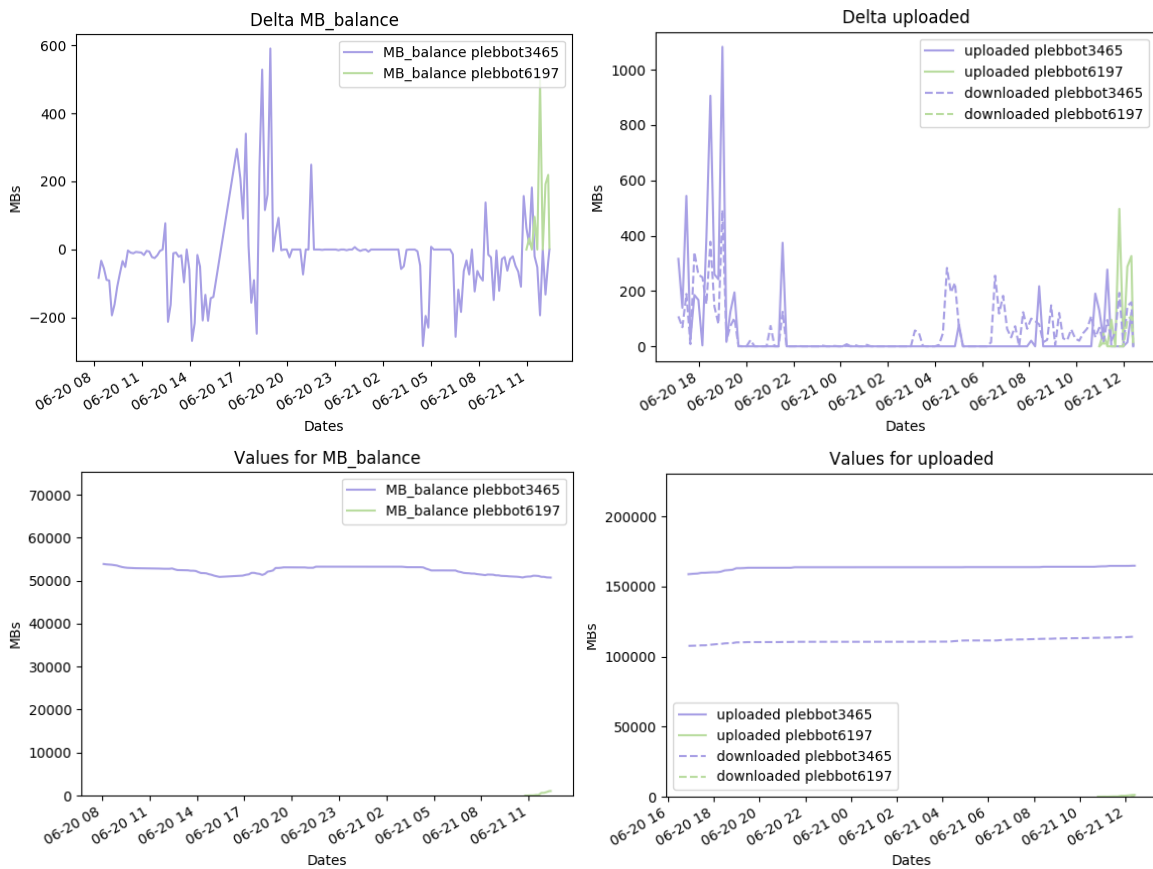
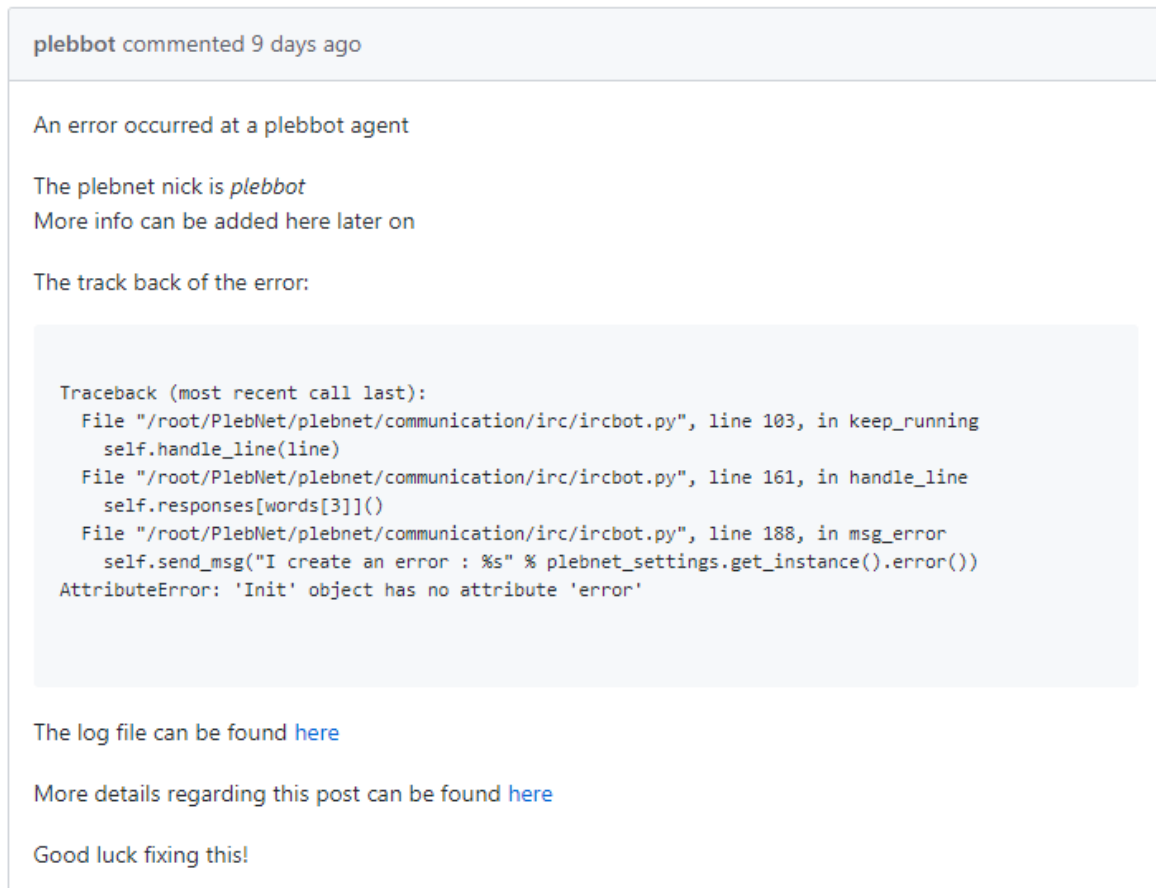


Figure 4.9: Multiple graphs created by the `trackerbot`.

The graph **Delta MB balance** shows the MBs earned or lost in a ten minute interval. It can be seen that there are times when the agent loses hundreds of MBs. This is also visible in the second graph: **Delta uploaded**, which shows an increase in downloads just after midnight, while the uploads are negligible. The bottom two graphs show the total accumulated MBs, uploads and downloads. The behaviour which was seen in the first two graphs results in a slight decrease in the amount of MBs over time. This behaviour was noted multiple times during the project and caused some of the agent to have negative amounts of MBs. There is not much the agent behaviour can do to change this, as it is relying on Tribler to upload more than it downloads.

4.6.3 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 a 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. 4.10**.



plebbot commented 9 days ago

An error occurred at a plebnet agent

The plebnet nick is *plebbot*
More info can be added here later on

The track back of the error:

```
Traceback (most recent call last):
  File "/root/PlebNet/plebnet/communication/irc/ircbot.py", line 103, in keep_running
    self.handle_line(line)
  File "/root/PlebNet/plebnet/communication/irc/ircbot.py", line 161, in handle_line
    self.responses[words[3]]()
  File "/root/PlebNet/plebnet/communication/irc/ircbot.py", line 188, in msg_error
    self.send_msg("I create an error : %s" % plebnet_settings.get_instance().error())
AttributeError: 'Init' object has no attribute 'error'
```

The log file can be found [here](#)

More details regarding this post can be found [here](#)

Good luck fixing this!

Figure 4.10: 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
```

4.7 VPN

As an additional functionality, a VPN is installed on the purchased server. The VPN protects the server from DMCA claims to which most providers unfortunately respond to, by shutting down the server. A DMCA claim, is a complaint made if someone suspects a website of copyright infringement. The Digital Millennium Copyright Act (DMCA) is a set of laws that exist to protect copyrighted content. The VPN purchasing functionality was previously implemented in Clouddomate, but the process of installing it by the agent still needed to be implemented.

Installing a VPN as it turned out poses 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* [32] to attach to it.[33] TUN/TAP is generally not enabled on servers by default, as can be seen in **Fig. 4.11**, and may not even be allowed by some providers. 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, Clouddomate is to be modified to access the control panel and change the TUN/TAP options.

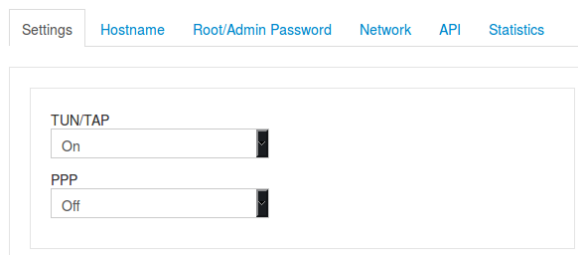


Figure 4.11: The TUN/TAP option in *Linevasts* control panel.

The VPN feature is added as a *proof of concept* due to limited time in the later sprint and the fact that VPN protection is a *should have*. *Linevast* was chosen as the provider for VPN functionality.

4.7.1 Gaining access to the control panel

Like most providers, *Linevast* sends an email containing the user information. The client panel offers the ability to read emails as shown in **Fig. 4.12**. In Clouddomate, the class `Linevast` is extended to parse these emails. The class `Linevast` is also extended to have access to the control panel, from where TUN/TAP as well as other settings can be changed by sending an `AJAX POST` request. This implementation could be extended to all *SolusVM* providers to enable the TUN/TAP option if allowed by the provider. Changing *Linevast's* control panel settings with Clouddomate was made possible by studying the control panel's source code; the ease at which this can be done for other providers depends on the implementation of the control panel. `POST` requests to the provider's server usually require some kind of `id` to be sent along with the request.

4.7.2 Handling VPN from PlebNet

As mentioned before, providers are quick to shutdown servers after receiving a DMCA complaint. Therefore, the agent should purchase a VPN for each of its children. The time it takes for a VPS to be activated differs from the time it takes for a VPN to be activated. The agent first purchases a VPS if it has sufficient funds, after a successful purchase, the agent will attempt to purchase a VPN. When a new child is to be installed on the server, the VPN configurations are fetched. The child will only

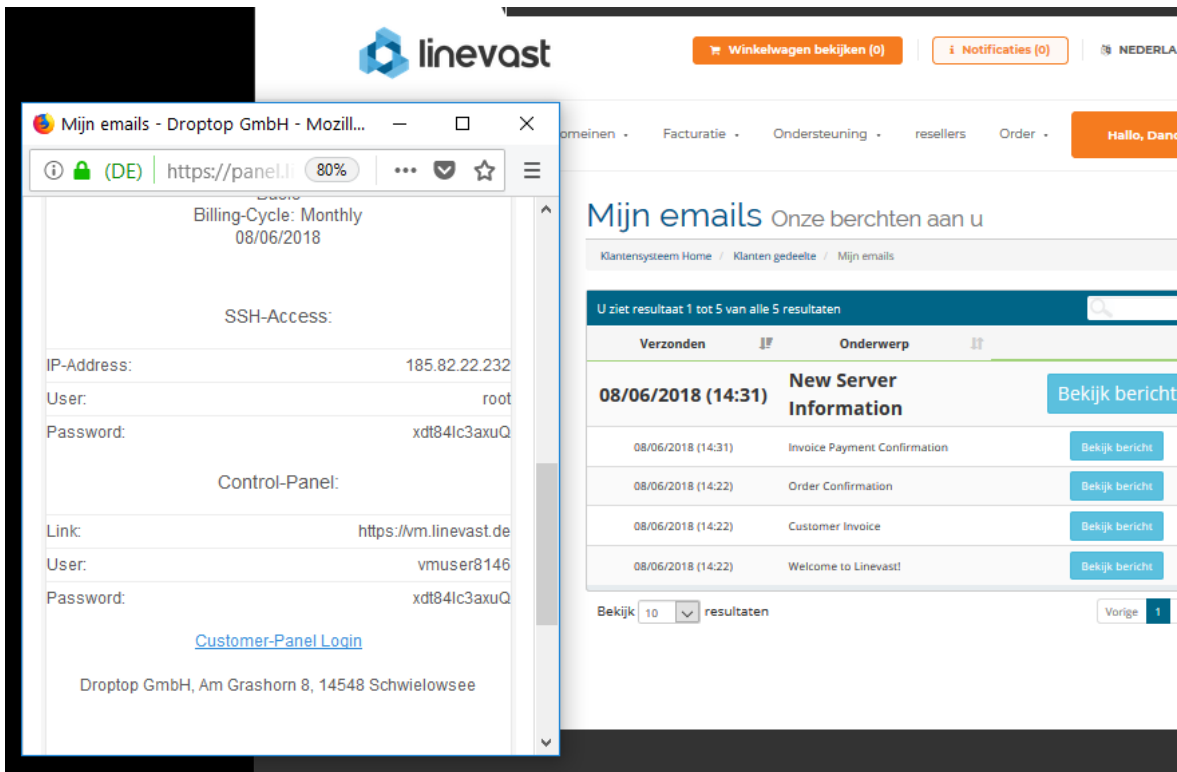


Figure 4.12: Emails can be read from within the client panel. Therefore, Clodomate is able to gain access to the control panel by parsing the credentials provided in the email.

be installed if a VPN configuration has been successfully fetched. The VPN configuration consists of an `.ovpn` file containing the configuration for OpenVPN and a `credentials.conf` file, containing the credentials with which the VPN was purchased.

On the child server, the VPN configuration files are searched in the `check` function of PlebNet and installed if present. The agent should ideally be stricter when checking for VPN configurations and only continue operating when the configuration files are found. Currently, due to limited time, only a warning is given when the configuration files are not found.

CHAPTER 5

Improving Clodomate

In this chapter, we elaborate on our improvements made to the Clodomate module. As mentioned before, Clodomate 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 Clodomate. However, to make PlebNet fully operational, certain changes have to be made to Clodomate.

The changes are classified in four categories. The first category is to repair the random user generation in Clodomate, The second category is repairing the VPS options. The third is to have Clodomate give its VPS options dynamically to PlebNet. And the final change is to add Testnet support to Clodomate. The following sections elaborate on these changes.

5.1 Completing the random user generator

For Clodomate to be able to purchase a server, it needs to fill in a user form. In order to be able to do so, Clodomate needs user information. A user can give his information to Clodomate or Clodomate can generate random user information for them. This second option is the option that the initial version of PlebNet used. However, there is an error where Clodomate creates random user data and uses it to buy a server, but Clodomate does not save this user information. This is a problem because the generated user information includes the root password and username needed to gain access to the purchased server. This means that Clodomate will purchase a server but has no access to it. Normally this information can be salvaged as most VPS providers sent the username and root password in an email; however, the email address is also randomly generated by Clodomate. This problem is solved by storing the user information in a configuration file which can then be read when the information is queried by either the user or the PlebNet agent.

5.2 Adapting Clodomate to changing website layouts

The goal of Clodomate is to purchase VPS providers for PlebNet to use as agents. However, many of the VPS providers implemented in Clodomate 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 Clodomate to buy VPSes of these providers. At the start of the project Clodomate had six VPS providers implemented these were: *Linevast*, *BlueAngelHost*, *CCIhosting*, *CrownCloud*, *PulseServer* and *UndergroundPrivate*. Of these six only *BlueAngelHost* still works for Clodomate making PlebNet completely reliant on one VPS provider. If that provider were to make changes PlebNet would not be able to replicate itself anymore. 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.

5.2.1 Incompatible 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, which 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 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 identification. 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 server. The way *Coinpayments* handles this invoice URL is fundamentally different from different gateways. Take *Bitpay* as an example: there the invoice can be added to the URL itself and an invoice URL would look like this:

```
https://bitpay.com/invoice?id=exampleID.
```

In *Coinpayments* this is not the case as their invoice URLs look like:

```
https://www.coinpayments.net/index.php.
```

Visiting this URL, one will be redirected to the main page of *Coinpayments*. Here the invoice identification is given as a secret token and the page has to be accessed with that token to be able to see the invoice. The Python library Cloudomate uses to access these URLs, `beautifulsoup`, does not have a way implemented to access these URLs with a secret token. It is for that reason that if Cloudomate makes an order to *CCIhosting* or *PulseServer*, it is not able to access the invoice URL and therefore it cannot retrieve the address and amount it needs for the transaction. Therefore Cloudomate is unable to use these two providers.

The other provider, *Crowncloud*, was not fixable for a different reason, which was also a problem with which previous development group had trouble. This problem is that it is possible to purchase a *CrownCloud* server, but *CrownCloud* does not provide the buyer an option to set the root password during purchasing. So when Cloudomate purchases a *CrownCloud* instance, Cloudomate generates and saves a root password, but *CrownCloud* creates its own root password. This means that the root password in Cloudomate is not correct and cannot be used to access the purchased *CrownCloud* server.

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 to retrieve the root password and set this password in the configuration file Cloudomate. Or the email would have to be scraped to find a link to the *CrownCloud* control panel. In this control panel a new root password for the bought server should be set. Both options require new scraping and form filling methods that are not implemented in Cloudomate. So for both ways to fix *CrownCloud* many methods have to be written specifically for this VPS provider and therefore it was decided to leave *CrownCloud* and focus on other VPS providers.

5.2.2 Updating and adding providers

Next the VPS providers that have been repaired and the ones that have been newly added will be discussed. These providers were *Linevast*, *UndergroundPrivate* and a new provider called *2Sync*. First *Linevast* will be discussed.

Linevast was relatively easy to fix. They were fully operational for most of the project, but at some point *Linevast* decided to change the main web page which listed all its VPS options. This made it impossible for Cloudomate to read the VPS options which included the purchase URLs for each option. Without these URLs it is impossible for Cloudomate to purchase a VPS option. After repairing the scraping methods, Cloudomate was able to retrieve these options again and after that Cloudomate was able to purchase *Linevast* servers.

The next provider is *UndergroundPrivate*. They had changed their Bitcoin gateway from *BlockChainv2* to *SpectroCoin*. To fix this, two things had to be done. Firstly when a VPS is ordered the link to the gateway was not automatically given. The page has to be loaded first and after five seconds of loading the user is automatically sent to the gateway invoice. Cloudomate uses `beautifulsoup` for webscraping and that library is a static browser, so automatically loading does not work. Once an order is made a button is displayed on the webpage and with a simple on click function `beautifulsoup` can proceed to the gateway. The second issue which had to be fixed was that once Cloudomate could access the invoice URL, it had to scrape the relevant information. After implementing this, it was possible to buy servers from *UndergroundPrivate* again.

Finally a new provider was added to ensure enough alternatives: *2Sync*. *2Sync* is a website that offers not one, but many types of VPS providers. Currently Cloudomate only has the Ukrainian VPS provider implemented. *2Sync* was chosen because the website design and forms used are similar to other already implemented VPS providers. Implementing *2Sync* itself did not cost a lot of extra work. Only the Ukrainian option was implemented as it uses a simple, self designed gateway from which the relevant information regarding the payment could be easily scraped, as can be seen in **Fig. 5.1**. This was not the case for many other *2Sync* providers, which used more complicated gateways such as *Coinpayments*.

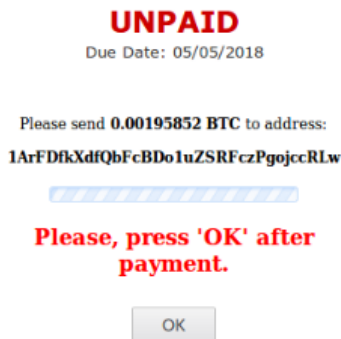


Figure 5.1: *2Sync* gateway.

At the start of the project six VPS providers were implemented, but not all were functional. After repairing and adding VPS providers, there are now four fully functioning VPS providers in Cloudomate. These are:

- *Linevast* <https://linevast.de/>
- *BlueAngelHost* <https://www.nleuangelhost.com/>
- *UndergroundPrivate* <https://www.undergroundprivate.com/>
- *2Sync* <https://www.ua.2sync.org/>

5.3 Dynamically sending VPS options to PlebNet

The problem with the delivered version of the previous development group was that the VPS options were hard coded both in PlebNet and in Cloudomate. 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 updated the options in PlebNet. This could lead to instances where 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. With this list PlebNet will stay in sync with Cloudomate.

5.4 Adding end-to-end testing support

One of the major additions to PlebNet is implementing end-to-end testing. Part of the end-to-end process of PlebNet is buying a server through Cloudomate. In order to implement end-to-end testing certain additions had to be made to Cloudomate.

The first update to be made is the addition of a mocking host: Proxhost. This can be done without any scraper, as the created host can be simply called with a proper json. The second addition is adding Testnet support for Cloudomate. 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 from 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 6

Quality assurance

In this chapter the quality assurance is discussed. The quality assurance in this project consists of two parts: testing and maintainability.

After refactoring PlebNet, both PlebNet and Clodomate had to be tested. As the main focus was on PlebNet, it was decided that the existing failing tests of Clodomate had to be fixed and no new tests would be written for Clodomate. PlebNet was not tested at all, so it was decided to create unit tests which test the features of PlebNet independent of each other as well as creating an end-to-end test for the system as a whole. Keeping track of the project was done using Jenkins as a continuous integration tool [26].

The second part of quality assurance is improving maintainability. In this project only the maintainability of PlebNet was improved. This was done because the focus of the project was on PlebNet and the code of Clodomate was already well maintained. Maintainability in this project is defined as a combination of two things these are writing comments for the code and minimising the external dependencies of the project.

6.1 Unit testing

6.1.1 Testing Clodomate

Certain tests that used to work on Clodomate failed at the start of this project. So one of the early priorities was to fix these tests to ensure that all Clodomate tests would pass and that the Clodomate Jenkins build would succeed. The tests that have been successfully fixed are as follows: First the way that Bitcoin URLs are split up had to be changed. This was done wrong and would lead to the Bitcoin address being null. Second the email had to be changed to not use @email.com as affix anymore, as this type of email address was blocked by multiple VPS providers. Clodomate now uses `usesauthentic9898 + someID @gmail.com` as its email. And finally the command line test class had to be fixed. This test class had an issue with mocking, which caused other test classes to fail. Some tests were not fixable: The tests for the *Coinbase* gateway failed as the test URL was not supported anymore. To fix this, a new test URL had to be set up, but this was not done as none of the providers used the *Coinbase* gateway anymore (more details about the gateways can be found at section 2.1.1). Therefore this 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 payment function was mocked, so that the tests would not actually purchase a VPS server. The order was still made at the VPS providers websites and if the test is run multiple times, this would result in multiple unpaid orders from the same IP address with roughly the same email address. This could lead to the VPS providers banning the IP/email address or changing their layout. To prevent this from happening these tests are now run anymore.

6.1.2 Testing PlebNet

As there were no tests written for PlebNet by the previous group and the main architecture was planned to change, it was decided to wait with testing until after 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. Writing the tests did not cause any major problems that could not be easily fixed.

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. The project is intended to run as root and uses several directories for storing data such as log files and configuration files. It had to be ensured that these directories are correctly called.

The creation of a virtual environment on Jenkins did cause a numbers of problems. The environment was required to be able to run the tests in the Jenkins environment and install all required Python dependencies, but proved to be harder than expected. This can also be a result of a lack of sufficient experience in Jenkins.

As can be seen in **Fig. 6.1**, the setting up of Jenkins took some trials. The graph shows the coverage percentage per build. As can be seen, there are some drops in coverage which are caused by the problems described above, while the general trend is an increasing line. The total coverage achieved for PlebNet is 85%. More details can be seen in **Fig. 6.2**.

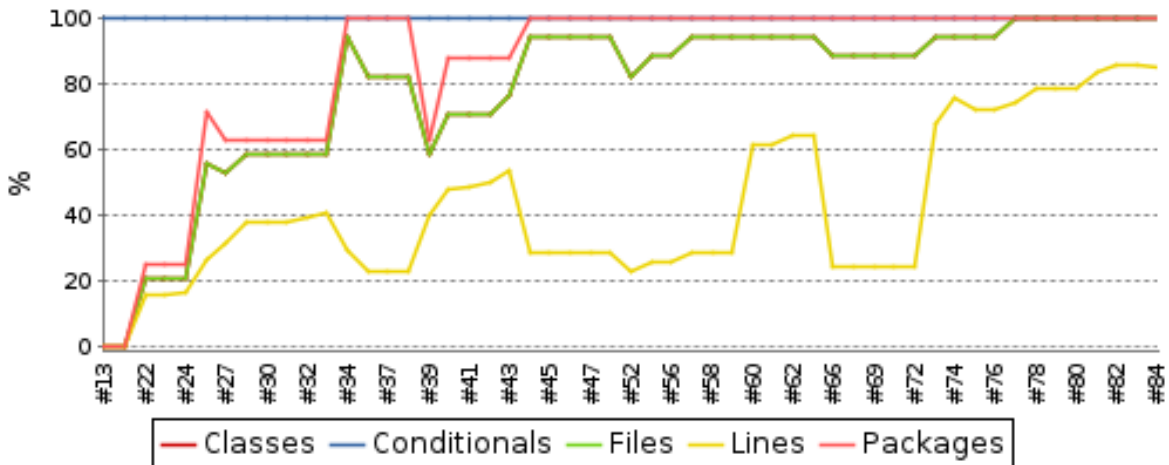


Figure 6.1: Coverage percentage per build.

Project Coverage summary

Name	Packages	Files	Classes	Lines	Conditionals
Cobertura Coverage Report	100% 8/8	100% 16/16	100% 16/16	85% 924/1085	100% 0/0

Coverage Breakdown by Package

Name	Files	Classes	Lines	Conditionals
plebnet	100% 1/1	100% 1/1	93% 76/82	N/A
plebnet_agent	100% 3/3	100% 3/3	85% 206/241	N/A
plebnet_clone	100% 1/1	100% 1/1	70% 37/53	N/A
plebnet_communication	100% 1/1	100% 1/1	19% 11/58	N/A
plebnet_communication IRC	100% 2/2	100% 2/2	84% 147/174	N/A
plebnet_controllers	100% 4/4	100% 4/4	94% 262/278	N/A
plebnet_settings	100% 2/2	100% 2/2	86% 87/101	N/A
plebnet_utilities	100% 2/2	100% 2/2	100% 98/98	N/A

Figure 6.2: Coverage report on Jenkins.

6.2 End-to-end testing

To fully test PlebNet not just Unit tests have to be written but end-to-end tests have to be performed as well. In PlebNet end-to-end testing is done by using a Proxmox server to emulate the VPS providers. The end-to-end test is initiated by placing PlebNet 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. And as explained in Chapter 6 this repairing of Cloudomate VPS providers can lead to difficulties. So for those reasons Proxmox was chosen to emulate VPS providers.

6.2.1 Proxmox

The Proxmox server is managed via Ansible scripts. Using Ansible, creating, cloning and restoring containers can be done automatically (**Fig. 6.3** gives an overview of this). The backend consists of three parts: the web API, used for managing bought containers; the `Flask` application which includes the payment controller and the Ansible scripts.

Ansible

The Ansible scripts contain tasks that create Debian or Ubuntu containers and set up the network including `Peervpn`. To emulate the process of setting up a container when a Testnet purchase succeeds, the Ansible scripts are executed using Python. Most container settings, such as disk space, CPU cores and memory size are defined in the `ansible/create_container.yml` script, the varying parameters `ct_id`, `ct_kind` (distribution: Ubuntu or Debian) and `ct_root_password` are dynamically set by ProxHost. The class `AnsibleRunner` is used to call Ansible scripts asynchronously. `ansible.cfg` is the configurations file used, the host settings (IP, `root password` or `ssh key`) are stored in the `hosts` file. Ansible 2.5.x is used to run the Ansible scripts, version 2.7.x is not recommended because of a bug causing the container creation script to fail during a waiting task.

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 access to individual containers, assigning a unique IP address to each container more realistic as far as trying to emulate a real provider.

In order for peers to be found, a relay peer needs to be running constantly. The relay peer can be initialised with the following configuration:

```
port 7000
networkname proxhost
psk 123123
enabletunneling yes
enablerelay yes
interface peervpn0
ifconfig4 20.10.0.1/24
```

The `enablerelay` option allows for the containers behind a NAT to be visible to each other. The last option, `ifconfig4` determines the IP of the computer within the network.

To connect to the Peervpn network (as a user or container), the following configuration file is used:

```
port 7000
networkname proxhost
psk 123123
enabletunneling yes
interface peervpn0
ifconfig4 20.10.0.50/24
initpeers <relay peers public ip> 7000
```

To connect to the network, execute Peervpn along with the correct configuration: `./peervpn configuration.conf`

In the end-to-end repository (also found in PlebNet as a submodule in the `tools` folder), a script (`getpeervpn.sh`) is provided, which automates the process of downloading and setting an IP address.

Flask

The *Flask* application (`connect.py`) provides access to the management section and contain routes which are accessible by Cloudomate, allowing for purchasing and status requests. A total overview of the end-to-end system is provided in **Fig. 6.4**

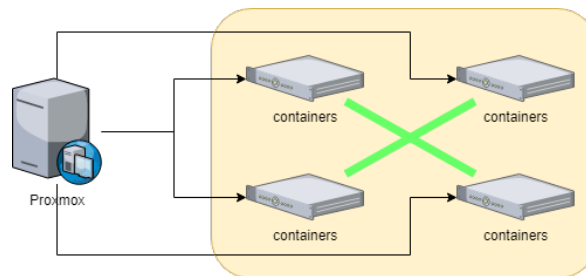


Figure 6.3: Overview of the Proxmox server. Containers are connected with each other through *PeerVPN*.

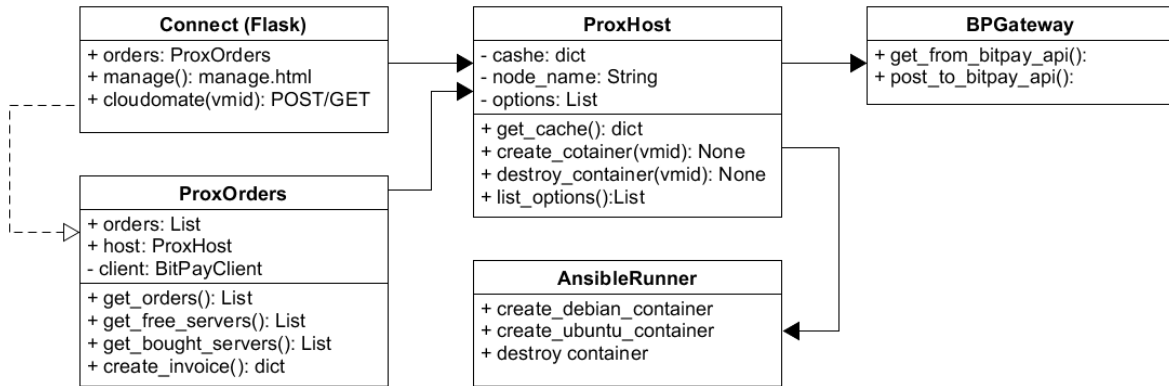


Figure 6.4: A simplified UML diagram of the end-to-end system. The Flask component "Connect" allows Cloumate or web users to purchase containers. When a purchase is made, ProxOrders creates an invoice which Connect returns as a HTTP response. ProxHost can either create or destroy containers by creating an AnsibleRunner object with the appropriate parameters.

The application also provides a web API for managing containers.

6.2.2 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, the *BitPay* library for Python2.7 was used. Each machine hosting *ProxHost* needs to first have a token verified from the *Bitpay* control panel. A screenshot of the *Bitpay* merchant page can be seen in **Fig. 6.5**. Because many providers implemented in Cloumate use *Bitpay* as their Bitcoin payment gateway, *Bitpay* is chosen as the gateway for emulating the payment process.

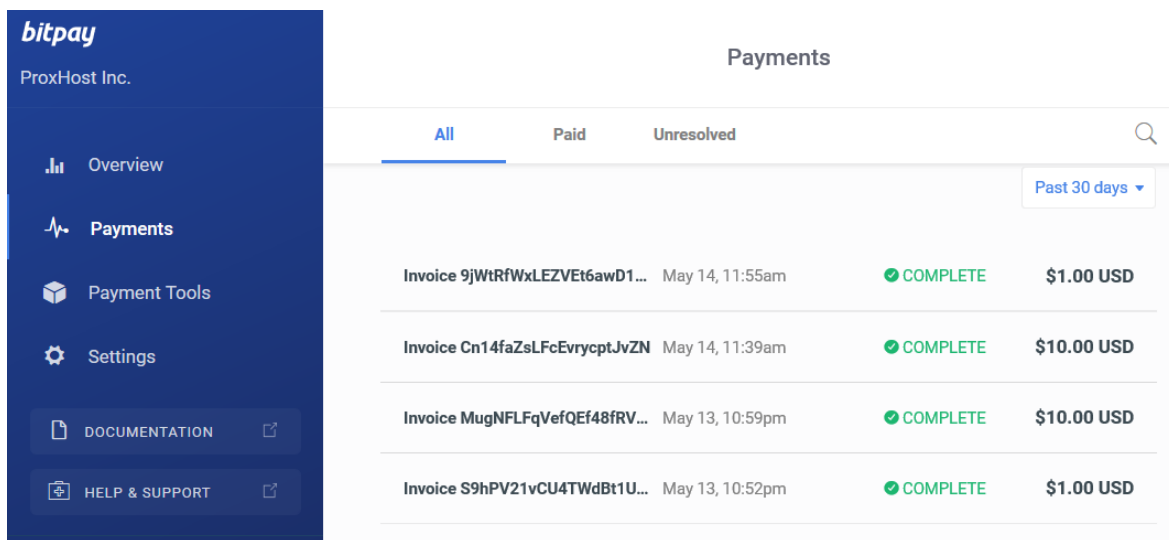


Figure 6.5: The testnet Bitpay merchant page. Invoices can be created and paid in testnet Bitcoins.

6.2.3 Web API

A minimal web API (shown in **Fig. 6.6**) was created to destroy containers bought by Cloudomate. Additionally, containers for testing can be created and destroyed. Due to time constraints, the web API is not securely configured. A self-signed SSL certificate was used as some functionalities needed HTTPS to function. Ideally, the SSL certificate should be properly signed. *Let's Encrypt* [34] could be used for this purpose. Furthermore, the `/manage` access point uses a simple *Flask* authorisation implementation where the username and password are stored in plain text. This could be improved by hashing (and salting) the password. For the purpose of this project, the security of the end-to-end is not a priority and kept simple.

ProxHost Manage

status	online destroy	online destroy
vmid	101	110
name	proxhost-ubuntu-101	proxhost-ubuntu-110
order_time	18-06-01 12:30:37	18-06-01 09:02:48
ip	20.10.0.101	20.10.0.110
username	demo	fox
note	turned everything off, but keep for data Add Note	Add Note
expiration	2018-07-01	2018-07-01
distro	ubuntu	ubuntu
root_password	plebnet	plebnet

Get a server

Figure 6.6: The minimal web API provides an easy way to destroy and create containers.

6.3 Maintainability

Aside from testing, quality assurance is also about improving the maintainability of PlebNet. There are multiple ways to define maintainability, but for this project it is defined as improving the code comments as well as minimising the external dependencies of the project. In this section it is first discussed why this metric of maintainability was chosen over others second the increase of code comments will be explained and finally the minimising of the external dependencies are discussed.

6.3.1 Maintainability metric explanation

There were alternative maintainability metrics considered for this project. The biggest one considered was the maintainability index. The maintainability index is a relatively old metric that combines three types of metrics into one. These three are lines of code (*LOC*), the cyclomatic complexity (*CC*) of the code and the Halstead volume (*HV*).

Cyclomatic complexity is a software metric that can be used to determine the complexity of the code. Cyclomatic complexity works by making a Control Flow Graph. A Control Flow Graph is a graph

that measures the number of linearly-independent paths through a program module. The number of linearly-independent paths in this Control Flow Graph is equal to the Cyclomatic complexity. The higher the cyclomatic complexity is, the higher the complexity of the code and risk of errors in the code.

The Halstead volume is a software metric that counts the number of operators and operands in the code. Operators are the special tokens that represent computations like addition, multiplication and division. The values the operators works on are called operands. Halstead volume uses this information to estimate the difficulty (the difficulty to understand the program), the effort (the time it takes to write code) and the number of bugs in the code.

Maintainability index measures these three metrics and combines them through the formula:

$$\text{Maintainability Index} = \text{MAX}(0, (171 - 5.2 * \log(HV) - 0.23CC - 16.2\log(LOC)) * \frac{100}{171})$$

This metric type was ultimately not used for two reasons. First, the metric is relatively old and researchers have criticised it. [35] Second, the metrics used in the maintainability index were things that were already well implemented in PlebNet. The maintainability index scores a project from 0 to 100 and every score of 20 and above is considered well maintainable. The old version of PlebNet had a minimum score of 44.15 which falls within the highest class of maintainability and therefore is a bad metric to measure improvement of the code as it was not maintainable (the new PlebNet version has a minimum score of 60.31). During the research phase there were two things discovered which did need improvement. These were the lack of code comments and the many external dependencies of PlebNet. So to make the code more maintainable these two things are considered for improving the maintainability.

6.3.2 Increasing code readability

The first metric that had to be improved was increasing the number of comments in the code. This is important for maintainability as comments are used to explain the functionality of classes and methods in the code. With these comments it becomes easier for new developers to gain an understanding of the workings of PlebNet. The old version of PlebNet had relatively little code comments which made it difficult to gain an overview of. In **Fig. 6.7** an overview of percentage code comments with the old version and with the new version is shown.

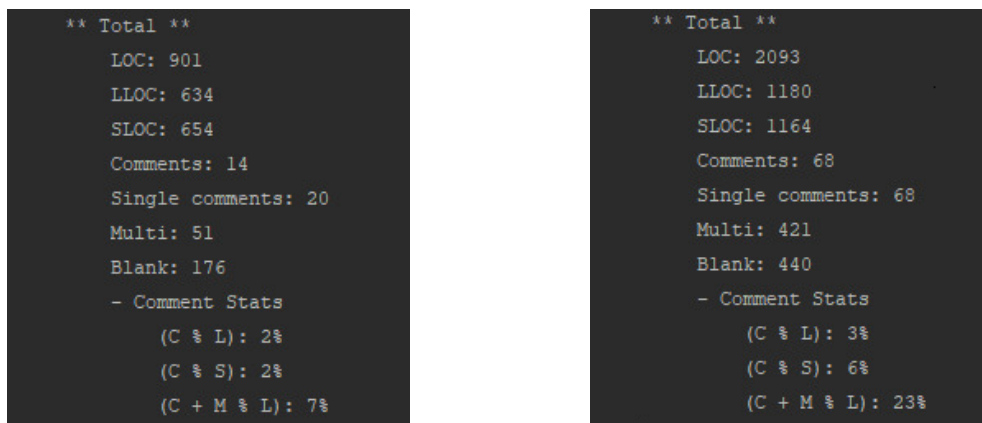


Figure 6.7: On the left the old percentage of code comments is shown and on the right the new percentage is shown. The total comment percentage is at the bottom. It is determined by dividing the total number of lines of code by the amount of comments.

6.3.3 Minimising external dependencies

The second metric that had to be improved, was minimising the external dependencies of PlebNet. The provided version PlebNet had many calls to different external packages from many different classes. This is a problem when one of these external packages is updated. If that were to happen, PlebNet would have to update all classes that make calls to this external package and this would take valuable time. To fix this, controller classes were created and only the controller classes can make calls to external dependencies. Each class in PlebNet that has to interact with the external dependencies will go through the controller classes. This is an improvement for maintainability as with the new PlebNet, whenever one of the external dependencies were to make changes only the corresponding controller would need to be updated, which costs less time than updating almost every class in PlebNet.

In chapter 4.1 the new architecture, which includes the new controller classes, is explained. In the old version of PlebNet 21 calls to external dependencies were made from 4 of the 6 classes (PlebNet had two extra old classes but these were not used). In the new version this number of calls to external dependencies is reduced to 17 calls from 4 of the 16 classes.

6.4 SIG feedback

The final part of quality assurance is making improvements based on *SIG* feedback. *SIG* stands for Software Improvement Group and in week 5 and 8 the PlebNet project was submitted to them. They analysed the code and provided feedback which was used to make improvements. Their full feedback report can be read in Appendix C and D.

SIG rated our project on a scale from one to five stars and gave it an overall rating of 4.1 in the first feedback session. This is considered by *SIG* to be above average and therefore they did not have any concrete recommendations for the code. They did warn that the level of quality should be maintained and that the number of tests for PlebNet could be further improved during the remainder of the project. This feedback was implemented by increasing the amount of tests. When PlebNet was submitted to *SIG* the first time there was 76% line coverage with PlebNet tests, at the end of the project this percentage was increased to 85%.

CHAPTER 7

Conclusions

This chapter provides a summary of the project. It looks back on the project and concludes what has been done. It also discusses the ethical considerations of the project and what can be done to improve PlebNet in the future.

7.1 Conclusions

The Must haves are all implemented. As all *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 coins and with actual Bitcoins. This results in a usable version of PlebNet which can keep it self alive by earning 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 is worked 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 PlebNet can be successfully build and that all tests succeed.

The should haves are all implemented. The first two *should haves* were regarding the project work flow: improving documentation and maintainability. This should allow the next generation of programmers 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 live monitoring and automatic GitHub issue creation features are added to 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 allows for the possibility to test certain behaviours, 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 of the agent are to be considered carefully. This will often mean that there should be some form of communication between agents, 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 for the implementation of multiple strategies, but they 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 were mostly left alone. During the project, however, it was decided that a new VPS host had to be implemented in order for the system to be more reliable, as most of the originally implemented providers became incompatible. This resulted in the addition of a new host to Cloudomate.

All in all, it can be concluded that minimal requirements are met and many other, lower priority, requirements are implemented as well. The working version of PlebNet can be used for further research in the field of self-sustaining botnets. With the implementation of PlebNet a step is made to a more anonymous use of the internet, as the Tribler network can handle more traffic.

7.2 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, Tribler and Transmission and streaming clients such as Popcorn-time.

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 [36]. Following the trends of these laws, it is not hard to imagine a world in which information sharing is strictly monitored, possibly 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 [37], 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 to support 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. For this reason, we believe that this project in conjunction with Tribler is ethical.

7.3 Reflection

Between the start and the end of the project, a learning curve was noted. For most of the team members it was a new experience to develop in Linux and fully use this operating system. Working with Linux demanded new skills to be developed, such as the use of the terminal, shell scripts and package managers. 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 somewhat chaotic start. The goal was clear, but there was much to achieve before PlebNet would be remotely functional. This resulted in quite some struggles during the start of the project, but after creating UMLs 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 organisation 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, it can be concluded that a lot was learned over the course of the project. While it is a challenging project, the knowledge gathered and skills learned from the previous courses in the Bachelor programme allowed us to quickly adapt to new challenges. The contributions between all team members were satisfactory as each member delivered their work according to the planning. The good communication, both online and in-person, allowed for most of the project to go as planned.

7.4 Recommendations

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. 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 both the online network and the data flowing through it.

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 available MBs in order to be able to purchase a new VPS after a single deal. Multiple strategies, some even including spending Bitcoin on MBs whenever the price is right, can be tested and implemented to increase the life expectancy of the network. The Bitcoin 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 genetic algorithms 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:

- PlebNet updating it's own dependencies to a new version or a known-to-be-stable version.
- using different trading tactics based on DNA.
- not always picking the cheapest option from a provider, but considering the differences in resources.

The agents should also be better armed against being banned from certain provider. The purchasing behaviour could be more human-like by automatically creating a new email address with a trusted extension (@gmail for example) and having a more realistic identity generator. This might make it harder to detect automated purchases.

Bibliography

- [1] Who Has Your Back? Government Data Requests 2017. <https://www.eff.org/who-has-your-back-2017#govt-requests>. Accessed: June 20, 2018.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] What Is Fair Use? <https://fairuse.stanford.edu/overview/fair-use/what-is-fair-use>. Accessed: June 11, 2018.
- [6] Youtube. <https://www.youtube.com/?gl=NL&hl=nl>. Accessed June 21, 2018.
- [7] How Much Data is Created on the Internet Each Day? <https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/>. Accessed: June 20, 2018.
- [8] More and more experts warn of the dangers of Article 13 upload filters. <https://www.communia-association.org/2018/06/16/experts-warn-dangers-article-13-upload-filters/>. Accessed: June 20, 2018.
- [9] BitTorrent. <https://www.bittorrent.com>. Accessed June 11, 2018.
- [10] Tor: overview. <https://www.torproject.org/about/overview.html.en>. Accessed: June 20, 2018.
- [11] Types of IAT - Advanced Open Source Multi-Hop. Accessed: June 20, 2018.
- [12] Tor browser. <https://www.torproject.org/projects/torbrowser.html>. Accessed: June 20, 2018.
- [13] Tribler - Privacy using our Tor-inspired onion routing. <https://www.tribler.org/>. Accessed: April 24, 2018.
- [14] The BitTorrent Protocol Specification. http://www.bittorrent.org/beps/bep_0003.html. Accessed: June 20, 2018.
- [15] N.C. Bakker, R. van de Berg, and S.A. Boodt. Autonomous Self-replicating Code. June 2016.
- [16] J. Heijligers, R. van den Berg, and M. Hoppenbrouwer. Plebnet: Botnet for good. June 2017.
- [17] M. van Beusekom, T. Catshoek, A. Kroon, J. Rijdsdijk, and J. Wooning. Cs4160 Blockchain Engineering Group 7 - Blockchain life. June 2017.
- [18] Electrum Bitcoin Wallet. <https://electrum.org/#home>. Accessed: April 27, 2018.

- [19] AzireVPN - Secure and Privacy-minded VPN service. <https://www.azirevpn.com/>. Accessed: June 5, 2018.
- [20] Veilig online betalen of betaald worden - PayPal Nederland. <https://www.paypal.com/nl/home>. Accessed: June 21, 2018.
- [21] BitPay - Get Started with Bitcoin Payments. <https://bitpay.com/>. Accessed: April 30, 2018.
- [22] Buy/Sell Digital Currency - Coinbase. <https://www.coinbase.com/>. Accessed: April 30, 2018.
- [23] Accept Bitcoin & Other Cryptocurrencies. <https://www.coinpayments.net/>. Accessed: April 30, 2018.
- [24] Cryptocurrency Exchange & Blockchain Wallet - Spectrocoin. <https://spectrocoin.com/>. Accessed: April 30, 2018.
- [25] Martin Fowler. Continuous Integration. <https://www.martinfowler.com/articles/continuousIntegration.html>. Accessed: June 22, 2018.
- [26] Jenkins. https://jenkins.tribler.org/job/GH_PlebNet/. Accessed: May 8, 2018.
- [27] Snapcraft. <https://docs.snapcraft.io/snaps/>. Accessed May 17, 2018.
- [28] Docker. <https://www.docker.com/>. Accessed June 20, 2018.
- [29] Gmail. <https://www.google.com/gmail/>. Accessed: June 22, 2018.
- [30] Twitter. <https://twitter.com/?lang=nl>. Accessed: June 15, 2018.
- [31] mIRC: Internet Relay Chat client. <https://www.mirc.com/>. Accessed: May 14, 2018.
- [32] OpenVPN - Open Source VPN. <https://openvpn.net/>. Accessed: June 12, 2018.
- [33] TUN/TAP. <https://www.kernel.org/doc/Documentation/networking/tuntap.txt>. Accessed May 20, 2018.
- [34] Let's Encrypt. <https://letsencrypt.org/>. Accessed June 20, 2018.
- [35] Think Twice Before Using the "Maintainability Index". <https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/>. Accessed June 06, 2018.
- [36] P Samuelson. Unbundling Fair Uses. 2009. http://fordhamlawreview.org/wp-content/uploads/assets/pdfs/Vol_77/Samuelson2_Vol_77_Apr.pdf Fordham Law Review. 77. Retrieved June 18, 2018.
- [37] The Piratebay remains. <https://torrentfreak.com/the-pirate-bay-remains-on-top-11-years-after-the-ra> Accessed June 01, 2018.

APPENDIX A

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. Your 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

APPENDIX B

Infosheet

Title of the project: Creating a Botnet of Self-Replicating Autonomous Organisms

Name of the client organization: Delft Blockchain Lab

Date of the final presentation: July 04, 2018

Description: The problem faced in this project was the implementation of a self-sustaining botnet, which could earn Bitcoin to buy new instances. The goal of this network is to create Tribler exit-nodes which allow for anonymous access to the internet. This botnet has to make multiple decisions regarding which server to acquire. This is achieved using genetic algorithms. The Tribler organisation provided us with the code from previous groups. This code was used as a initial guide, but not much of it is left in the delivered code, as it was unstructured and not functional. The delivered code will be used for further development and research regarding the autonomous self-replication code. It is currently ready for use, as it is functional, but there are also many features which can be added or researched for further work.

Teammembers: The team consisted of four members:

Member 1	Name	T.S. (Thijmen) Jaspers Focks BSc
	Interests	Stock market, Blockchain, Concrete Bridges
	Main contributions	Communication, Main architecture, Graphical design
Member 2	Name	W. (Wing) Nguyen
	Interests	Machine Learning, Big Data, Linux
	Main contributions	Cloudomate, VPN, ProxMox/End-to-end testing
Member 3	Name	H. (Harvey) van Velton
	Interests	Blockchain, P2P networks, Artificial intelligence
	Main contributions	Connection to Tribler, Cloudomate, Unit testing
Member 4	Name	V. (Viktor) Wigmore
	Interests	Artificial Intelligence, Big Data, Blockchain
	Main contributions	Connection to Cloudomate, PlebNet general

All team members contributed to the testing of the software, preparing the report and the final project presentation.

Client: ir. M.A (Martijn) de Vos, Employee of Delft Blockchain Lab

TU Coach: Dr. ir. J.A (Johan) Pouwelse, TU Delft, Distributed Systems

Contact Person: T.S. (Thijmen) Jaspers Focks BSc., thijmensjf@gmail.com

The final report for this project can be found at: <http://repository.tudelft.nl>

APPENDIX C

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 D

SIG feedback week 8

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

[Hermeting]

In de tweede upload zien we dat het project een stuk groter is geworden. De score voor onderhoudbaarheid is in vergelijking met de eerste upload ongeveer gelijk gebleven.

In de feedback op de eerste upload werden twee mogelijke verbeterpunten genoemd: Unit Size en Unit Interfacing. Bij Unit Interfacing zien we een kleine verbetering, zowel wat betreft het refactoren van de bestaande code als de kwaliteit van de nieuwe code. Bij Unit Size zien we echter dat jullie in de tweede upload wat minder strikt zijn dan eerst. Dat is normaal als je project groeit, maar probeer hier in de toekomst op te letten. Voorbeelden van methodes die in de nieuwe code langer zijn geworden zijn `create_gist()` in `git_issuer.py` en `create_wallet()` in `wallet_controller.py`.

Naast de toename in de hoeveelheid productiecode is het goed om te zien dat jullie ook nieuwe test-code hebben toegevoegd. De hoeveelheid tests ziet er dan ook nog steeds goed uit.

Uit deze observaties kunnen we concluderen dat de aanbevelingen uit de feedback op de eerste upload grotendeels zijn meegenomen tijdens het ontwikkeltraject.

