Autonomous self-replicating botnet

Harvey van Veltom (4350073)
Thijmen Jaspers Focks (4158393)
Viktor Wigmore (4279638)
Wing Nguyen (4287118)

June 4, 2018

Abstract

Your abstract goes here...

Contents

CHAPTER 1

Introduction

In recent years, privacy on the Internet is becoming a more and more concerning issue. Governmental agencies are more frequently asking ISPs for data about their customers. And even without permission they are still allowed to gather data due to the introduction of several new laws. Because they are able to look at our data, we are vulnerable to the potential abuse of it.
In P2P networks, computers are connected to each other without a central node. This way there is no point through which all data flows. Whenever a node in such a network goes offline, the other nodes can still provide the requested data as they have it too. Because of this, P2P networks are mostly used for file sharing. However, with current P2P networks privacy is nonexistent as it is clear from who to whom data is flowing.
Tribler is an open source P2P file sharing program developed as a research project at Delft University of Technology. It keeps improving on the old BitTorrent protocol, adding new features while operating in a distributed manner, meaning there is no centralised component in Tribler. Tribler introduces proxy layers to add privacy for both the downloader and the uploader. With a single proxy layer there is still a risk of the proxy being corrupt and listening to the data sent. For this reason, Tribler has three proxy layers between the user and the rest of the P2P network.
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.
In order to create a robust network of exit nodes, this project will focus on software which will result in a network of autonomous self maintaining exit nodes. Buying VPSs and using them as an exit node would create such a network, but this requires money as these VPSs have to be bought. The system to be implemented earns Tribler Market Coins by functioning as an exit node, these coins have a theoretical value as they give the owner certain benefits within Tribler. The obtained coins are then to be sold for Bitcoin with which VPSs can be bought.

1.1 Previous work

The idea of a network of autonomous self maintaining exit nodes started with the development of TENNET. After this, a second BEP group started over from scratch, using the experience of the first group, and developed two modules called PlebNet and Cloudomate. Cloudomate was solely responsible for buying servers and PlebNet was responsible for installing Tribler, setting itself up as an exit node, selling Tribler Market Coins for Bitcoins, activating Cloudomate and ultimately installing itself again on the newly acquired server.
Cloudomate was implemented to select a VPS provider from a selection of providers who accept Bitcoin, register an account at that provider, and buy a server with the given account. After the second BEP group finished working on Cloudomate, master students continued working on it without considering that it was supposed to be a module for PlebNet. This caused PlebNet to not work anymore with Cloudomate. Some methods were either removed, replaced or the parameters it needed had changed. The master group added new VPS providers to the list of available providers and the possibility of buying VPN protection.

After looking at both Cloudomate and PlebNet and some experiments as will be discussed in the next chapter, we decided that we should mainly work on PlebNet as its core functions were working independently but not together and probably never worked together. With Cloudomate there were some problems as well but these were the cause of changes to the providers themselves as will be discussed later on.

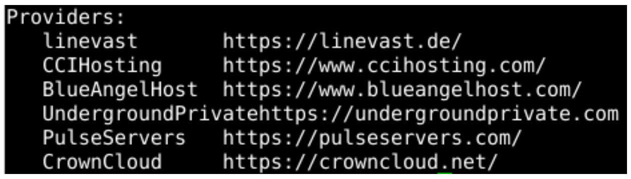
CHAPTER 2

Initial experiments

The initial steps were to assess the functionality of the works delivered by the previous groups as well as the bitcoin wallet library Electrum and Tribler.

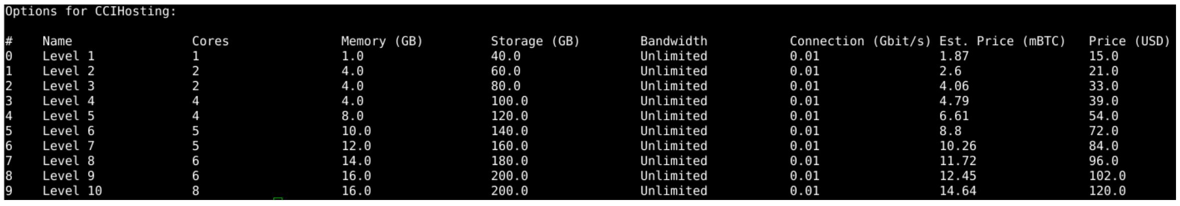
2.1 Cloudomate

Cloudomate works by crawling the webpages of VPS and VPN providers for options and filling out the HTML forms on the purchase page. The main concern with this approach is that providers may change their page layouts, resulting in non-functionality of Cloudomate and therefore PlebNet. However, unless providers are willing to create APIs for this kind of task (thus allowing possible harmful bots to purchase their services), this approach is the only way of achieving automated VPS/VPN purchases at the moment.
The user information can either be from a real identity or from a randomly generated identity. The information is stored in a configuration file which can be read from when the server information is queried by the user (or agent).
VPS providers offered by Cloudomate are shown in **Fig. 2.1**. The options that are purchasable from these providers can be queried and shown in **Fig. 2.2**. The webpage of provider UndergroundPrivate had changed in layout and was unusable.



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

Figure 2.1: VPS providers



```
Options for CCIHosting:

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

Figure 2.2: CCIHosting's options

Gateways

The different gateways used by each provider were: Bitpay this gateway is used by Linevast, BlueAngelHost and CrownCloud. CCIHosting used to work with the Coinbase gateway however they changed this to the Coinpayments gateway. UndergroundPrivate used to work with the Blockchainv2 gateway however they changed this to the Spectrocoin gateway. And while PulseServer used to work with Coinbase now they don't use any form of bitcoin gateway anymore instead they now only accept paypal payments.

bitpay issues, fix (wing) Initially, some errors were encountered when purchasing from providers that used Bitpay. The Bitpay invoice page was updated to include a choice between Bitcoins and Bitcoin-cash **add pic** , eventhough the Bitcoin addresses were accessible through Bitpay's API, the user has to click on either Bitcoin or Bitcoin-cash for the transaction to succeed. This was solved by sending a GET/ request to the correct Bitpay invoice URL, mimicking the user's choice.
Bitpay also enforced the payment protocol BIP-70 [1], which requires the buyer to use a compatible wallet (such as Electrum) to fulfil the payment request. A payment can be made by the user by either scanning the QR code or copying the payment URI and pasting it into their wallet **addpic** .
The URI has the form *bitcoin:?r=(merchant invoice url)*.
In order for Cloudomate to function, the Bitpay gateway class was modified to decode and fetch the Bitcoin address for payment. This was done by mirroring the method calls of Electrum's internal libraries. **add pic**

Electrum

Electrum is the Bitcoin wallet that was used in Cloudomate to purchase VPS/VPNs. The version that was used was 2.8.3, which was a Python 2.7 version. Because Tribler, PlebNet and Cloudomate were written in Python 2.7, upgrading Electrum was not possible. Electrum 2.8.3 was fortunately fully functional and is being used in this iteration of PlebNet as well.

2.2 Tribler

The ultimate goal of the agents is to generate money by running as an exit node for Tribler. If an agent does not have access to Tribler it won't be able to run as an exit node and earn tokens. And with no tokens the agent is unable to generate any money and therefore unable to buy new servers for new agents. So for an agent to be successful in generating money it needs to be able to install and run Tribler from source. And on top of that it needs to run Tribler as an exit node.

The installation and running of Tribler is done as follows. First the latest build .deb file is pulled from the latest stable Jenkins build. the .deb filed is used instead of the tar.gz file because the tar.gz file had issues with installing the submodules necessary for Tribler to run correctly and the .deb file did not have that issue. After the .deb file is downloaded the necessary packages are installed and after the .deb file is installed using dpkg. After this is done Tribler can simply be started by running run_tribler.py

The second part is ensuring that the agent can run Tribler as an exit node. This is done by going to the config map of the installed Tribler file. Then starting the python compiler from command line and import the class tribler_config. This class is responsible for getting and setting the values of the config file for Tribler. After importing the class an instance of that class is created. With that instance the value for tunnel_community_exitnode_enabled is set to True and this change is than written to the config file. Changing this value ensures that whenever Plebnet is started it is automatically set to run as exit node ensuring that the agent is able to generate tokens.

2.3 Plebnet

In the previous sections the dependencies of PlebNet have been made operational again. With these applications (Tribler, Cloudomate, Electrum) working again, it should be possible to get PlebNet running. The steps taken to achieve this are discussed in this section.

2.3.1 main purpose of PlebNet

PlebNet self is responsible for the deployment of as many Tribler exit-nodes as possible on the long-term with a stable expaning network of independent agents. This should be achieved by earning credit for the Tribler market and sell this for Bitcoin. This credit is earned by being operational as an Tribler exitnode and provide allow users to upload/download anonymously.

The next step is to acquire new VPS via Cloudomate and install a new agent. If the deployment rate of new agents is larger each iteration (one month per VPS), this will result in a stable network. An overview of this behaviour is provided in **Fig. 2.3**.

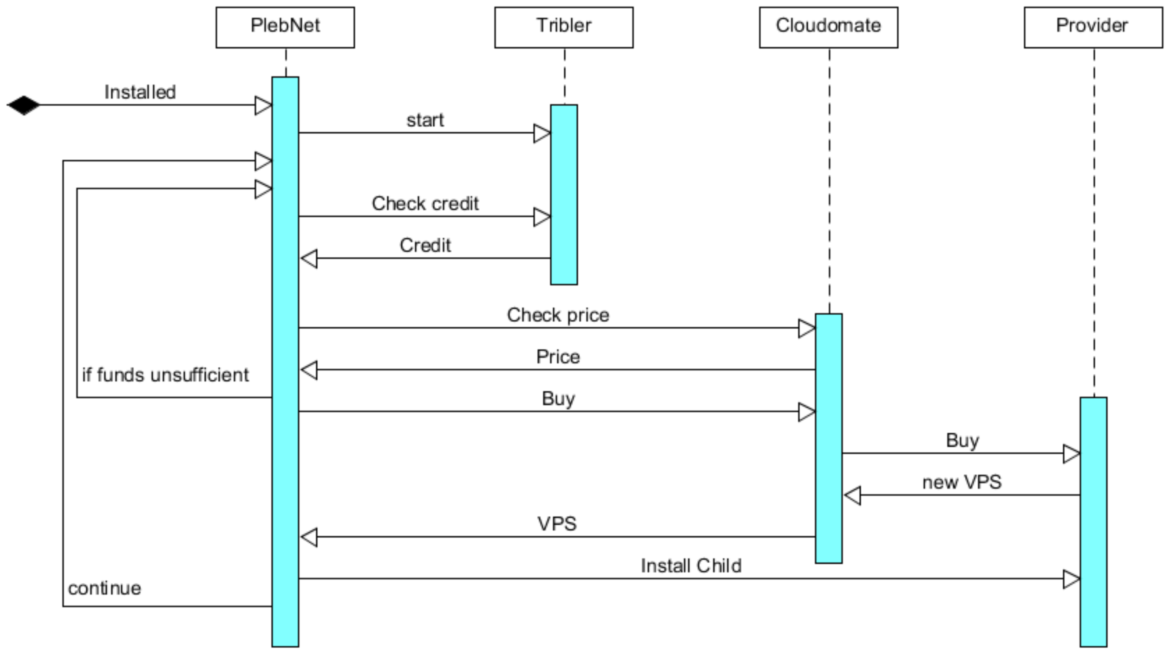


Figure 2.3: Communication diagram of PlebNet

2.3.2 original state

The received code was not and could not have been functional. An example was the way the configuration was handled. It contained a single `config` file, which gave errors as a new server was obtained. The emailaddress could not be reused. The code also contained wrong method calls and depricated packages.

2.3.3 maintainability

The level of documentation and commenting on the code was low, which made it harder to grasp the purpose of the different files and understand the relation between different packages. The system architecture was not structured and files were too large to understand (450+ loc) and methods to large to be effective (100+ loc).

This resulted in a relatively low quality of code and requires major refactoring before new feature can be added.

CHAPTER 3

System requirements

Together with the client of the project, M. de Vos, a list of requirements for the new version of PlebNet was derived. This is done using the MOSCOW method, which groups all requirements based on their priority:

- Must haves, these are the functionalities which should be implemented in order for the project to be successful;
- Should haves: these functionalities are wanted by the client, but the software is usable without their implementation;
- Could haves: these functionalities will only be implemented when there is time available at the end of the project;
- Won't haves: these functionalities are out of the scope of the project, but might be nice to implement later on.

Firstly a short summary of all functionalities per group is given. These are elaborated on further in the remainder of this chapter.

3.1 MOSCOW elements

3.1.1 Must haves

- Fully operational end-to-end system, including Cloudomate, Tribler and PlebNet
- The ability to autonomously install and execute PlebNet (Cloudomate, Electrum and Tribler) on a new server
- The ability to acquire new Bitcoin funds bought via the marketplace for Tribler tokens
- The ability to monitor the status of the online agents
- Passing builds on Jenkins

3.1.2 Should haves

- Proper documentation
- Highly maintainable code (comments + structure)
- The ability to dynamically add new VPS provider to the existing DNA list if implemented in Cloudomate
- The possibility to install VPN protection

3.1.3 Could haves

- Transfer obsolete funds
- The ability to create an issue on GitHub (if an error occurs)
- Genetic Algorithm deciding which VPS to buy
- Simulate VPS provider to allow for free end-to-end testing
- Ability to monitor the status of the online agents live
- Package PlebNet with its dependencies for easier installations

3.1.4 Won't haves

- New VPS and new VPN added to the list of available VPS and VPN
- The ability to extend the lease of the current VPS
- Improve Cloudomate for standalone use

3.2 Must haves

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

Autonomous installation: When a new server is bought, PlebNet is responsible for installing Tribler, Electrum, PlebNet and Cloudomate as well as installing all required dependencies on the new server.

Status monitoring: In order to know how well the system is functioning, the status of the servers must be accessible. This means that the servers should sent their status to a central receiver which collects and displays them.

Passing build on Jenkins: In order to ensure the quality of the project, Jenkins is used as a continuous integration tool.

3.3 Should haves

Proper documentation: The current version of the project has no good documentation and there is no clear overview of how the project is structured. In order improve the maintainability of the project, a so called UML diagram should be made explaining the basic function and interaction of classes with each other.

Highly maintainable code: The current version of the project has little to no comments in the code explaining how functions work and what they should do. In order to improve the maintainability of the code, comments for new classes and functions should be written.

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

Installing VPN protection: Because most VPS providers do not accept their services being used as exit nodes in P2P networks. PlebNet should be able to install VPN protection in order to prevent the agent from being banned by the VPS provider.

3.4 Could haves

Transfer obsolete funds: At the end of the life cycle of an agent, the agent will have money left which was not enough to buy a new server with. To prevent losing this money, the agent could be able to transfer this money to a central node or to one of the child nodes (inheriting).

The ability to create an issue on GitHub In order to be able to improve the agents on 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 neat way to handle these notifications is by implementing an automated issue creator. When an error occurs, this posts an issue on GitHub, including all important information such as provider, trace call-back and other settings.

Genetic Algorithm for reproduction The bot could be programmed to dynamically chose a survival strategy whenever it creates a new child. Possibilities are the decision which VPS to acquire, or the trading strategy on the Tribler market.

Simulate VPS provider to allow for free end-to-end testing Testing in a fully controlled environment allows the developers to simulate different settings and prove the concept of PlebNet. The configurations can be quickly restored and the system improved.

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

Package PlebNet with its dependencies for easier installations Delivering a preinstalled package would result in relatively simple and clean installation of PlebNet and all dependencies.

3.5 Won't haves

Adding new VPS/VPN services This project focuses on implementing the end-to-end usability of PlebNet and improving or adding new functionalities. The addition of new services is done rather easily afterwards and can be done by new groups.

The ability to extend the lease of the current VPS As extending a lease on a server is a complete different operation than the installation of a new one, this would require extra work to implement. As the result is the same, it is not useful to implement such a feature.

Improve Cloudomate for standalone use The functionality of Cloudomate can be improved to be able to be usefull outside PlebNet as well. This, however, is outside the scope of this project, but could be done later on by another projectgroup.

CHAPTER 4

Research

In this chapter **to be expanded later on**

4.1 Monitoring

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

4.1.1 installation

As a new server is acquired, the host sends an email with the specifications to an email address. This email address can be chosen to be accessible by the person who installed the initial bot and provides some information regarding the clone speed of the online servers and the amount of online agents.

However, this does notify anyone as the server is banned or runs into other trouble such as errors in the the code. This can be solved by implementing a better notification strategy.

4.1.2 IRC

An example of a solid communication network is Internet Relay Chat, or IRC. This protocol is in use since 1988 and still has hunderds of thousends of users. IT has porven to be reliable and stable.

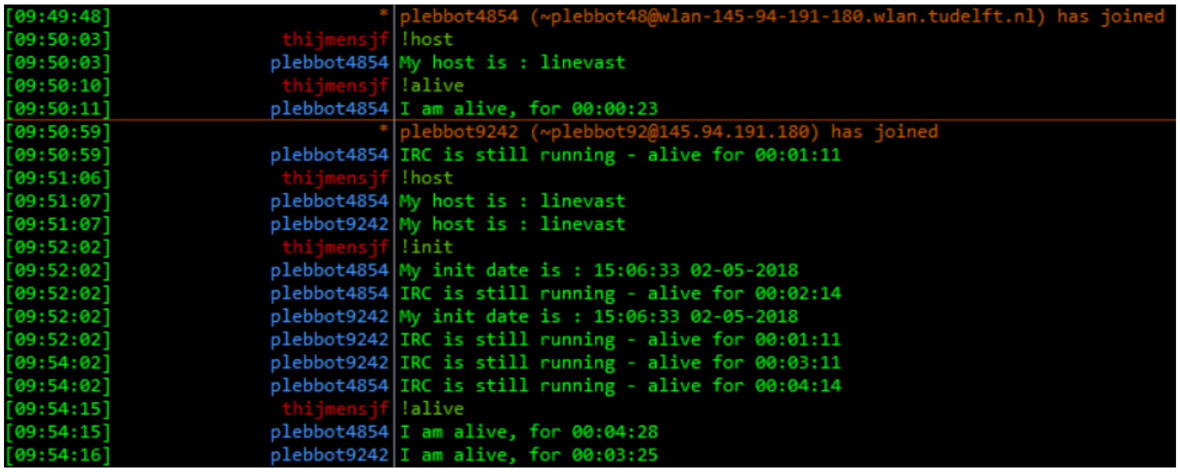
IRC works by setting up a connection to a server node and once the bot is only, it can join a certain channel. As long as the IRC client is online and responsive to PING messages, it is notified about all messages in its channels. This means that all agents can join a chosen channel and provide information in heartbeats. This way it is possible to keep track of the alive agents. As a new agent is installed in can send information regarding its DNA and configuration. By sending these notifications to a public chat, it is possible for any to keep track of the events in the botnet. This also prevent single points of failure in the network. Contrary to sending messages to a single access account, which could be lost.

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

- `!alive` asks for a heartbeat
- `!host` asks for the host information

- `!init` asks for information regarding the initialization of 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, nut 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.



```
[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
```

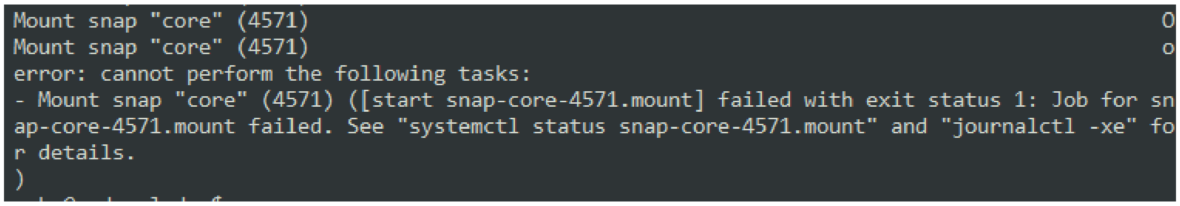
Figure 4.1: IRC communication

4.1.3 Git issues

to be added later on

4.2 Snaps

We've researched the possibility of packaging code as Snaps[2], allowing for a more stable way of installing PlebNet in a containerized fashion. Unfortunately, Snaps works by mounting a virtual filesystem which won't work on most VPS due to restricted access.



```
Mount snap "core" (4571)                                                  O
Mount snap "core" (4571)                                                  o
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 4.2: The virtual filesystem could not be mounted on a VPS

CHAPTER 5

System architecture

When reviewing the code written for PlebNet by the first group, we saw that there were some classes which had too many responsibilities as well as some classes which were not used at all by PlebNet. Because of these reasons, we decided to restructure PlebNet in such a way that every package within PlebNet would have its own responsibility. The structure of the refactored PlebNet can be seen in

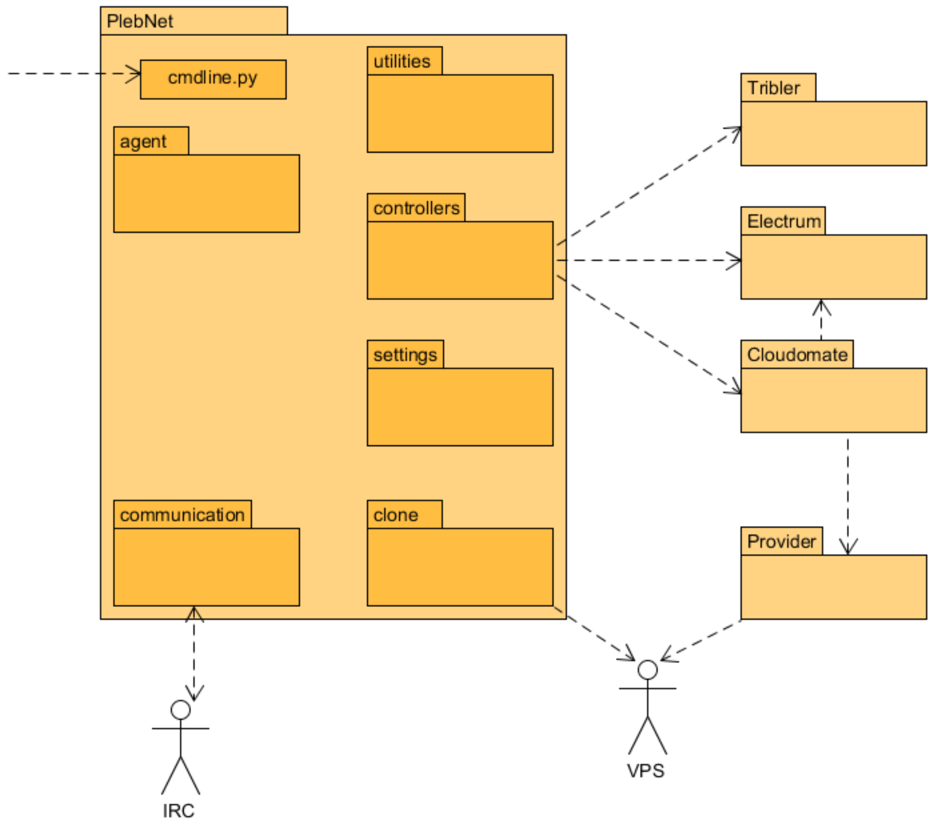


Figure 5.1: Package diagram

the package diagram in **Fig. 5.1**. The responsibility of every package is as follows:

The controller package contains the only code which makes calls to Tribler, Electrum and Cloudomate. This way, whenever something changes in these dependencies, the only code which needs to be updated is located inside the controller package.

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

The clone package is responsible for installing PlebNet on the newly bought servers.

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 send a twitter message upon spawning a child.

The settings package contains the server data per agent such as server used and what the expiration date is as well as the configuration of the IRC client.

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

A more detailed diagram of how the various classes within PlebNet work together can be found in appendix A

CHAPTER 6

System testing

After refactoring PlebNet, both PlebNet and Cloudomate had to be tested. As our main focus was on PlebNet, we decided to fix the existing failing tests of Cloudomate and not write any new tests for Cloudomate. PlebNet was not tested at all, so we decided to test functionalities of PlebNet independent of each other as well as end-to-end test the whole system. Keeping track of the project was done using Jenkins as a continuous integration tool. [3]

6.1 Unit testing

6.1.1 Testing Cloudomate

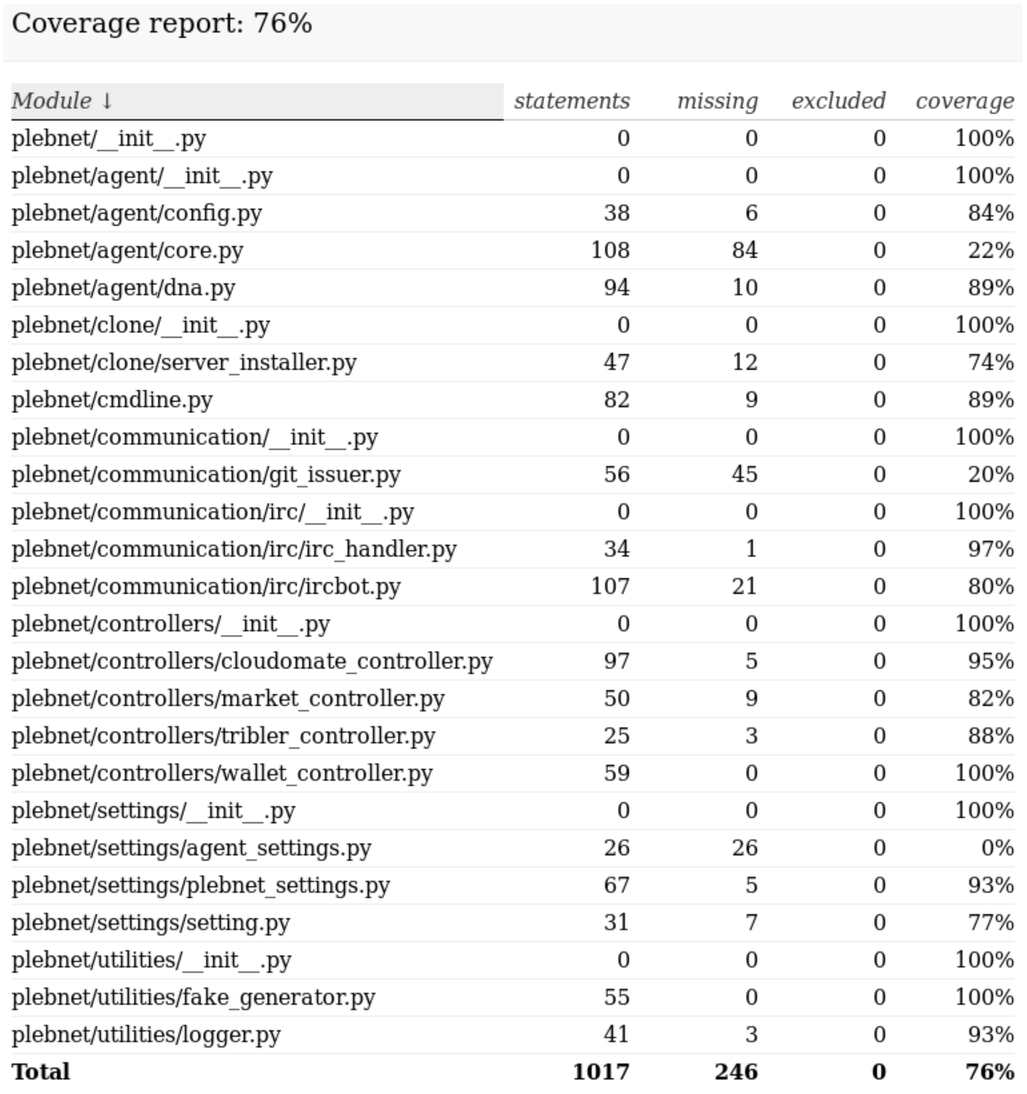
Certain tests that used to work on Cloudomate failed at the start of this project. So one of the early priorities was to fix these tests and make sure that the Cloudomate Jenkins build would succeed. The tests that have been successfully fixed are as follows: fixed the way that bitcoin urls are split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

6.1.2 Testing PlebNet

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

Jenkins

As stated earlier, Jenkins is the framework used for continuous integration. This includes running all tests with each pull request as well as creating a coverage report with each build. Jenkins is chosen, as the Tribler organisation uses it already and the previous group used it for Cloudomate as well. This way all projects are maintained by a single service. The total coverage we have achieved for PlebNet can be seen in **Fig. 6.1**



Coverage report: 76%

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

Figure 6.1: Coverage report on Jenkins

6.2 End-to-end testing

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

6.2.1 Proxmox

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

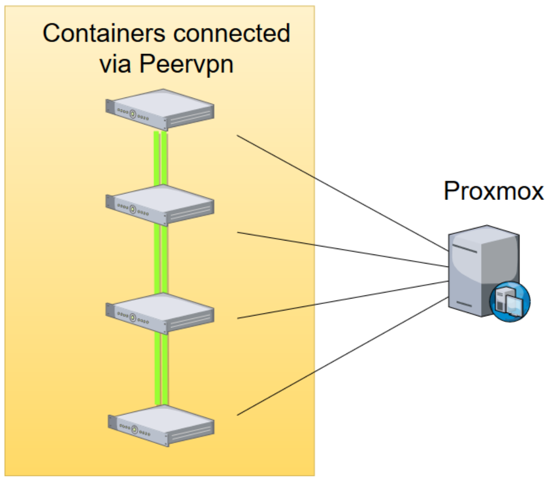


Figure 6.2: Overview of the Proxmox server. Containers are connected with each other through PeerVPN.

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 Bitpay provides, the BitPay library for Python2.7 was used. Each machine hosting ProxHost needs to first have a token verified from the Bitpay control panel.

6.2.3 Web API

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

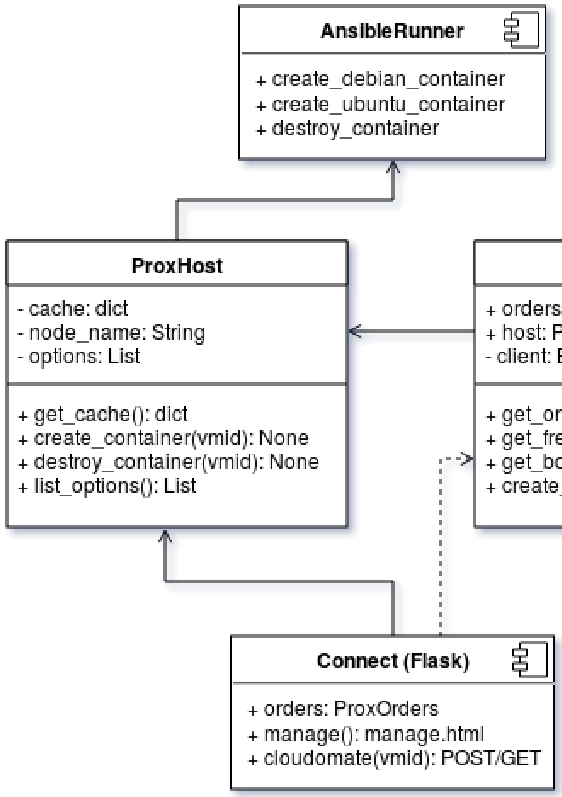
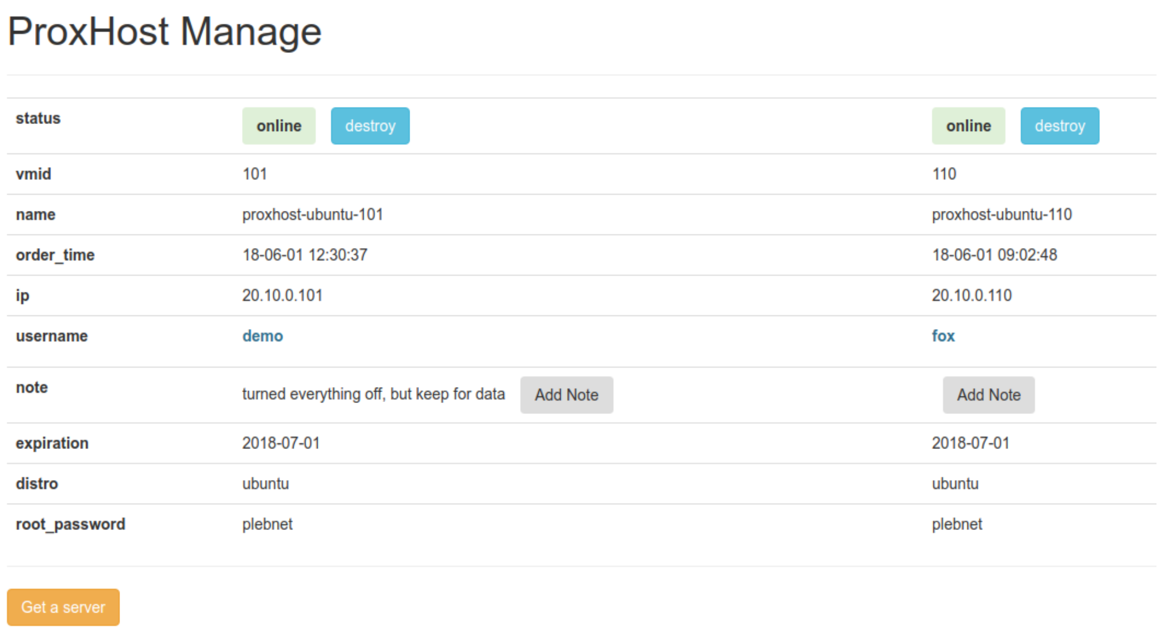


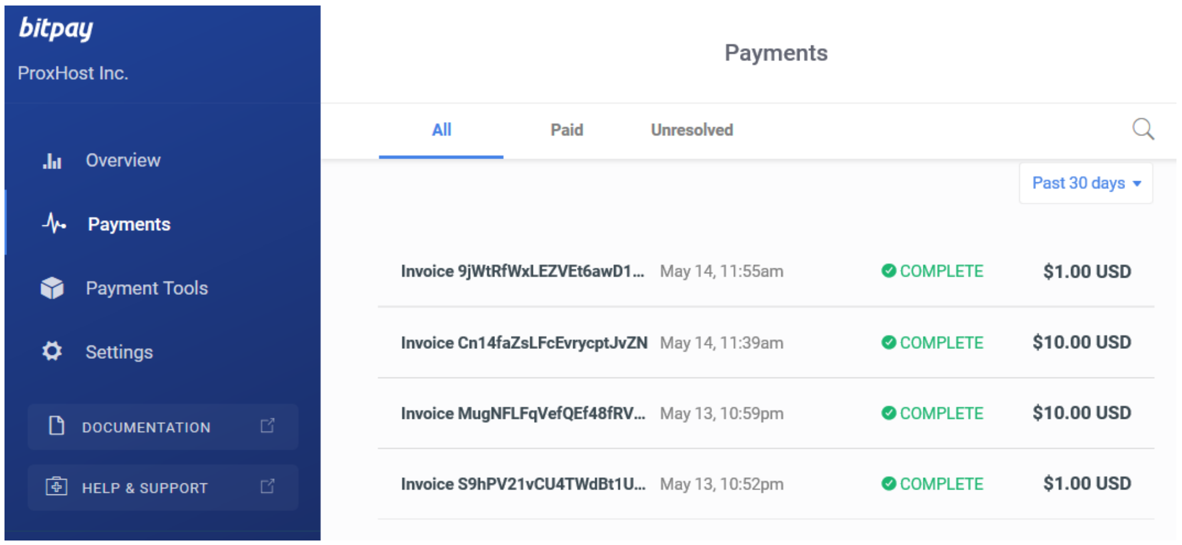
Figure 6.3: A simplified UML diagram of the end-to-end system. The Flask component ”Connect” allows Cloudomate or web users to purchase containers. When a purchase is made, ProxOrders creates an invoice which Connect returns as a HTTP response. ProxHost can either create or destroy containers by creating an AnsibleRunner object with the appropriate parameters.



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

Figure 6.4: The minimal web API provides an easy way to destroy and create containers.



Invoice 9jWtRfWxLEZVEt6awD1...	May 14, 11:55am	COMPLETE	$1.00 USD
Invoice Cn14faZsLFcEvrycptJvZN	May 14, 11:39am	COMPLETE	$10.00 USD
Invoice MugNFLFqVefQEf48fRV...	May 13, 10:59pm	COMPLETE	$10.00 USD
Invoice S9hPV21vCU4TWdBt1U...	May 13, 10:52pm	COMPLETE	$1.00 USD

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

CHAPTER 7

Marketplace

Plebnet uses the Tribler Marketplace API for two things. First Plebnet uses the Tribler API for its wallet creation. And secondly Plebnet generates its tokens by running as an exit node. These MB tokens can't be used to buy VPS servers themselves so instead Plebnet will sell these tokens for bitcoins and that is done via the Tribler marketplace. And after the agent has gotten bitcoins it uses the marketplace API to make transactions.

7.1 wallet creation

At the start of the project Tribler didn't use the Tribler marketplace API for its wallet creations. So to make this possible changes had to be made to both Plebnet and cloudomate. The respective changes are described in the below subsections.

7.1.1 Plebnet side

At first Plebnet imported Electrum directly. And it used this to create its Bitcoin wallets. This was changed so that Plebnet now uses the Tribler markeptlace API to create its wallets. This change was made so that Plebnet becomes more easily maintainable. Before the change Plebnet had to import both Tribler and Electrum separately and if either of these import made changes Plebnet's code had to updated as well. With the new method only Tribler has to be imported and the Plebnet code now only has to be updated if Tribler adjusts it code. Which makes Plebnet as a whole easier to maintain.

The Tribler marketplace API creates three types of wallet for Plebnet. The wallet that is used to store the MB tokens. This wallet is automatically created by Tribler and doesn't require interaction on Plebnets end. The second type is the Bitcoin wallet. This wallet is not created automatically by Tribler and requires a call from Plebnet for it to be created. This call is made to http://localhost:8085 which is where the marketplace is located. The call looks as follows:

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

In Plebnet this call is made with the requests library. After this call Tribler takes care of the rest and creates an Electrum bitcoin wallet that can be interacted with through the marketplace API. It is currently not possible too create an Electrum bitcoin wallet through the Tribler marketplace without a password so the password is pulled from the Plebnet setup config and is currently set to "plebnet" but this could be changed.

The third type of wallet is a new addition that wasn't used by Plebnet at the start of the project. This is the tesnet wallet. The wallet is created by a similar call to the marketplace as used for creating a Bitcoin wallet. The only difference between the calls is that for Bitcoin the call is made to http://localhost:8085/wallets/BTC and for testnet the call is made to http://localhost:8085/wallets/TBTC. Testnet is an alternative to the Bitcoin blockchain that is designed for the purpose of testing. the testnet coins themselves don't hold any value. Plebnet uses testnet wallets for the purpose of end-to-end testing on Proxmox servers.

7.1.2 Cloudomate side

The previous subsection explained how Plebnet creates the wallets using the Tribler marketplace API. Theses bitcoin and testnet wallets need to be passed on to cloudomate if cloudomate needs to buy a new VPS. To do this a new class is made in Plebnet called Triblerwallet. this class can either represent a Bitcoin or a testnet wallet. it contains two methods get balance and pay. When Plebnet creates a Bitcoin or a testnet wallet it also creates a Triblerwallet object for this wallet. This Triblerwallet object is passed on to cloudomate so that cloudomate can use it to interact with the created wallet and use it to buy new VPS's. Cloudomate does have another way to access Electrum wallets. Cloudomate has two classes Wallet and ElectrumWalletHandler. these classes look to see if an Electrum wallet has been created in the standard path 'usr/local/bin/electrum'. and if it finds it uses the electrum wallet found there to make its transactions. Cloudomate uses these classes if a user uses cloudomate standalone. If a user is interacting with cloudomate through Plebnet these classes are not used and instead the Triblerwallet class from Plebnet is used.

7.2 Marketplace interaction

Besides creating wallets the Tribler marketplace API is also used for the agent to sell its MB tokens which it generates by running as an exitnode for Tribler.

When Tribler is running the marketplace is located at localhost:8085. localhost:8085 also contains much more information besides just the market and the wallets but for the purposes of the agent this information is excluded. The commands necessary for the agent to successfully be able to interact with the marketplace to sell its tokens are as follows(Plebnet uses the requests library to be able to make the following requests):

```
curl -X GET http://localhost:8085/wallets/TYPE/balance
```

Herein TYPE can be: MB for MB token wallet, BTC for Bitcoin wallet or TBTC for testnet wallet. This GET request returns the balance of the addressed wallet.

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

This GET request pulls all the bids on the market. A bid being the maximum price a buyer is willing to pay for a commodity.

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

This GET request pulls all the asks on the market. An ask being the minimum price a seller is willing to recieve for a commodity.

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

This PUT request makes a new bid of how much the agent is willing to pay for a certain quantity. The command used to make a new ask is almost the same. the only difference is that the put request for asks have to be made to http://localhost:8085/market/asks.

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

This POST method is used to make a transaction from the specified wallet to a certain destination of a certain amount. This call is used in the Triblerwallet class. This class is created and passed on to cloudomate. So cloudomate uses this call if it wants to make a transaction to buy a new VPS. With these tools the agent is successfully able to interact with the Tribler marketplace so that it can sell its tokens for bitcoins and make transactions to buy new VPS's.

Bibliography

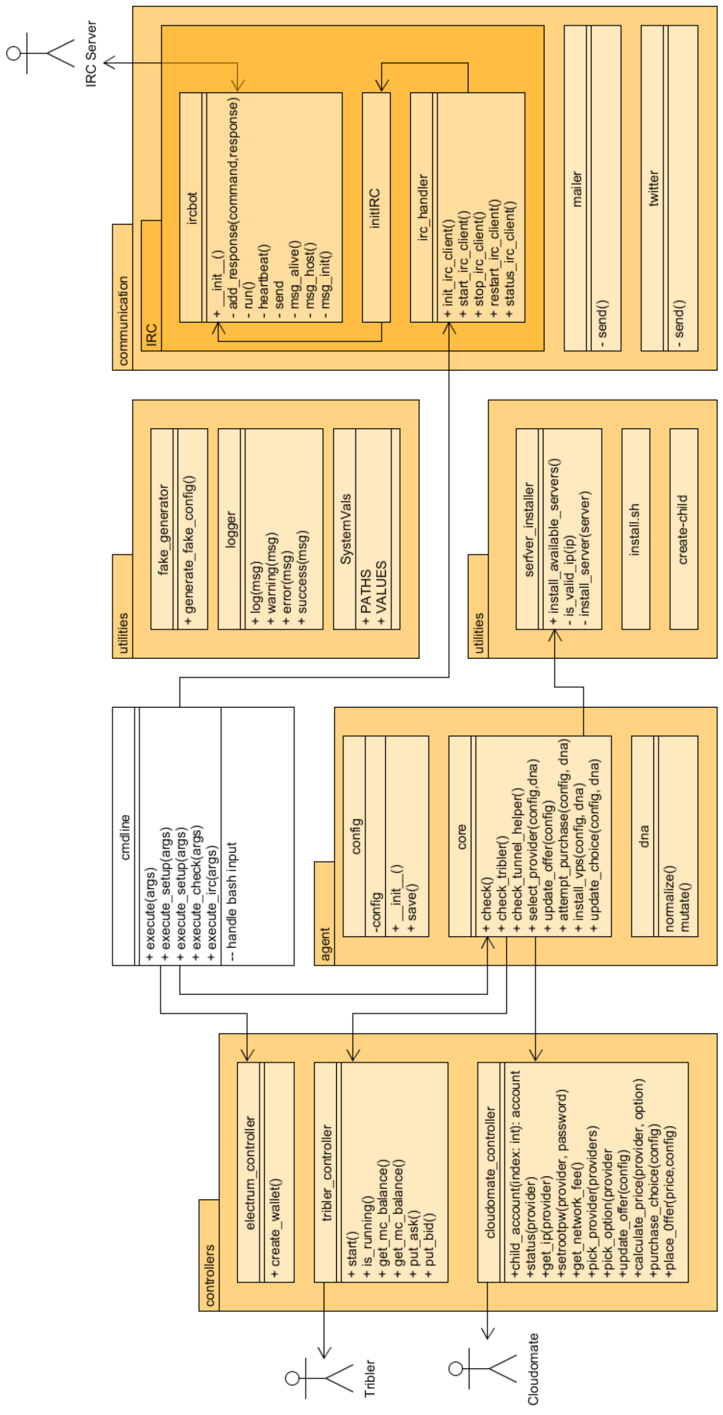
[1] Gavin Andresen. Bip: 72. 2013 (accessed May 16, 2018). https://github.com/bitcoin/bips/blob/master/bip-0072.mediawiki.

[2] Snapcraft. 2018 (accessed May 17, 2018). https://docs.snapcraft.io/snaps/.

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

APPENDIX A

Class Diagram


IRC Server
communication
IRC
ircbot
+ __init__()
- add_response(command,response)
- run()
- heartbeat()
- send
- msg_alive()
- msg_host()
- msg_init()
initIRC
irc_handler
+ init_irc_client()
+ start_irc_client()
+ stop_irc_client()
+ restart_irc_client()
+ status_irc_client()
mailer
- send()
twitter
- send()
utilities
fake_generator
+ generate_fake_config()
logger
+ log(msg)
+ warning(msg)
+ error(msg)
+ success(msg)
SystemVals
+ PATHS
+ VALUES
utilities
serfver_installer
+ install_available_servers()
- is_valid_ip(ip)
- install_server(server)
install.sh
create-child
cmdline
+ execute(args)
+ execute_setup(args)
+ execute_setup(args)
+ execute_check(args)
+ execute_irc(args)
-- handle bash input
agent
config
-config
+ __init__()
+ save()
core
+ check()
+ check_tribler()
+ check_tunnel_helper()
+ select_provider(config,dna)
+ update_offer(config)
+ attempt_purchase(config, dna)
+ install_vps(config, dna)
+ update_choice(config, dna)
dna
normalize()
mutate()
controllers
electrum_controller
+ create_wallet()
tribler_controller
+ start()
+ is_running()
+ get_mc_balance()
+ get_mc_balance()
+ put_ask()
+ put_bid()
cloudomate_controller
+child_account(index: int): account
+status(provider)
+get_ip(provider)
+setrootpw(provider, password)
+get_network_fee()
+pick_provider(providers)
+pick_option(provider
+update_offer(config)
+calculate_price(provider, option)
+purchase_choice(config)
+place_0ffer(price,config)
Tribler
Cloudomate