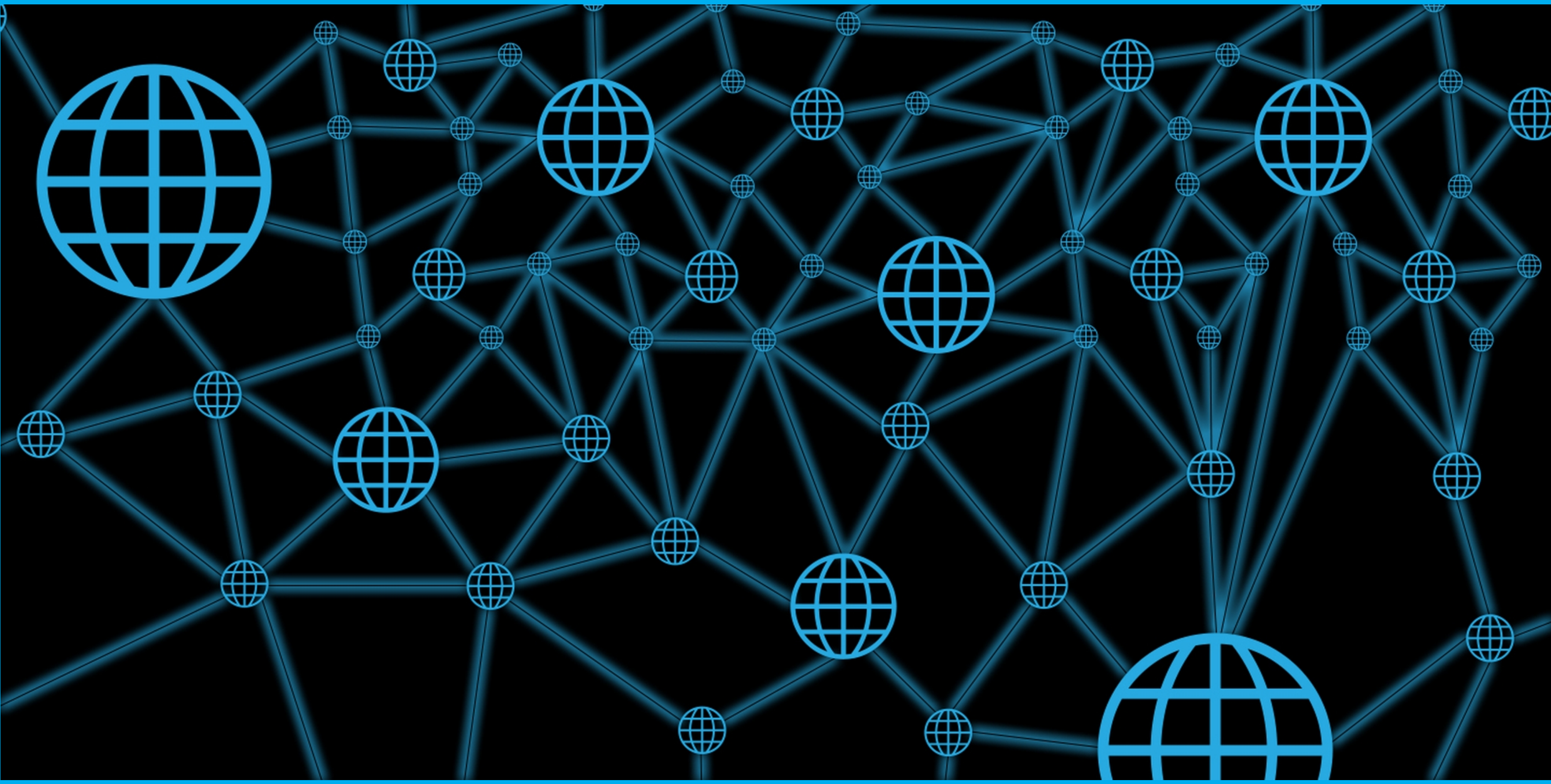
PlebNet

Botnet for good

R. van den Berg
J. Heijligers
M. Hoppenbrouwer



PlebNet

Botnet for good

by

R. van den Berg
J. Heijligers
M. Hoppenbrouwer

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be presented publicly on Monday July 3, 2017.

Project duration:	April 24, 2017 – June 26, 2017	
Thesis committee:	H. Wang,	TU Delft, supervisor
	Dr. Ir. J. Pouwelse,	TU Delft, coach
	M. de Vos,	Tribler

TU Delft

Preface

The Bachelor End Project is the final step towards completing the bachelor of Computer Science at Technical University of Delft. A group of students works on a problem posed by a client in order to demonstrate their proficiencies as individuals and as a team. The problem, which is both technically challenging as well as of importance to the client, is researched and a solution is presented. This solution is then designed, implemented and tested taking into account all techniques and knowledge acquired during the bachelor. At the end of the project the final solution is presented and evaluated by a committee consisting of the client, the TU Delft coach and the TU Delft supervisor.
In this document we describe our methods, designs and solutions for the problem posed by the Tribler Team.
We would like to thank dr.Ir. Johan Pouwelse and Martijn de Vos for their guidance during our Bachelor End Project.

R. van den Berg
J. Heijligers
M. Hoppenbrouwer
Delft, January 2017

Summary

Throughout this project two components have been created: Cloudomate and PlebNet. Cloudomate is a program that can purchase virtual private servers (VPS) from a number of providers with Bitcoin, it is accessed through a command line interface and published as Python library. PlebNet is the implementation of an autonomous self-replicating entity. It uses Cloudomate to purchase servers through which it provides bandwidth for anonymous downloading as exit node in exchange for reputation. This reputation is then sold on the Tribler market so that the autonomous entity can purchase new servers to join the process. Effectively creating a botnet for good.

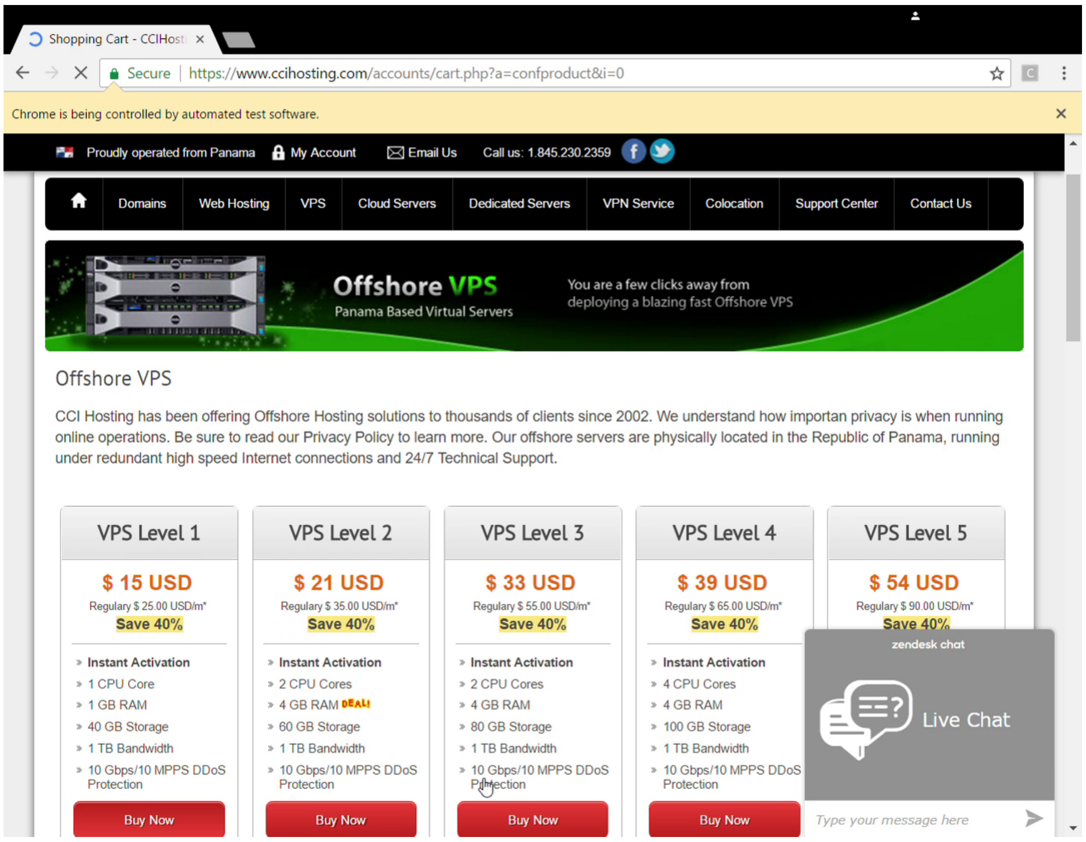
Contents

1 Introduction

For this Bachelor End Project, BEP, we were asked to create an Internet-deployed system which earns money, replicates itself and has no human control. To earn money we help others protect their privacy using Tribler, developed by the Delft University. This is done by earning reputation, which is represented by Trustchain coins. We will sell our reputation for Bitcoin.
Our system will earn income in the form of currency, sell this currency for Bitcoin, use the Bitcoin to buy a server automatically and as a final step install itself on the server. The system will also have a simplistic form of genetic evolution. This genetic evolution will influence which server provider it chooses. When the system has favorable past experiences with a provider it will be more likely chosen. This makes the system more robust.

1.1. Servers

Servers are considered an Internet Hosting Service. With servers you can run your own copy of an operating system and you can install almost all software that can run on the operating system. There are 3 types of servers available for the regular consumer: Shared Hosting, VPS Hosting and Dedicated Hosting. These types of servers can be seen as different types of restaurants. In this analogy food can be seen as the server resources, such as CPU time, memory and disk space, and the chefs can be seen as the hardware.
Shared Hosting is equal to a buffet. Everyone at the restaurant shares all the food. You can take what you want as long as the chefs can produce the food. Every account on the server shares all the resources with all the other accounts on the servers.
VPS Hosting is similar to a normal restaurant. Everyone still has to share the chefs, but you can order your own food. The chef will give you the amount of food that you want. No one can take any food from your plate. The overall resources of the server are shared, but everyone has a dedicated portion.
Dedicated Hosting is like having a private chef. He is going to make what you want and you do not have to share him with anyone. The entire server and all the resources are yours to use. There is no other account that you have to share with.
For our project we are using VPS Hosting. As can be expected Dedicated Hosting is expensive, while VPS Hosting is much more affordable. The reason we choose VPS Hosting over Shared Hosting is the fact that we get a dedicated portion. We want our program to be able to run consistently and with VPS Hosting we have influence over the amount of resources that we are getting. We can choose the providers that will give us what we need for our project to run effectively.
VPS Hosting allows for a dedicated portion of the resources. Therefore the less resources you need the less you have to pay. When looking at a VPS provider, you will see multiple VPS options. Figure 1.1 is an example of the VPS options from the provider CCIHosting [7]. During our project we will look at the resources we need and use that to determine for every provider which option is most suitable.


Shopping Cart - CCIHost ×

Secure | https://www.ccihosting.com/accounts/cart.php?a=confproduct&i=0

Chrome is being controlled by automated test software.

Proudly operated from Panama | My Account | Email Us | Call us: 1.845.230.2359

Domains | Web Hosting | VPS | Cloud Servers | Dedicated Servers | VPN Service | Colocation | Support Center | Contact Us

Offshore VPS
Panama Based Virtual Servers
You are a few clicks away from deploying a blazing fast Offshore VPS

Offshore VPS

CCI Hosting has been offering Offshore Hosting solutions to thousands of clients since 2002. We understand how importan privacy is when running online operations. Be sure to read our Privacy Policy to learn more. Our offshore servers are physically located in the Republic of Panama, running under redundant high speed Internet connections and 24/7 Technical Support.

VPS Level 1	VPS Level 2	VPS Level 3	VPS Level 4	VPS Level 5
$ 15 USD	$ 21 USD	$ 33 USD	$ 39 USD	$ 54 USD
Regulary $ 25.00 USD/m*	Regulary $ 35.00 USD/m*	Regulary $ 55.00 USD/m*	Regulary $ 65.00 USD/m*	Regulary $ 90.00 USD/m*
Save 40%	Save 40%	Save 40%	Save 40%	Save 40%
» Instant Activation	» Instant Activation	» Instant Activation	» Instant Activation	
» 1 CPU Core	» 2 CPU Cores	» 2 CPU Cores	» 4 CPU Cores	
» 1 GB RAM	» 4 GB RAM DEAL!	» 4 GB RAM	» 4 GB RAM	
» 40 GB Storage	» 60 GB Storage	» 80 GB Storage	» 100 GB Storage	
» 1 TB Bandwidth	» 1 TB Bandwidth	» 1 TB Bandwidth	» 1 TB Bandwidth	
» 10 Gbps/10 MPPS DDoS Protection	» 10 Gbps/10 MPPS DDoS Protection	» 10 Gbps/10 MPPS DDoS Protection	» 10 Gbps/10 MPPS DDoS Protection	
Buy Now	Buy Now	Buy Now	Buy Now	

zendesk chat
Live Chat
Type your message here

Figure 1.1: VPS plan selection

1.2. Tribler

"Create a censorship-free Internet" [6]
Everything that is moved around the Internet is vulnerable to both surveillance and censorship. Both governments and Internet providers have the ability to view everything that happens from central infrastructure points. This is concerning due to the potential of abuse. In time of crisis governments have shown their ability to disable communications networks. The Egyptian government demanded telecommunications companies to shut down their mobile networks and broadband connections during the Arab Spring. Because the providers were forced to comply, Egypt vanished from the digital map. Tribler aims to fight this [6].
Tribler is a peer-to-peer file sharing system created by Delft University of Technology. In a peer-to-peer system, the peers are computers that are connected with each other on the Internet. The system allows for files to be shared on the network. Even though the peer-to-peer system allows for an easy way to share files with each other, it also has made it easier to pirate software and other media. Some of the more popular peer-to-peer networks are BitTorrent, eDonkey and Gnutella [3].

1.2.1. Exit Node

Tribler aims to protect privacy. BitTorrent is a fast network, however it offers no privacy protection. When using BitTorrent you can be easily tracked if your downloading 'controversial' content. By introducing a proxy layer direct contact with the BitTorrent network is gone. However you do need to trust the proxies inside the proxy layer as they can see exactly what you do. Using three proxy layers increases the privacy given to you. A single proxy in one of the proxy layers cannot see what you are downloading. On Tribler this proxy protection is used for both the downloader and the uploader [5].
To access the Tribler network, you need an exit node. The exit node is a publicly visible entryway into the anonymous Tribler network. Being the bridge between an anonymous and public network leaves the exit node vulnerable. For this reason not a lot of people are willing to be an exit node. This means there is a lack of exit nodes. This project aims to create exit nodes that can be used to access the Tribler network. Currently

Tribler has a number of exit nodes located at LeaseWeb. All of the exit nodes are at one location making it a centralized network. To fortify the network of exit nodes this project will spread the exit nodes around creating a decentralized network.

1.2.2. Market

The use of the proxy layers in the Tribler network affects the download speed that can be achieved. In order to incentify people to seed, Tribler uses bandwidth as a currency. Basically by uploading you 'earn' bandwidth and by downloading you 'sell' bandwidth. The Tribler market lets users buy and sell bandwidth in the form of reputation.
The market is a distributed system, so there is no centralized point where users can 'meet' to trade. Instead local markets are created. A local market consist of the people a user has interacted with before. If the interaction between two users was positive, a user can pass along the trade information to its own local market. This will then increase the size of the user's local market.
As of the start of the project the market is still in development. Once the market and our system go live, we will have a chance to determine the rate of exchange between reputation and bitcoin. However since it is a free market, the price is determined by demand and supply. Therefore we expect to see a race to the bottom.

1.3. Ethical Considerations

Peer-to-peer networks are very controversial. They can be useful to share files and software, however some (or most) of these files are copyright protected. A peer-to-peer network also offers a censorship resilience, because it is a decentralized system. Once something is on a peer-to-peer network it is nearly impossible to remove. This is useful to uphold freedom of speech. However a peer-to-peer systems do have ethical challenges. Historically a significant portion of the data on peer-to-peer networks consist of copyright protected content [27].
Pirating copyright protected material is illegal, however it might not be unethical. There is a widespread acceptance of pirating music and movies. It seems to be more like recording a song off the radio and less like stealing a CD from a record store [28].
A system that creates exit nodes for Tribler brings some ethical questions with it. Exit nodes create bandwidth for people to share files including the copyright protected ones. However Tribler and the exit nodes that this project is producing help to protect an essential freedom people should have: privacy. People should be able to use the Internet without being watched or controlled. If someone has a dissident opinion they should not be censored by a government or any other entity. We are therefore of the opinion that our project is ethically acceptable.

1.4. Previous work

Last year another BEP group, TENnet [26], worked on this project. Before we started with the project we analyzed what they had done and what was usable for our vision.
While the previous group was working on their project another group was developing a decentralized marketplace for Tribler. Due to this the group was unable to connect their bot network with the decentralized marketplace. Therefore we could not use any of their code relating to the market.
The bot network that TENnet built had 2 main functions besides the decentralized marketplace: buying servers and a basic genetic algorithm.
To let the bot network buy servers autonomously they used Selenium. With Selenium they open a browser and go to the web-page to order a new VPS. Using Selenium they are able to fill the sign-up forms.
For the basic genetic algorithm they used a weighted array. The array contained all the VPS hosts they are using. When a VPS host is used a mutate rate is added to that VPS in the normalized array. With this technique their bot network learns which VPS host it finds preferable.
For our project we had to decide whether or not the work of the previous group was usable with our vision. After some experimentation, see chapter 3, we found some problems with Selenium. It is not able to run headless, it has a lot of dependencies and is too heavy-weight for our vision. Furthermore the genetic algorithm used by the previous group only used positive influence and only used 1 parameter to change their weighted array. However we did use this as a basis for our own genetic algorithm.
Since the VPS buying was the largest part of the previous project and our problems with it, we decided to start with a clean slate while learning from the previous group.

2 Product Design

For our project we needed a design. How are we going to build a self replicating system? In order to understand what we needed we started with a Moscow list to discover all the requirements our project has. After we knew the requirements we made a system design to understand all the components our project will consist of. And finally we made roadmap for our agent to know which steps are needed for our system to perform.

2.1. Product Requirement

To understand what is needed we created a list of the product requirements for this project. The requirements are put in a Moscow list. This list represents the must have, should have, could have and won't have requirements of the product. Each item in the Moscow list is explained below.

2.1.1. Moscow

Must Have

1. The agent must earn reputation
2. The agent must sell this reputation on the market for bitcoin
3. The agent must buy new servers
4. The agent must have a genetic algorithm
5. The agent must have multiple VPS(Virtual Private Server) options

Should Have

1. The agent should have error notifications
2. The agent should send a postcard with server information
3. The agent should have a bandwidth based evolution algorithm
4. The agent should predict its own life expectancy
5. The agent should have 6 different VPS options

Could Have

1. The agent could have 15 different VPS options
2. The agent could share evolution data with its children
3. The agent could have an advanced trading system
4. The agent could create an issue on Github on error

5. The agent could update itself mid life cycle

Won't Have

1. The agent won't extend the lease on the current server
2. The agent won't update the VPS purchase forms automatically
3. The agent won't use other currencies
4. The agent won't combine resources with other agents

2.1.2. Must Have

Earning reputation Tribler has a reputation system. With this system the agent will be able to earn reputation. This reputation is represented by Trustchain coins. Trustchain coins are a way to register the users upload/download ratio. In order to earn a lot of reputation the agent will have to upload a lot of data, while downloading as litle as possible.

Selling reputation Tribler has a decentralized market. On this decentralized market the agent will be able to sell reputation. After the agent has earned Trustchain coins, it has to sell this for Bitcoin on the market. The bitcoins the agent earns can then be used to sustain itself by creating children.

Buying servers In order to make this a viable product it has to be able to sustain itself. For the agent to sustain itself it has to be able to move to another server when the lifecycle of the current server is ending. To achieve this it has to buy servers with the bitcoins it has earned by selling reputation. After it has bought a new server it has to install itself on the new server.

Genetic algortihm When the agent creates a child, it should pass on data to the new instance. This will be done by the evolution algorithm. This algorithm can be seen as a simple form of genetic evolution. Reproduction will be done by buying a server for a child. This child should get the evolutionary data from its parent.
At the end of the lifecycle of the server the agent has to move to a new server. When the agent moves to the new server it brings its evolutionary data as well as its wallet. By bringing the wallet itself, the transaction fees can be avoided and the Bitcoins the agent has earned won't be lost.

Multiple VPS options For the product to be viable the agent needs multiple options for a VPS. If the agent gets booted by a VPS provider, it is imperative that the agent has other options. Otherwise the whole product will die.

2.1.3. Should Have

Error notifications The agent should send error notifications to the client. If the agent runs into an error while buying a VPS, or at any other moment, it should notify the client. It is possible that the VPS provider has changed the sign up form. The client can then correct the error.

Postcard The agent should send a postcard with server information. When the agent has bought a new server, it should send server information to the client. The client can keep track of the agents that are running.

Bandwidth based genetic algorithm The agent should have a genetic algorithm based on bandwidth. For a system that deals in bandwidth it is important that it can use a lot of data. Therefore using bandwidth to evolve is important

Predict life expectancy It is useful for an agent to know how long it can still sustain itself. This information can be used to determine the strategy the agent will deploy. With a long life expectancy it can be beneficial to create a child, while with a low life expectancy creating a child is not a good idea.

6 VPS options As stated before in order for the product to be viable the agent needs multiple options for a VPS. With 6 different VPS options the agent should have enough options to survive even when one or more options are down.

2.1.4. Could Have

15 VPS options As stated before in order for the product to be viable the agent needs multiple options for a VPS. With 15 different VPS options the agent should have more than enough options to survive even when one or more options are down.

Share evolution data The agent could share evolution data with its children even after the conception. This way the children won't just evolve through their own experience, but through the experience of their parent as well.

Trading System The agent could have an advanced trading system. With the agent selling reputation for Bitcoin, it should have a trading system. The agent should be able to anticipate the market. This way it will be able to get the best price.

Create issue As stated before the agent should send error notifications to the client. However instead of sending an error notification, the agent could create an issue on Github.

Update mid lifespan It is possible that an agent runs into errors while running on a server. These errors can be life threatening if not addressed. However in order for the agent to survive it than has to be able to update mid lifespan instead of updating while moving to a new server.

2.1.5. Won't Have

Extend lease The agent won't extend the lease of the current server. Buying a new server gives the same result. It is therefore not worth to spend time implementing and testing code that can extend the lease.

Update VPS purchase form If the VPS provider changes their application form the agent won't update itself to adjust. To implement this would take a lot of time, because it is incredibly complicated.

Other currencies The agent won't use currencies besides Bitcoin (and Trustchain coins).

Combine resources with another agent When an agent won't make it to the next lifespan it won't combine resources with another dying agent to buy one server. It is very complicated to find matching pairs.

2.2. System Design

One of the things we saw in the previous group was that their project had multiple responsibilities. To avoid that we made a system design (figure 2.1) to better understand which components we needed and how they would interact.

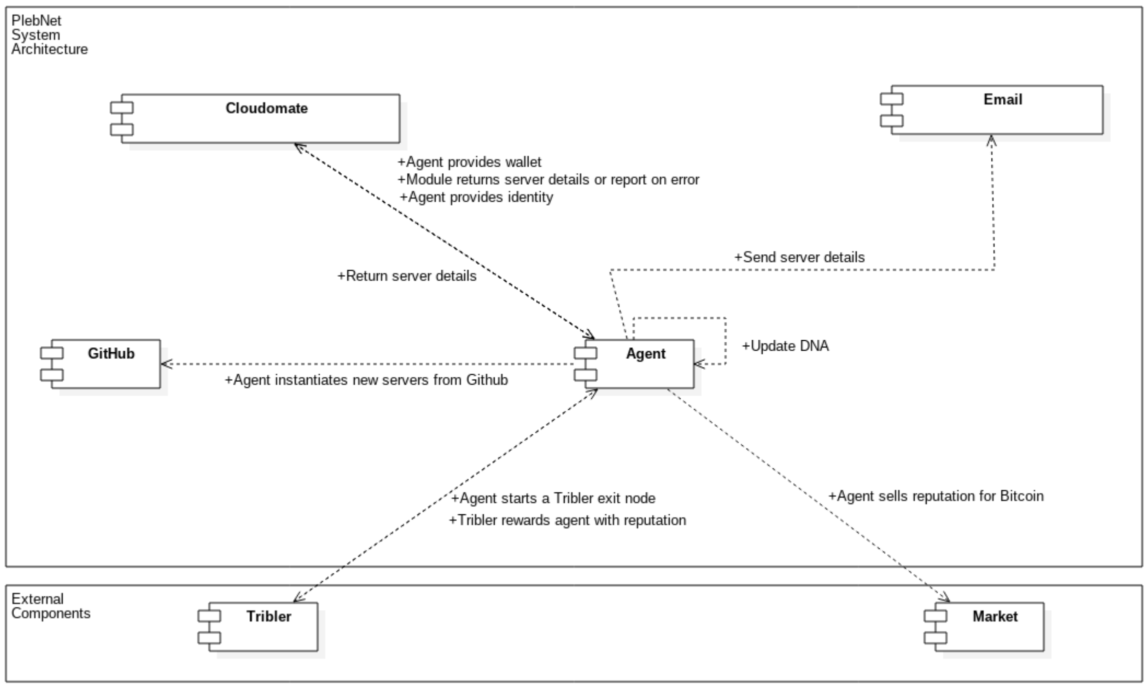

PlebNet System Architecture
Cloudomate
Email
+Agent provides wallet
+Module returns server details or report on error
+Agent provides identity
+Send server details
+Return server details
GitHub
Agent
+Update DNA
+Agent instantiates new servers from Github
+Agent starts a Tribler exit node
+Tribler rewards agent with reputation
+Agent sells reputation for Bitcoin
External Components
Tribler
Market

Figure 2.1: system design

The basis of our system is the agent. This is the core of our system, which decides what has to be done. With this logic it acts as a command center. The agent has access to the other components which will help it complete its tasks.
The agent has two external components: Tribler and the Tribler market. The agent uses Tribler to earn reputation by starting an exit node. The Tribler market will allow the agent to sell reputation for Bitcoin. Both Tribler and the Tribler market are used te generate revenue. Without these components our system would not be able to replicate.
The agent has three internal components: Github, Email and Cloudomate. Github is used to make sure the agent is running the most recent version of the code. It ensures that if any problems arise, the client can fix them and the agent will be able to adapt.
Email is used to communicate with the client. It sends server details like IP address, root password and evolutionary data. This allows for the client to keep track of the exit nodes that are running.
Cloudomate allows the agent to buy new servers. It is also the largest internal component. Cloudomate uses the identity that the agent has given it to buy a server. After it has bought a server it will return the server details the agent can then use to create a child on that server. It also has a Bitcoin wallet that the agent can use.

2.3. Agent Design

For PlebNet to be a replicating system it needs a plan. All of its agents will follow certain steps while they are online. This plan is the life cycle of the agent (figure 2.2)

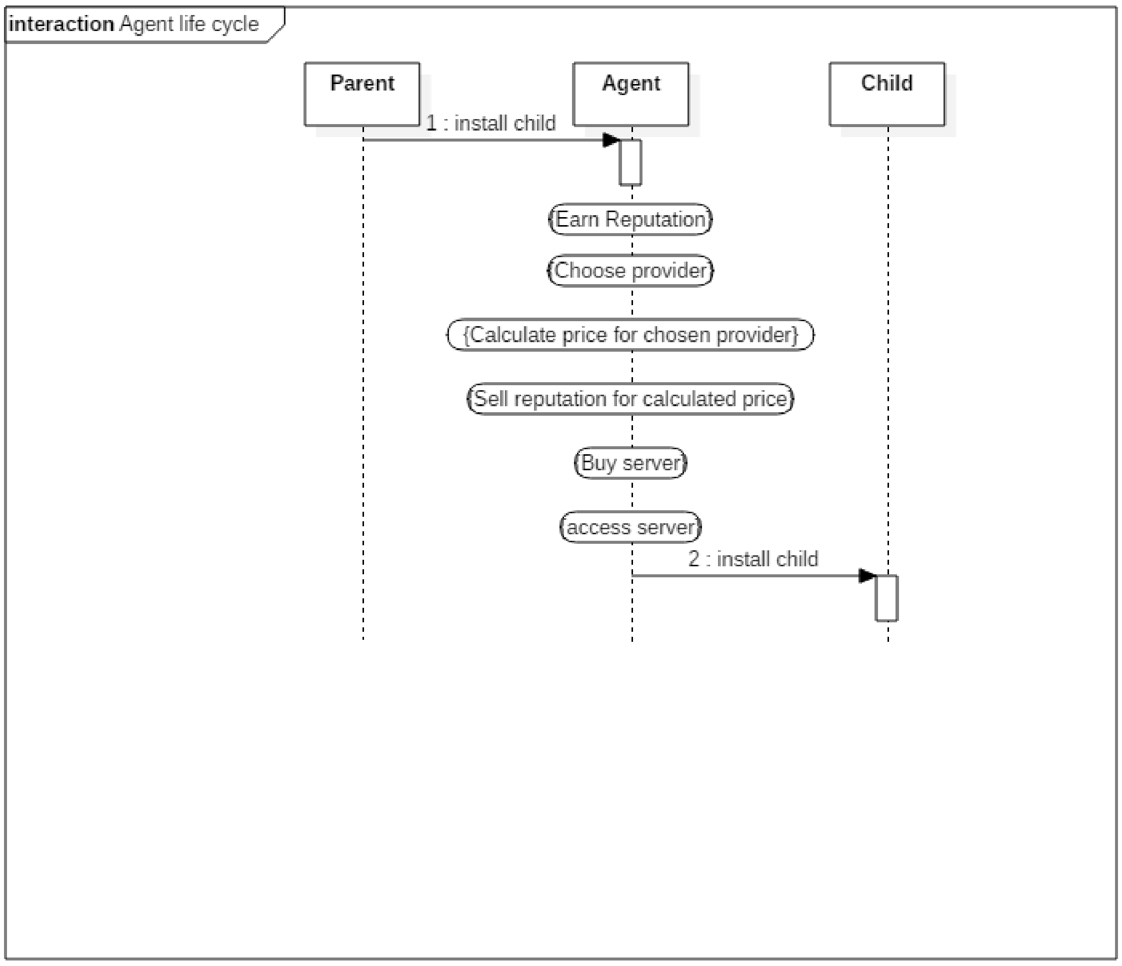


Figure 2.2: The life cycle of a PlebNet agent

To look at the life cycle of the agent we start at the moment the parent installs the agent on the server. During the install procedure the parent has started Tribler. This allows the agent to start earning reputation. Once it has started to earn the reputation it chooses a VPS provider. This VPS provider will be where the agent installs its own child. Knowing which provider it wants to buy it will calculate how much it is going to cost. This way it will have a clear view of what is needed to successfully create a child. The agent will sell its reputation on the Tribler market at the price it needs to buy the VPS it has chosen. Once it has enough money, it will buy the server and access it. After that it will install a child on the server, ending its own life cycle and starting its child's life cycle.

3
Initial experiment

For our first sprint we decided to look at the work that has been done prior. We want to know if the previous project[26] yielded anything that we could work with within our own vision of the project. Besides looking at the previous work we also look at the Tribler market. If we want to make an autonomous agent that sells on the market, we need to know how the transactions work.
As mentioned a previous group worked on this project. While analyzing their work we noted that they had one project with different responsibilities, which made it very difficult to work with. Their structure was unclear and did not have sufficient documentation. Furthermore a bulk of their project consisted of buying VPS using Selenium [2]. Therefore we decided to experiment with Selenium to see if we could use that in our project.

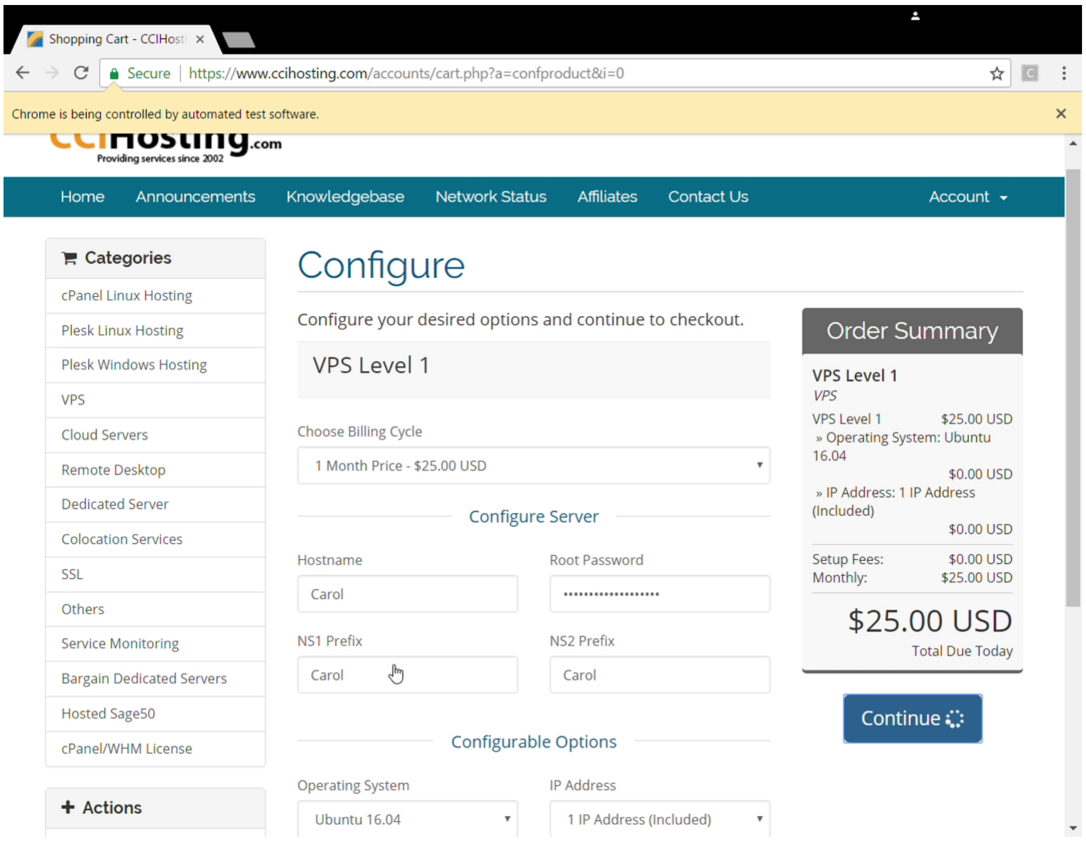
3.1. Automated Browser

Using Selenium you don't have to fill out boring forms yourself. With Selenium you can go to a website and extract information as well as fill out forms. During the process of buying VPS online, a number of forms will needed to be filled in. These forms include account creation and server settings. With Selenium it is possible to perform all of these tasks.

3.1.1. Trial 'without' Error

To show how Selenium works we ran a trial. The different VPS providers have the same basic requirements, such as first name, last name, email, billing address. Not only do they have the same basic requirements, the process of buying a server is also similar. In this example Selenium will try to buy a server from the provider CCIHosting. Because the providers have a similar process, this example is representative of all the providers we used.

Selenium starts by opening a Google Chrome browser. It uses the browser to navigate through the website and collect information. Using the URL it goes to the CCIHosting VPS plan selection page (This was shown in figure 1.1). From all the options it chooses the cheapest. After the selection Selenium is taken to the configure screen (figure 3.1). In this screen it fills in the host name, root password, ns1 prefix and ns2 prefix. This is followed by putting the server in the shopping cart. From there it goes to the checkout screen (figure 3.2). Here the personal information, such as first name, last name, address and email, is requested by the provider. After filling out all the information it is time to pay. Selenium goes to the payment screen (figure 3.3). In the payment screen it reads the amount of bitcoin that has to be paid as well as the bitcoin address.



Shopping Cart - CCIHosti ×

Secure | https://www.ccihosting.com/accounts/cart.php?a=confproduct&i=0

Chrome is being controlled by automated test software.

Configure

Configure your desired options and continue to checkout.

VPS Level 1

Choose Billing Cycle: 1 Month Price - $25.00 USD

Configure Server

Hostname: Carol

Root Password: ••••••••••••••••••••

NS1 Prefix: Carol

NS2 Prefix: Carol

Configurable Options

Operating System: Ubuntu 16.04

IP Address: 1 IP Address (Included)

Order Summary

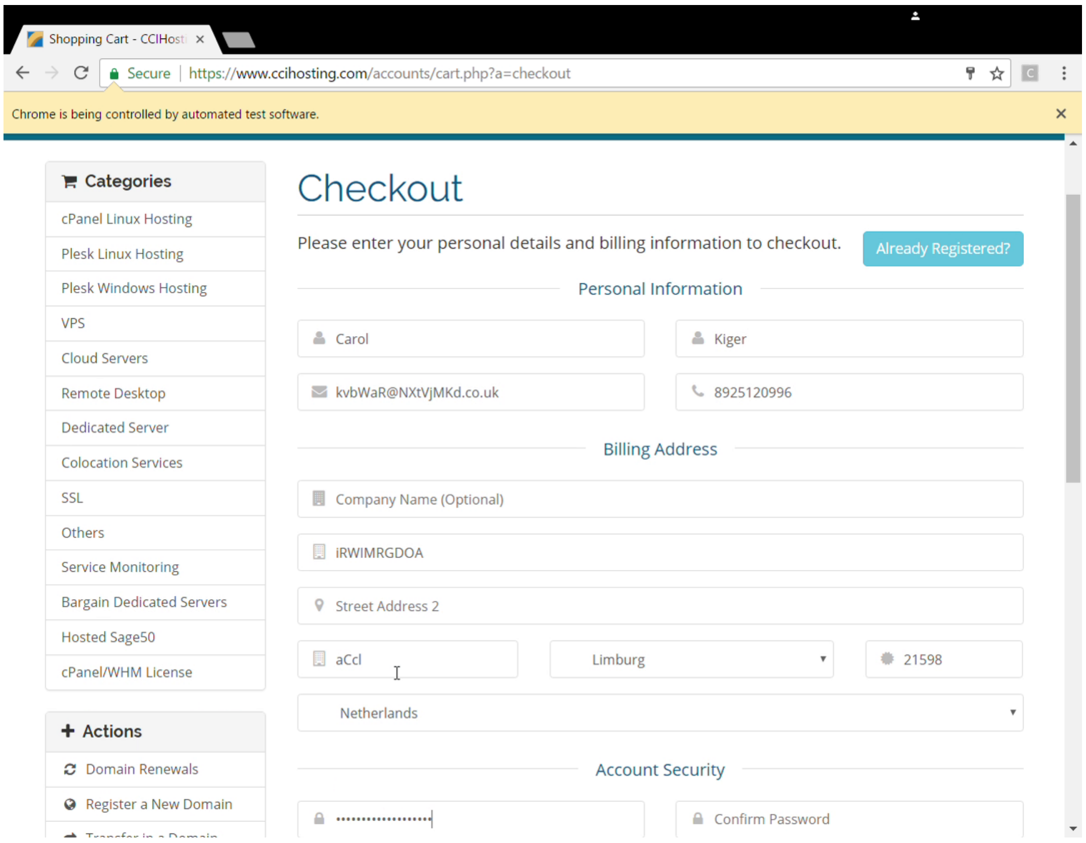
VPS Level 1
VPS

VPS Level 1	$25.00 USD
» Operating System: Ubuntu 16.04	$0.00 USD
» IP Address: 1 IP Address (Included)	$0.00 USD
Setup Fees:	$0.00 USD
Monthly:	$25.00 USD

$25.00 USD
Total Due Today

Continue

Figure 3.1: configure screen



Shopping Cart - CCIHosti ×

Secure | https://www.ccihosting.com/accounts/cart.php?a=checkout

Chrome is being controlled by automated test software.

Checkout

Please enter your personal details and billing information to checkout. Already Registered?

Personal Information

Carol | Kiger

kvbWaR@NXtVjMKd.co.uk | 8925120996

Billing Address

Company Name (Optional)

iRWIMRGDOA

Street Address 2

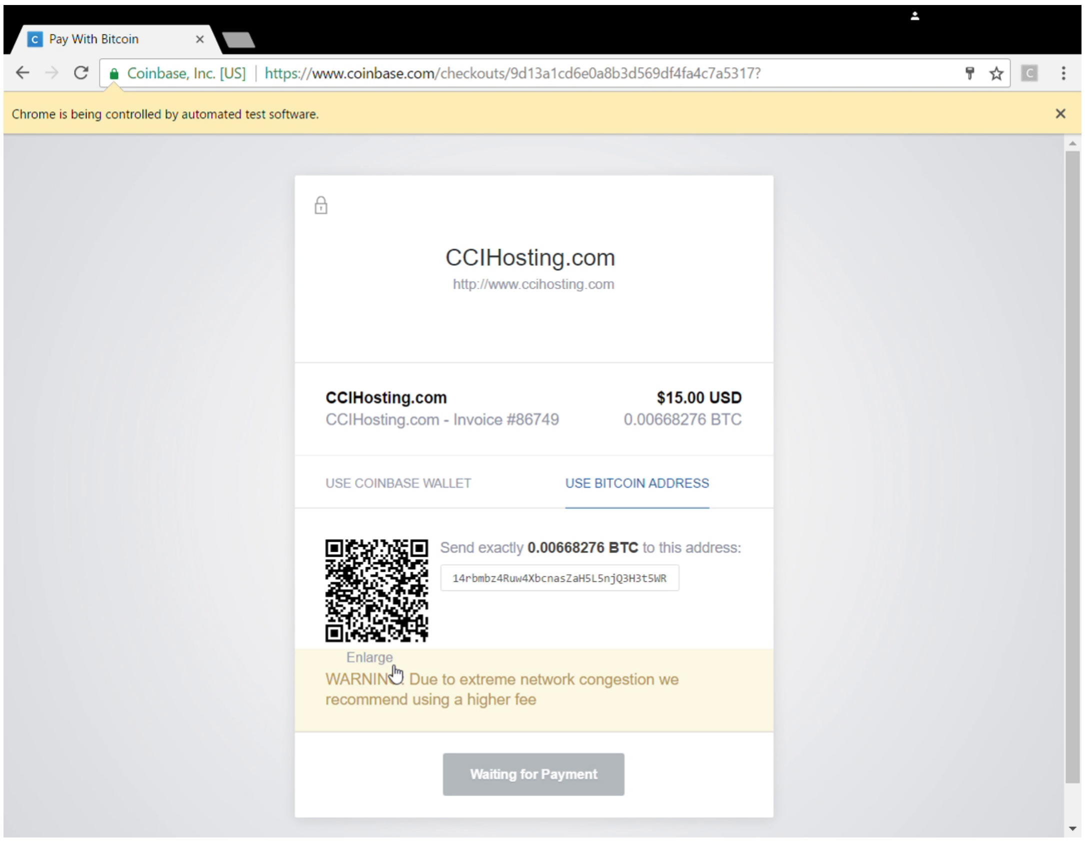
aCcl | Limburg | 21598

Netherlands

Account Security

•••••••••••••••••••• | Confirm Password

Figure 3.2: checkout, personal information


Pay With Bitcoin

Coinbase, Inc. [US] | https://www.coinbase.com/checkouts/9d13a1cd6e0a8b3d569df4fa4c7a5317?

Chrome is being controlled by automated test software.

CCIHosting.com
http://www.ccihosting.com

CCIHosting.com **$15.00 USD**
CCIHosting.com - Invoice #86749 0.00668276 BTC

USE COINBASE WALLET USE BITCOIN ADDRESS

Send exactly **0.00668276 BTC** to this address:

14rbmbz4Ruw4XbcnasZaH5L5njQ3H3t5WR

Enlarge

WARNING: Due to extreme network congestion we recommend using a higher fee

Waiting for Payment

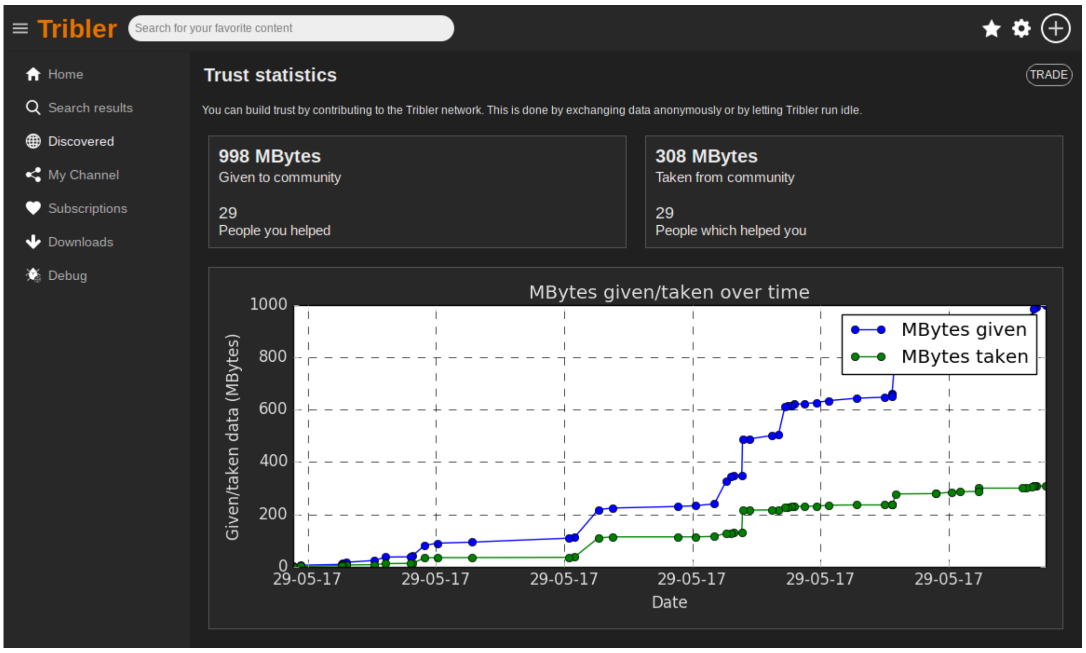
Figure 3.3: coinbase

3.1.2. Analysis

With Selenium it is possible to buy VPS autonomously. Selenium can go to a designated website. From there it is able to go through the site and fill in forms that are required to buy a server. It is also able to extract the bitcoin amount and address from the invoice.
A requirement of Selenium is that it needs to open a browser. At the start of the process Selenium opens a Google Chrome browser and uses this to go through the designated website.
For this project Selenium has to run on a VPS . However a VPS does not have a monitor and thus requires Selenium to run headless. If Selenium needs to run a graphical environment needs to be simulated. To simulate a graphical environment and run Selenium headless the previous group used Xvfb.

3.2. Test Transactions

We created a test transaction to sell bandwidth for Bitcoin in the existing code base using existing software. An important part of this project is the ability to earn money. The agent will earn bitcoin on the Tribler market. Currently the market is in development, for our test transaction we used the existing code. We were able to simulate transactions by using dummy wallets. With the use of two dummy currencies (dummy 1 and dummy 2), we executed transactions to understand how the market is going to work once it is released.



Trust statistics

You can build trust by contributing to the Tribler network. This is done by exchanging data anonymously or by letting Tribler run idle.

998 MBytes
Given to community

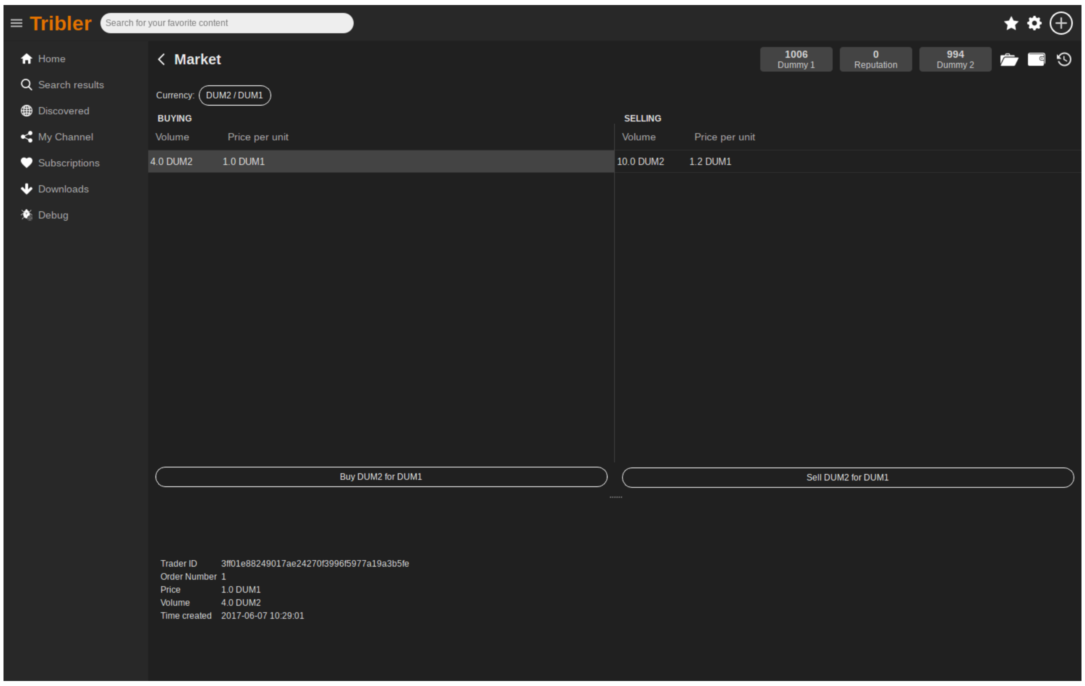
29
People you helped

308 MBytes
Taken from community

29
People which helped you

Figure 3.4: trust statistics on Tribler

The Tribler market is based around reputation, which is the trust you build with other nodes in the Tribler network. In figure 3.4 the trust statistics on Tribler are shown. The statistics are separated into two blocks. One block shows how much you have contributed to the community and the other block shows how much you have received from the community. Each block shows the amount of data that has been either given or taken and how many people you helped or helped you. From our experimentation we have concluded that the amount of people helped or helped by has no influence on your reputation. The only data that determines your reputation is the amount uploaded and downloaded. Reputation is equal to the uploaded data minus the downloaded data. Using the statistics from figure 3.4 the amount of uploaded data is 998 Mbytes and the amount of downloaded data is 308 Mbytes, this means the amount of earned reputation is 690. The earned reputation can be traded for bitcoin. In the top right corner of the page is a trade button, which brings you to the Tribler market.
In the Tribler market (figure 3.5) reputation can be sold for Bitcoin. During the experiment we were able to use dummy wallets with the 2 dummy currencies. The market shows your wallets as well as your reputation. In the market you can either make a bid or an offer. The transaction offers consist of a volume and a price per unit, so if the volume is 10 and the price per unit is 2 then the total will be 20. The bids and offers are shown on their respective side. If you click on a offer, you can see the offer information: traderID, order number, price, volume and time created.
If an offer and a bid match, a transaction is started. This can be seen in figure 3.6. For a match to be made the offer and the bid don't have to be exactly the same. It is possible that someone offers 10 reputation for 3 dummy 2 per unit and someone else wants to buy 5 reputation for 3 dummy 2 per unit. In this case a transaction is made and the offer is then lowered to the remaining reputation, which is 5 in this case. One of the issues with the market is that each exchange has 2 markets. You can trade reputation for Bitcoin and you can trade Bitcoin for reputation. If an offer of 3 reputation for 1 Bitcoin is made in the first market and an ask of 1 Bitcoin for 3 reputation in the second market, there will be no transaction. After a transaction is completed both parties receive a notification (figure 3.7).



BUYING

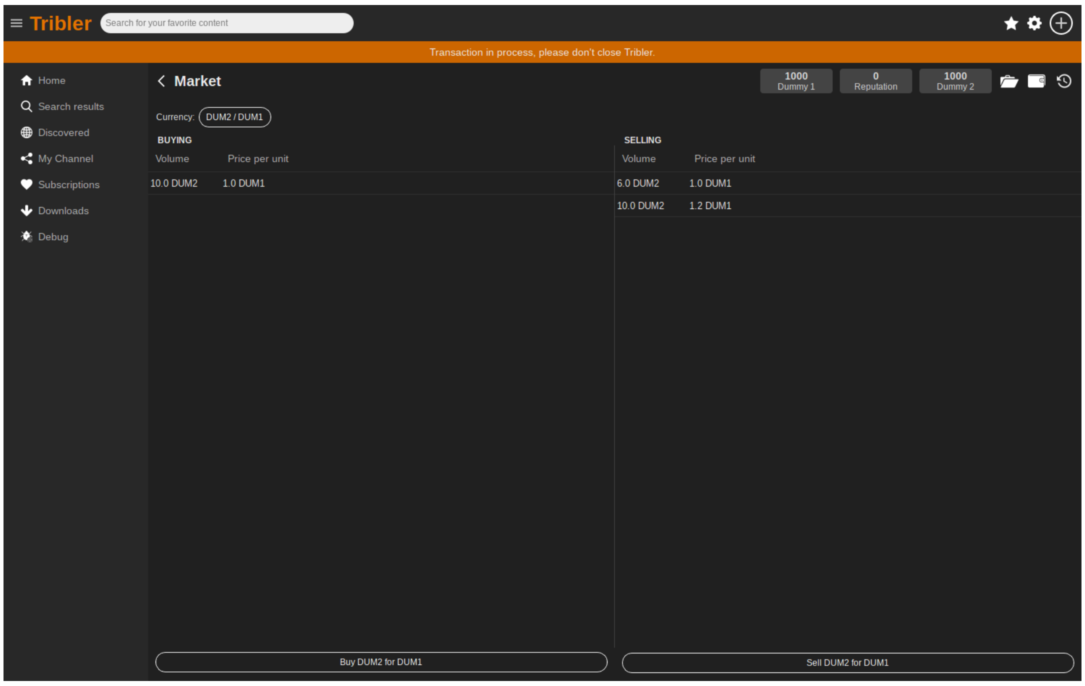
Volume	Price per unit
4.0 DUM2	1.0 DUM1

SELLING

Volume	Price per unit
10.0 DUM2	1.2 DUM1

Trader ID	3ff01e88249017ae24270f3996f5977a19a3b5fe
Order Number	1
Price	1.0 DUM1
Volume	4.0 DUM2
Time created	2017-06-07 10:29:01

Figure 3.5: the Tribler market



Transaction in process, please don't close Tribler.

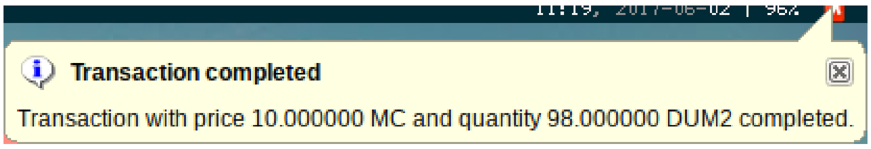
BUYING

Volume	Price per unit
10.0 DUM2	1.0 DUM1

SELLING

Volume	Price per unit
6.0 DUM2	1.0 DUM1
10.0 DUM2	1.2 DUM1

Figure 3.6: a transaction in progress

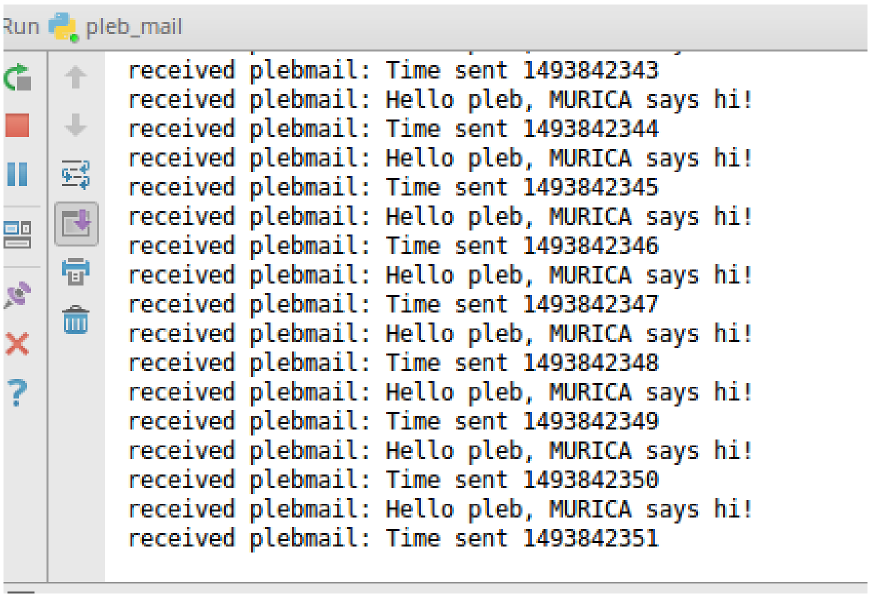

11:19, 2017-06-02 | 96%

Transaction completed

Transaction with price 10.000000 MC and quantity 98.000000 DUM2 completed.

Figure 3.7: a completed transaction

3.3. Decentralized Communication



```
received plebmail: Time sent 1493842343
received plebmail: Hello pleb, MURICA says hi!
received plebmail: Time sent 1493842344
received plebmail: Hello pleb, MURICA says hi!
received plebmail: Time sent 1493842345
received plebmail: Hello pleb, MURICA says hi!
received plebmail: Time sent 1493842346
received plebmail: Hello pleb, MURICA says hi!
received plebmail: Time sent 1493842347
received plebmail: Hello pleb, MURICA says hi!
received plebmail: Time sent 1493842348
received plebmail: Hello pleb, MURICA says hi!
received plebmail: Time sent 1493842349
received plebmail: Hello pleb, MURICA says hi!
received plebmail: Time sent 1493842350
received plebmail: Hello pleb, MURICA says hi!
received plebmail: Time sent 1493842351
```

Figure 3.8: PlebMail in action

We were able to send messages across the globe using decentralized communication. With Dispersy [4] we were able to receive a message from a server in America (figure 3.8).
Distributed Permission System, or Dispersy, is a messaging system that allows decentralized systems to communicate. Dispersy itself is also decentralized, every node in the system runs the same code and perform the exact same tasks. By doing this all nodes are of equal importance and this makes the system robust. All data is forwarded by the node, this will make sure the data is received by all nodes eventually.
Using Dispersy to create PlebMail is one of our could haves. We were able to create a successful test, however due to time constraints we were unfortunately unable to implement a working version of PlebMail into our project.

3.4. Conclusion experiment

After all of our experiments we were able to decide what we wanted for our project. Looking at the work from the previous group, which consisted mostly of Selenium, we decided to start with a clean slate. The previous group had one project with different responsibilities, we wanted to separate this. Buying VPS automatically is not something that is only useful for our project. For this reason we decided to publish our VPS purchasing code as library.
Selenium is very dependent on the interface of the websites. If they make a change the code will fail. Furthermore Selenium is very high level and has a lot of dependencies. Since we wanted to create a public library, we were looking for something low level that does not have a lot of dependencies.

4 Cloudomate

Computational Infrastructure is an essential requirement in creating a network of exit nodes. Accessing this infrastructure without intervention by any human operator necessitates the ability for agents to buy servers. The goal of the second sprint is to create autonomy for the agent in the form of a robust module. The module will provide the means to register an account at a hosting provider, purchase a server, pay the hosting provider and retrieve the authentication details of the server. For its functionality to automate the creation of a cloud of servers we name this module Cloudomate.

Cloudomate

In this chapter we first discuss the vision for a module that automates the purchase process. Second we elaborate on the requirements that are set on the implementation of Cloudomate. Third we show the design of the module and interaction with other modules. Last we provide considerations encountered during the implementation.

4.1. Module Vision

PlebNet requires autonomous, distributed operation to be resilient. By giving it access to a wide range of services it can use to operate the exit node network, we can increase the probability of it finding a suitable set of hosting providers that offer plentiful, reliable bandwidth. Large hosting providers offer documented APIs to purchase services, but would limit the servers it acquires to a small subset of providers that may have unfavorable aspects. For smaller providers, more often than not, there is no API available. In some cases it is forbidden in the Terms of Service to purchase servers by an automated process. By automating the purchase and registration processes that smaller hosting providers have in place, Cloudomate opens up a wide range of services otherwise unavailable in a programmatic fashion.
The time available on this project only allows for the implementation of a limited amount of hosting providers. And these implementations will only function until a hosting provider or payment provider updates their processes. This means that there is a constant effort required to keep the module up-to-date. In a small team like Tribler this manpower may not always be available. By making the project Open-source we do not only aim to provide this functionality to the general public after the Bachelor End Project finishes, but others will be able to contribute to the project in the form of updates to implementations, adding new hosting providers and payment providers. If the project turns out to be succesful it may also garner the interest of the hosting providers themselves, as the module is setup to provide them with new customers that have a particular use case involving the automation of computational infrastructure.

4.2. Module Requirements

Figure 4.1: List of Functional Requirements

- Use commandline to access functions
- View supported hosting providers
- View details of servers previously purchased
- Register an account
- Purchase a server
- Pay for server using bitcoins

Figure 4.2: List of Non-Functional Requirements

- Dependencies are lightweight
- Implement functionality for five to ten hosting providers
- Documentation of Cloudomate is provided
- Available as a PyPi module
- General use explained in PyPi

4.3. PIP

The previous group used Selenium for purchasing VPS instances. This turned out not to be a very streamlined approach. Even though implementing a new provider is relatively easy through Selenium, the approach is error prone and resource heavy due to its graphical rendering in an actual browser. Especially on servers, on which no graphical environment is present, this requires unncessary hacks such as using a virtual framebuffer to render graphical components. We decided to make our implementation more robust and ready for real world usage. As a result, Cloudomate implements a lower level abstraction and uses a command line interface for easy accessibility. The project has been published as Pip module: Cloudomate is available for installation through the official Python package manager [15].

4.4. Webscraping

VPS hosters generally don't offer an API to activate their services. Therefore webscraping is the only way to purchase servers from those providers in an automated fashion. Where Selenium is able to do this on a high level including JavaScript execution and graphical rendering, a lower level approach that involves HTML parsing and creating HTTP requests offers a more robust alternative. We decided Scrapy [24], a Python library for web scraping, made a good fit for this. Our Scrapy project implements three phases: retrieving VPS options offered by the provider, purchasing a chosen option, and retrieving server details of a purchased option. The Scrapy framework itself provides a convenient, easily extensible, command line interface for separate scraping tasks.

4.5. Hosting Providers

An essential part of Cloudomate is picking the right hosters. The major constraint of VPS providers is their acceptance of Bitcoin payments. Amongst providers that do accept Bitcoin the most important aspects for our project are bandwidth value and the strictness of Terms of Service. As reputation mining requires a lot of bandwidth and our project isn't very heavy on other resources, the other resources like CPU cores and RAM weren't a big factor in our choice. Unlimited bandwidth under a fair use policy is offered by Legion Box [18], Rockhoster [23] and Linevast [19], which makes these providers interesting for our project.

4.5.1. Strict providers

Some providers don't allow torrents to be downloaded on their servers, and some have a very strict DMCA policy. We would not want to use servers which are quickly taken down due to exit node usage. Ramnode [22], the first implemented provider, on which testing and exploration is done, banned our IP address due to repeated registration with fake details. After being turned down as response to a support ticket, it has been decided to stop supporting Ramnode in Cloudomate.

CrownCloud[16] appeared to use a very strict DMCA handling policy: when running Tribler as exit node for testing purpose, our CrownCloud instance was suspended within a minute of receiving the first DMCA complaint.

Table 4.1: Hosting provider offerings

Name	Price/month	Traffic (TB)	Bandwidth (Mbit/s)	Storage (GB)	RAM (GB)
CrownCloud	$6	2	1000	30	1
Legion Box	$7	unlimited	1000	15	0.5
Rock hoster	$5	unlimited	500	25	1
Linevast	€7	unlimited	1000	50	2
BlueAngelHost[14]	$8	1	1000	20	1
CCI Hosting	$15	1	10000	40	1
Pulse Servers	$10	unlimited	1000	30	1

4.6. Payments

For payments it has been decided to use Electrum[17], a command line driven Bitcoin wallet written in Python. The main reason therefore is the usage of Electrum by Tribler. Using Electrum for both the Tribler market and Cloudomate payments saves the effort to transfer money between different Bitcoin wallet implementations, and it avoids having to spend transaction costs by doing so.

4.6.1. Gateways

The majority of hosting providers use either BitPay[13] or Coinbase[1] for handling Bitcoin payments. After registration, a user is redirected to a payment page of a gateway which polls for received payments. A gateway abstraction has been made in Cloudomate which takes a URL to the payment page of a gateway and returns the desired transaction amount and address. This is then passed to Electrum code to make the payment.

5 Hardening Cloudomate

In this sprint, besides expanding the number of service providers, we have made the overall program more robust. The level of abstraction of Scrapy wasn't suitable for our project: Scrapy works as a black box API which is fit for writing simple data scrapers but isn't fit for being used programmatically by other programs. We therefore replaced Scrapy's functionality with a low level programmatic browser and an HTML parsing library.

5.1. Problems with Scrapy

Scrapy is a high level scraping framework that has a well defined way of being used. The user writes a spider that is capable of scraping one or multiple websites, after which the relevant data is parsed and either exported to a file or to a database. The program can be executed with a range of commands and options through the scrapy command line. This structure allows for clean code that only exists of the actual scraping, typical functions that would have to be implemented like having different parsers for different types of fields are already apparent in Scrapy's framework. Overall Scrapy is a well maintained and easy to use scraping framework with good documentation, however the framework does come with some limitations that eventually got our project to use a different approach.

The first limitation was the focus on command line usage and with that the lack of support for running Scrapy from a script. There is documentation about how to run Scrapy from a script, however the parsing of data is done within Scrapy's framework without a clean access point outside of the Scrapy framework. The way to start Scrapy from a script and then process the data that has been extracted by Scrapy from our own program would be to let Scrapy store the data in a file or database which would then be read by our code. This is of course not an optimal approach as writing to a file or database to transfer Python objects from one point of a program to another is unnecessarily complicated.

A second limitation of Scrapy is the inability to start scraping by script multiple times within the runtime of the program. Scrapy internally starts an asynchronous network engine before starting one or more scraping jobs and closes this network engine on finish of these jobs. When a new set of jobs is requested, Scrapy reports the network engine has already been closed and cannot be restarted. This again shows the focus of Scrapy on being executed from the command line, and not being very compatible with being run from scripts. Due to these limitations it has been decided that we could try a lower level approach, step away from the Scrapy framework and use a combination of Mechanize and BeautifulSoup instead.

5.2. Mechanize

The Mechanize[20] Python library is a stateful programmatic web browser. The library differs from a usual web browser by being completely operated programmatically as opposed to through a graphical interface or a command line interface. Mechanize is lightweight and easy to operate from scripts: parsing of web pages and forms is implemented, but not rendering or JavaScript execution. The library differs from basic network libraries in Python like urllib by being stateful and by parsing HTML. A cookie jar is used internally which is useful for our project as pages that require login will work without explicitely specifying which cookies have to be sent along with the request, as those cookies have already been saved in the cookie jar when logging in at an earlier request, and they will be sent with every subsequent request to the same domain.

5.3. BeautifulSoup

BeautifulSoup[12] is a HTML parsing library. Where Mechanize only parses HTML to the point of finding and using links and forms in the page, BeautifulSoup goes beyond that by representing the entire HTML model as a Python object. This makes it easy to extract text from web pages without having to refrain to manual parsing with, for example, regular expressions. In our project we use Mechanize to retrieve web pages and to submit web forms, and we use BeautifulSoup to extract information like IP addresses and passwords from the configuration pages of VPS providers, and also to extract available VPS options from the different provider's webpages. These libraries work conveniently together as the pages retrieved by Mechanize can be passed to BeautifulSoup in string format to be fully parsed.

For registering VPS services the form feature of Mechanize turned out to be quite convenient. Only the name value pairs had to be filled in for the different form elements, they are automatically verified by Mechanize, these name value pairs are then sent to the server as POST request when the submit method is called.

5.4. Single vendor

We have found that the large majority of VPS providers, including every single provider that we have implemented in our project, uses one single technology vendor, SolusVM[25], as the backbone of their web interface. SolusVM allows users to register and manage VPS instances through a web interface. More specifically it provides the entire web interface for VPS providers to host registration and management of their serverpark. The VPS provider creates a website listing the VPS options that can be bought, then the user is lead through the registration software which is made by SolusVM and somewhat modified by the VPS provider. Then SolusVM allows the user to pay through a number of by the VPS provider chosen payment options. SolusVM handles managing invoices and an online control panel that can be used to see information about and to control the purchased VPS instances. Curiously, the vast majority of VPS providers use SolusVM for their registration and management. For our project, this is a convenient coincidence as the registration progress from the perspective of our codebase is very similar for every VPS provider. Each provider has a page for selecting a VPS configuration, a next page for overview of the product to buy, a registration page on which user details are filled in and an acount is created, and lastly a checkout screen with a link to the payment provider. This enables us to have a similar implementation for these similar providers, and this has greatly sped up our VPS provider implementation process.

5.5. Command Line

As our program is to be controlled both programmatically and through a command-line interface, we have implemented argparse to provide a convenient way for parsing arguments and subcommands.

Cloudomate is called in the following way :

```
cloudomate [-h] <subcommand> [<args>]
cloudomate list [-h]
cloudomate options [-h] <provider>
cloudomate purchase [-h] [-c CONFIG] [-f] [-e EMAIL] [-fn FIRSTNAME]
                    [-ln LASTNAME] [-cn COMPANYNAME] [-pn phonenumber]
                    [-pw PASSWORD] [-a ADDRESS] [-ct CITY] [-s STATE]
                    [-cc COUNTRYCODE] [-z ZIPCODE] [-rp ROOTPW]
                    [-ns1 NS1] [-ns2 NS2] [--hostname HOSTNAME]
                    <provider> <option>
cloudomate status [-h] [-e EMAIL] [-pw PASSWORD]
cloudomate setrootpw [-h] [-n NUMBER] [-e EMAIL] [-pw PASSWORD] -p ROOTPW <provider>
cloudomate getip [-h] [-n NUMBER] [-e EMAIL] [-pw PASSWORD] <provider>
cloudomate ssh [-h] [-n NUMBER] [-e EMAIL] [-pw PASSWORD] [-p ROOTPW] [-u USER]
cloudomate info [-h] [-n NUMBER] [-e EMAIL] [-pw PASSWORD] <provider>
```

The following subcommands have been implemented:

```
list                List providers
options             List provider configurations
purchase            Purchase VPS
status              Get the status of the services
setrootpw           Set the root password of the last activated service
getip               Get the ip of the specified service
ssh                 SSH into an active service
info                Get information of the specified service
```

Where <provider> is one of the following VPS providers:

```
blueangelhost  https://www.blueangelhost.com/
underground    https://undergroundprivate.com
legionbox      https://legionbox.com/
ccihosting     http://www.ccihosting.com/
linevast       https://linevast.de/
pulseservers   https://pulseservers.com/
rockhoster     https://rockhoster.com/
crowncloud     http://crowncloud.net/
```

Since the number of arguments that have to be passed to purchase a provider can become quite large, a .ini-like configuration file can be created to store arguments for multiple uses. The configuration file is located in the OS-specific configuration directory. By default, a User, Address and Server section are used, and provider-specific arguments can be put in the configuration file under a section with the provider's name.

The default example configuration file looks like this:

```
[User]
email =
firstName =
lastName =
companyName =
phoneNumber =
password =

[Address]
address =
city =
state =
countryCode =
zipcode =

[Server]
rootpw =
ns1 =
ns2 =
hostname =
```

5.6. Implementation difficulties

Even though the backends of provider websites are generally similar, there have been some implementation difficulties along the way.

5.6.1. JavaScript execution

The most apparent one, compared to Selenium, was the lack of JavaScript execution. Every implemented provider has a similar first page in the registration process where fields like hostname and OS are filled in. The submit button seems like a normal `<button type="submit">` button, but in practice the button contains an onclick function which intercepts the form submission and adds parameters to the POST form request. Because Mechanize doesn't consider onclick functions, or any JavaScript for that matter, the default parsed forms must be manually extended with the form fields added by the JavaScript function.

5.6.2. The options purchase gap

There are separate commands for extracting the VPS options from the provider's main website and purchasing this option. A robust way had to be found to be able to select the right option through the purchase command without confusion. The options command lists the options that are extracted from the provider's website and numbers them in the listing so that the user can specify this option's index when calling the purchase command. As the website could change between the execution of the options and purchase commands, say the user remembered a choice from options to use through the purchase command at a later time, the purchase command extracts the options again, takes the option of the index that the user passed to the purchase command, and lists this option so that the user can verify if this is really the option that the user intended to buy.

An option specific URL which starts the registration process is also extracted from the website so that the registration process can start from this URL without danger of registering a different option than the option the user confirmed to.

5.6.3. Gateways

There are two different payment gateways that are used by our implemented providers: BitPay and Coinbase. To extract the amount of Bitcoin to pay and the address to pay this to, to purchase the service, we made an abstraction for gateways that takes an URL and returns both the amount and the address. Each provider now calls the extract_info method of their gateway.

5.6.4. Client area

As part of SolusVM, all our implemented providers use clientarea.php, a VPS configuration web interface, which looks very similar for each provider. There are some significant differences, however: some clientareas list the IP address directly on their service page, while some call an AJAX method to retrieve it. Some providers provide a way to set the root password of their VPS on the service page, some provide a way through a seperate homemade management panel, and one, CrownCloud, only provides a way to reset the root password to a random password. For CrownCloud, therefore, we have not implemented setrootpw. It would be possible to change the root password through ssh, but logging in through ssh is considered out of scope of Cloudomate as it would have implications on dependencies, cross-platform compatibility and testing possibilites. A user can retrieve the default root password through Cloudomate and manually login and change the root password instead.

5.6.5. Mail

While some VPS providers send information as the root password and login credentials to the user through email, fortunately we have found a way to circumvent this for every of the implemented providers. Optionally a temporary mail service or an own mail server could have been implemented to extract important information from provider emails, but this would require a lot of work and likely be rather error prone. Many providers list the default root password on their service page of the clientarea. CrownCloud is different as it sends the default root password only through email. The clientarea allows for reading emails that have been sent to the user through their web interface, our project uses this feature to extract the root password without requiring a separate email service to be implemented.

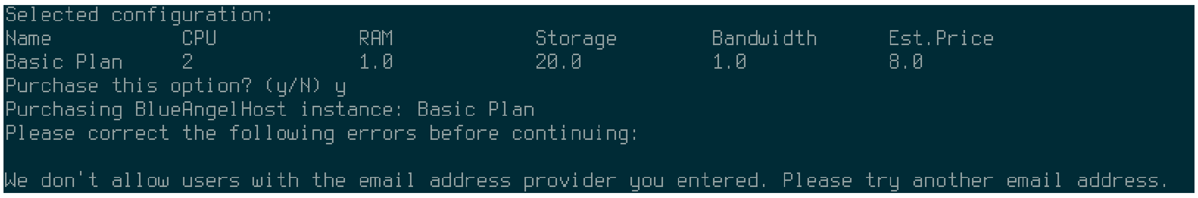
Some providers require a root password to be given through registration but actually set a different, random, password for the VPS instance. As the time between purchasing a VPS and having the payment verified and the server set up can become quite large, the root password can't be automatically set after purchase by our program. If this would be implemented, the command line program would have to keep running to periodically check the status of the purchased service for up to several hours, or even days. This wouldn't be convenient towards the user. Thus it has been decided that the user-given password is only used when the provider allows to do this, when this is not possible, the user is notified and given the ability to acquire the password that has been randomly set by the provider.

5.6.6. Getting banned

Not all providers like having the registering of their service being automated. Many providers use MaxMind, a third party service for fraud detection, as a way to block high risk registrations. While implementing code for providers we have encountered this barrier a couple of times where our IP address has been banned temporarily. This is mainly caused by registering many times from one IP address and using similar user information for multiple registrations. We would use a different IP address to continue our work and usually the next day we could use the initial IP address again. Generally we haven't had much anti-automation trouble from providers.

BlueAngelHost is an exception: for testing purposes we have used different email addresses of the domain of one of our developers: the catch-all '*@heijligers.me', BlueAngelHost discovered a lot of account creations from this domain and has blocked this entire domain. This was automatically detected by our program thanks to our flexible error handling.

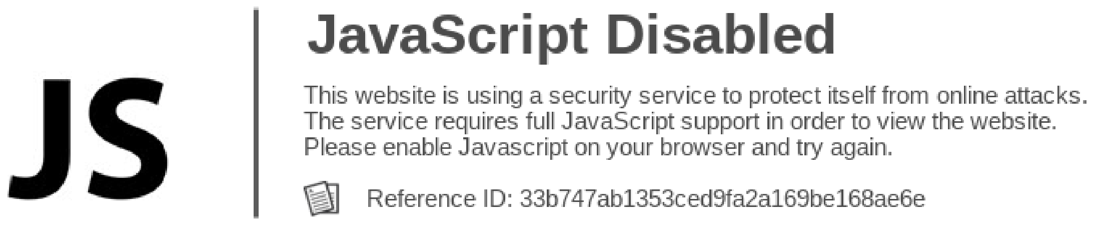
Additionally, after Cloudomate's implementation was finished, BlueAngelHost implemented a JavaScript and Cookie check which broke our implementation. Continuous integration tests failed because of this change, as a solution the JavaScript checking script has been reverse engineered and used in our implementatinon after which BlueAngelHost worked fine again.



```
Selected configuration:
Name            CPU             RAM             Storage         Bandwidth       Est.Price
Basic Plan      2               1.0             20.0            1.0             8.0
Purchase this option? (y/N) y
Purchasing BlueAngelHost instance: Basic Plan
Please correct the following errors before continuing:

We don't allow users with the email address provider you entered. Please try another email address.
```

Figure 5.1: BlueAngelHost error handling



JS

JavaScript Disabled

This website is using a security service to protect itself from online attacks.
The service requires full JavaScript support in order to view the website.
Please enable Javascript on your browser and try again.

Reference ID: 33b747ab1353ced9fa2a169be168ae6e

Figure 5.2: BlueAngelHost JavaScript blocking

6 Actual money trials

Now that Cloudomate has been tested for proper functioning and robustness, we have integrated Cloudomate in our main project, PlebNet, in order be used in a real world use case. To make the project work, there were a couple things that still had to be implemented. Scripts are created for installing Cloudomate and Tribler on clean VPS instances. System resources have been monitored to see what VPS configurations are optimal for Tribler exit nodes, and a DNA-like evolutionary algorithm has been implemented to maintain long-term robustness for the recreation of the botnet.

6.1. Creating revenue as exit node

The goal of this project is to create a cost-effective way to host exit nodes. There are a couple of factors to consider throughout this process: different providers offer different VPS configurations, some offer more RAM, some offer more CPU cores or more bandwidth, some are superior to other providers. Furthermore some providers don't tolerate the use of exit nodes, or even torrents at all. The websites of providers could change, rendering Cloudomate useless for the provider, and some providers have a very strict DMCA complaint handling policy. All these factors have to be taken into consideration when buying VPS instances for exit node usage. We have researched the resource usage of VPS instances running as exit nodes to see what resources are important when choosing which server to buy. Furthermore we use a genetic algorithm to cover providers that ban servers or do not allow the purchase of it for any reason, so that our botnet will be less likely to purchase instances from this provider in the future.

6.2. Server configuration picking

VPS providers offer different tiers of servers: RAM, CPU cores and storage space typically scale linearly with the configuration's price. Tribler has been installed on a server as exit node to get insight in what configurations fit our purpose. Monitorix[21], a system monitor, has been used to record system usage and to generate charts. The system usage has been recorded for one week.

6.2.1. Resource usage

These statistics teach us that 2 Gigabytes of RAM is plenty for running an exit node, 1 Gigabyte is sufficient, and servers with 512 MB of RAM might have some trouble. The tests are ran on the cheapest of all Linevast VPS options. The CPU usage and storage space weren't close to hitting any limits. The storage space usage has barely changed over the course of a week. This shows that the bandwidth and link speed are not only the main factors for choosing which VPS is optimal in terms of performance, but it shows that other specifications don't have to be considered when making the decision as they are present plentiful in all our implemented providers.

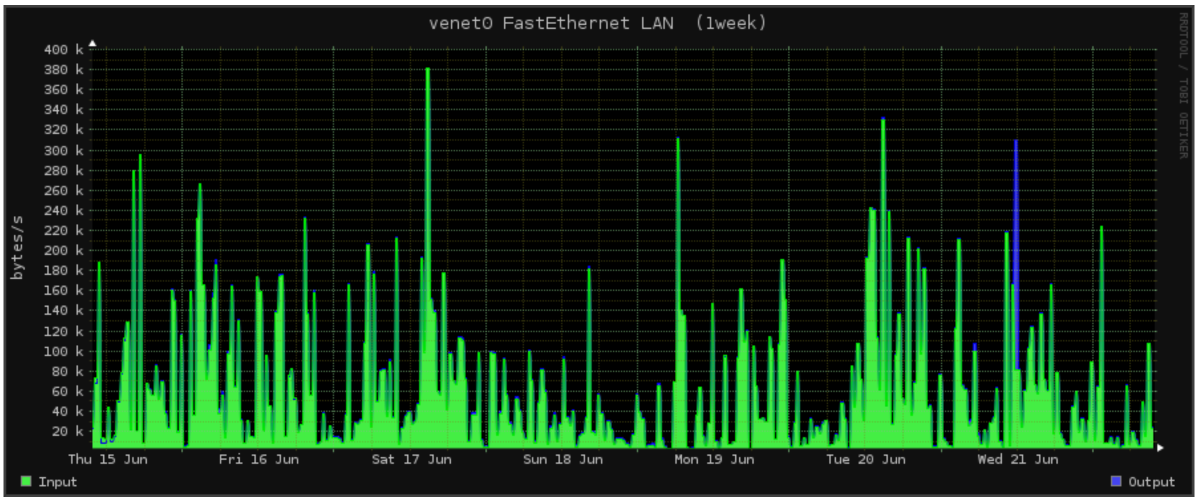


Figure 6.1: Network throughput

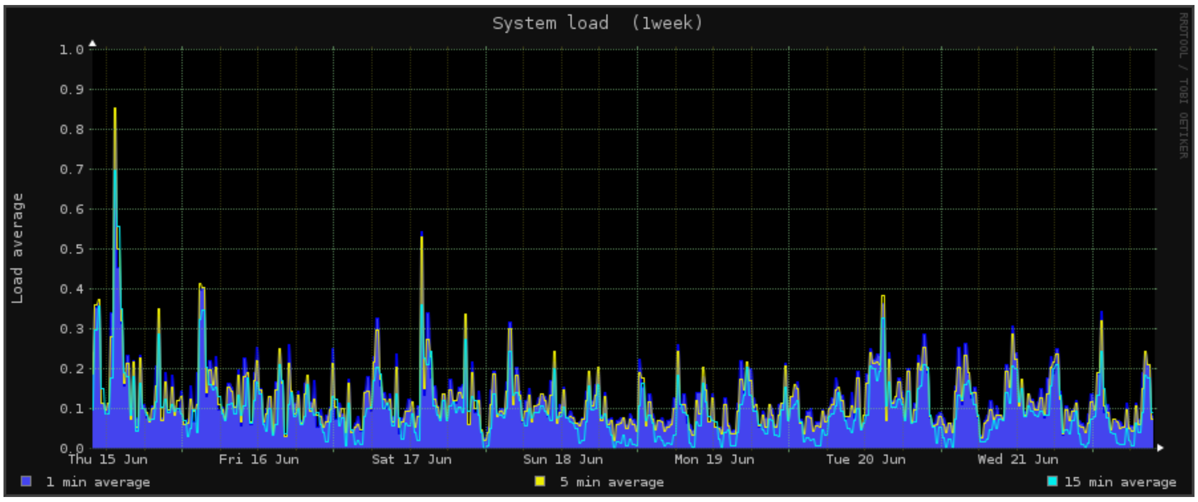


Figure 6.2: System load

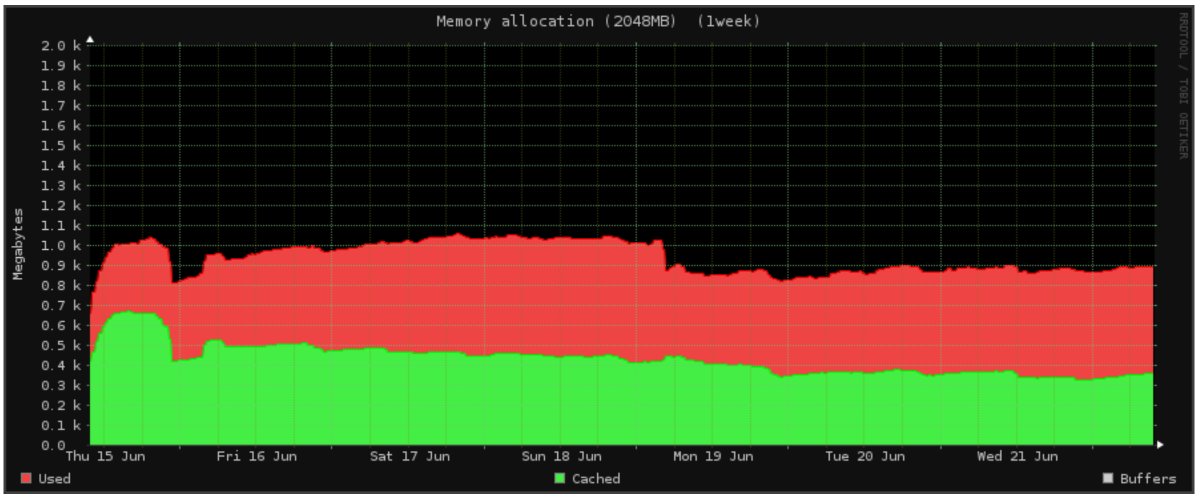


Figure 6.3: RAM usage

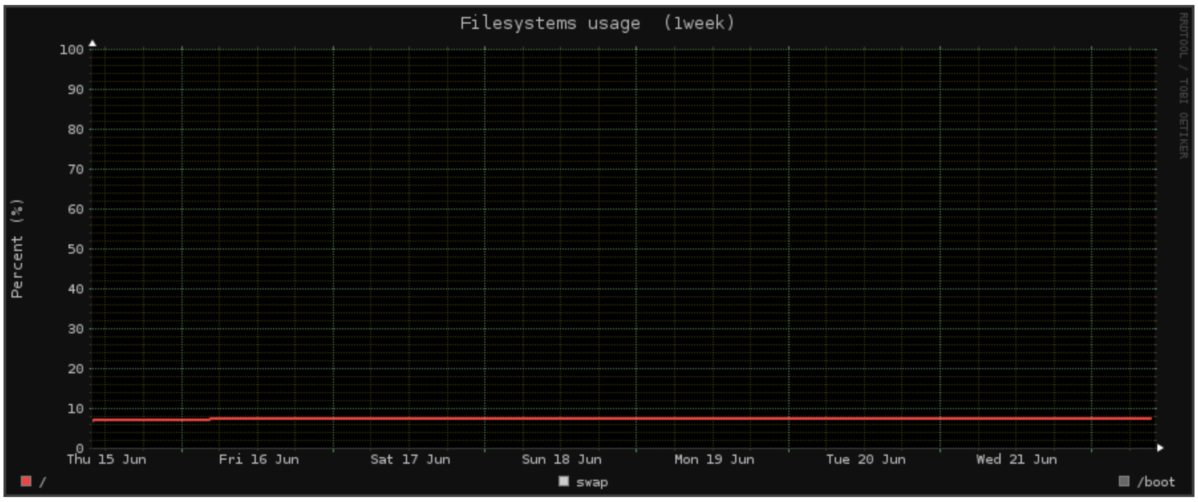
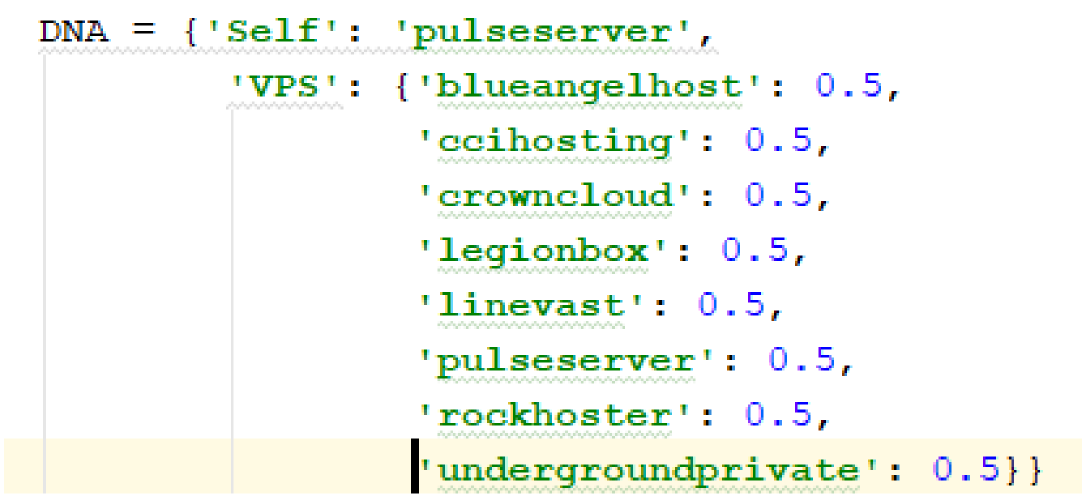


Figure 6.4: Storage

6.3. Darwinean algorithm

Learning from past experience is important during evolution. With evolution PlebNet will be able to adapt based on the performance of the VPS providers. For our genetic algorithm we used the basis of the previous group[26]. Tennet used a vector (DNA) with a starting value of 0.5 for all of their providers. They would add a mutate value to the normalized vector when they could successfully bought a VPS from one of their providers. We used their idea and made some improvements. We did not want to just mutate our DNA positively upon success, but also mutate negatively whenever failure occurs. A very realistic reason for failing to purchase from a provider is that they changed their purchase procedure. If this happens there is no way for PlebNet to correct this itself. By giving a consequence for failing to purchase a VPS the vector value of that provider will near 0 faster effectively taking it out of the rotation.



```
DNA = {'Self': 'pulseserver',
       'VPS': {'blueangelhost': 0.5,
               'ccihosting': 0.5,
               'crowncloud': 0.5,
               'legionbox': 0.5,
               'linevast': 0.5,
               'pulseserver': 0.5,
               'rockhoster': 0.5,
               'undergroundprivate': 0.5}}
```

Figure 6.5: DNA basis

6.3.1. Mutation Condition

Besides just adding a feature to negatively effect a provider, we changed our conditions for change a provider in a vector. The performance of a provider is very important to us. We want to reward providers for being able to give of the performance we need for PlebNet to survive. The previous group rewarded a provider if the purchase was successful. If a Linevast server buys a Legionbox server, Legionbox would be positively mutated. Instead of rewarding the server that was bought, we reward the server that is buying the new server (Linevast). Because the Linevast server can buy a new server instance, it has proven that it can earn enough money to keep PlebNet alive. For this reason we have ‘Self’ attribute in our DNA. This is to keep track of which server provider PlebNet is currently using.

6.3.2. Unused Potential

Some of our ideas for evolving were unfortunately not implemented during this project due to time constraints. Bandwidth is an important resource for PlebNet. This led us to two ideas on how to incorporate bandwidth into our DNA. The first was to calculate the amount of data a server could use during a month and divide it by the price of the server. This will result in a score that represents the data per Bitcoin. Using this in conjuncture with the DNA basis will reward providers with a better bandwidth.
The second idea is to average the data that has passed through the provider with a learning rate. Let’s say Pulseservers has an average of 200 TB passing through in a month. A Pulseservers instance just finished its life cycle and used 250 TB. We would then create a new average by giving the 250 TB a learning rate of 20%. The 200 TB will count for 80% of the new average. This is because there are more past experiences, if the new data is just a one time performance the average is not defined by an outlier.

6.4. Construction of PlebNet

Where Cloudomate, as the largest building block of PlebNet, is responsible for purchasing servers, PlebNet itself is responsible for earning reputation, selling reputation for Bitcoin, letting Cloudomate purchase a new server when ready, and installing PlebNet on the new server. Upon creation of a new server, the DNA and wallet of the parent server is copied, and a new identity, in the form of a cloudomate configuration file, is generated. To manage the different events in a simple way, a checking script is called every minute.

6.4.1. Selling reputation

Firstly, the script verifies if the Tribler exit node is still running and mining reputation. If this is not the case, Tribler is started in exit node mode. Then, the amount of available reputation is obtained. If this amount is more than zero, a selling offer is made on the Tribler market, selling all available reputation for the amount of Bitcoin required to purchase two chosen servers. A new offer of the price of a new selection of two providers is published every day. The providers aren't chosen completely randomly. The DNA is used to choose providers that have proven to be reliable in the past. Additionally a price/bandwidth score is calculated for each option of the DNA chosen providers for optimal value.

6.4.2. Avoiding reputation loss

Next, the remaining lifetime of the server is checked. As servers have a lifespan of 30 days, it is necessary to sell all available reputation before the end of the lifespan, for else the reputation will be lost when the server is shut down. In the last 5 remaining days, not the price of two, but of a single server is asked on the market in exchange for all available reputation. While reputation is lost when a server is shut down, the wallet is shared amongst servers and thus survives shut downs of single servers.

6.4.3. Purchasing servers

Now the available amount of Bitcoin that is present in the wallet is checked. A rise in Bitcoin indicates that a trade has been successfully completed. If possible the cheapest of the chosen configurations is purchased through Cloudomate. The success of a purchase reinforces the DNA for the current provider, similarly an error in the purchasing process is reflected in a negative reinforcement of the provider. The purchased provider will be marked as "bought" in PlebNet's configuration file.

6.4.4. Setting up children

Lastly, a status check is done for all providers that have been bought, but haven't been initialized yet. The time it takes for the Bitcoin transaction to complete and for the provider to allow for server access can vary from a minute to multiple hours. When a status changes from pending to active has been constitated through a "cloudomate status" call, PlebNet knows that the new server is ready to be initialized. First, the ip address of this server is obtained, then the root password is changed to the desired password as some providers set a random root password on new servers. Then the create-child bash script is ran which in turn copies the DNA and wallet to the new child, after which PlebNet is cloned and the install script is ran to fully initialize this new agent. Finally an email is sent to us with server information and DNA parameters.

6.5. Real world experiments

While the implementation of the initial version of PlebNet went smoothly, a lot of corner cases and difficulties have been encountered.

6.5.1. Automating installation

There are a number of tools that can simplify automating the setup of new servers. Initially we looked into Ansible, an automation engine for application management and deployment. There already existed a couple of configurations to set up Tribler, but because Ansible mainly focuses on managing multiple servers as human, while our project focuses on autonomous servers which are installed once and never controlled by humans at all, and because installing Tribler is only a small part of our project, we decided it would be beneficial to look into alternatives.

Next we looked into using Docker. Docker provides containers which form an abstraction on operating system level. The advantage of using Docker would be to not have to worry about different OSes as Docker can provide e.g. Ubuntu's system and package management on all our systems. A Docker script has been implemented and tested, however the script could not be used on many servers. Providers generally offer two ways of virtualization: OpenVZ and KVM. While the buyer has full control over kernel updates on KVM virtualized servers, this can only be done by the server provider on servers virtualized by OpenVZ. The majority of cheap configurations use OpenVZ and every OpenVZ server we encountered uses an ancient Linux version 2.6.x. Linux 2.6.x is unofficially long term supported, but released in 2006, for comparison: the current oldest supported long term Linux version is 3.2.89 [10]. The usage of an old kernel version is a problem because Docker doesn't support any Linux kernel before version 3.10.0 [8]. For this reason we decided to use bash scripts to handle our Tribler and PlebNet installations. Since all VPS providers support Ubuntu, we chose to go along with that.

7 Testing

To test our project, we have created a number of unit tests for Cloudomate, we used continuous integration to make sure all tests pass whilst making changes to our project.

7.1. Jenkins

Jenkins[9] is the backend we have used for continuous integration. The Tribler team already uses this to continuously test many other projects, and we were able to use this backend to test Cloudomate. Two different configurations have been used to test Cloudomate. The main configuration runs all unit tests whenever the master branch of the project is changed, this makes sure that our project builds and functions as desired, and indicates failure thereof through a badge on the Github and Pip page of the project. Secondly, tests are ran for each change to each pull request submitted to the project. This shows, on the Github page of the pull request, whether tested functionality is preserved when the changes of the pull request are made.

7.1.1. Build issues

Throughout the project we have encountered a number of failing tests by Jenkins due to different causes. Two tests of a total of 50 failed on master. One of the failing tests on master indicated the lack of an installed dependency. Python dependencies of the project are installed by the Jenkins script, but python-lxml, a dependency that has to be installed not by Python's package manager but by the operating system was missing.

7.1.2. Third party changes

Some failing tests indicated a new problem related to third parties, the second failing test on master indicated BlueAngelHost implementing a JavaScript and cookies check to counter automated use of their website. Some tests on pull requests failed due to connection issues with third parties: RockHoster, Linevast and the used Bitcoin exchange rate API have been down at times. Downtime of providers cannot be fixed by our project, though we have at a certain point replaced the old Bitcoin exchange rate API with a more stable one. Overall continuous integration has helped us discover a number of problems, it has been an effective approach.

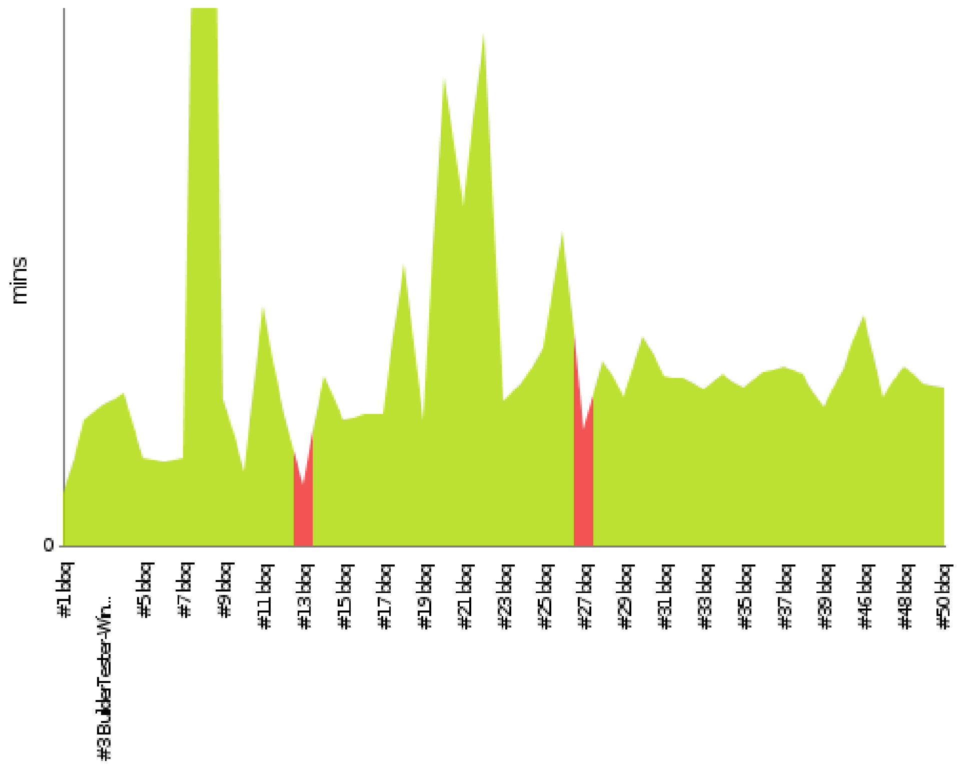


Figure 7.1: Jenkins master branch status and duration

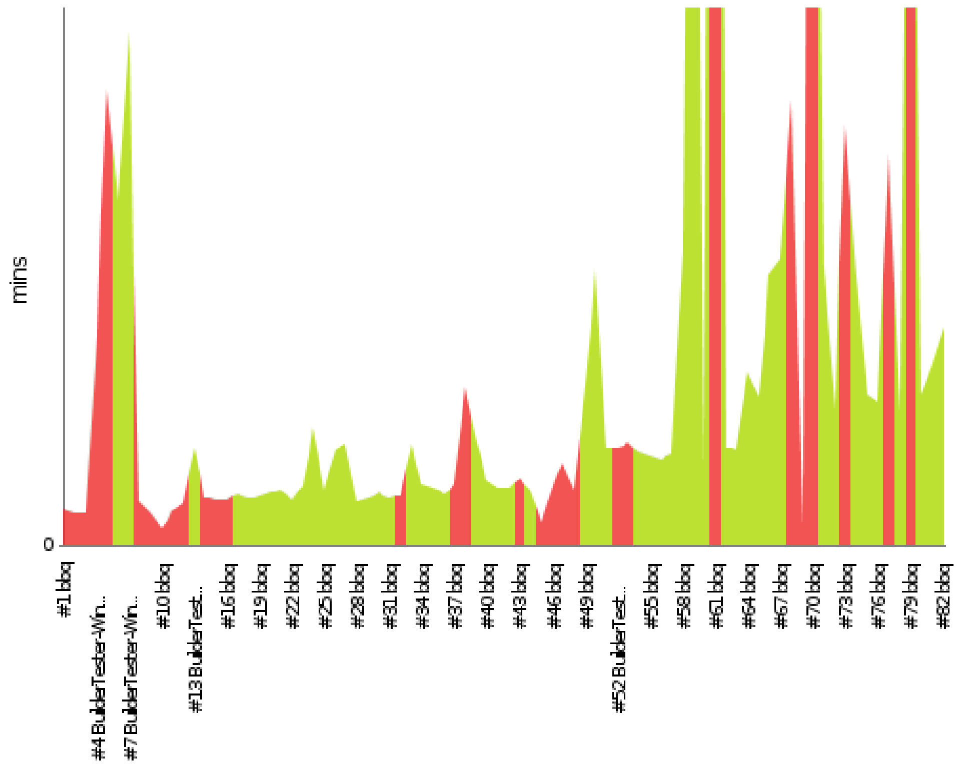


Figure 7.2: Jenkins PR status and duration

7.2. Testing difficulties

Next to third party dependencies and downtime, a large part of our project has been very difficult to test. Money spending code, in Cloudomate as well as in PlebNet, is not convenient to test through continuous integration for the simple reason that spending $8 on 8 providers every time a change is made to the project gets expensive. A lot of our code involves setting up a server from scratch, we have looked into testing this through unit tests, but considered the implementation too time consuming for what it's worth as VM technology would have to be implemented, a clean server would have to be created every time our project would be tested, and this would be a time consuming operation.

7.3. SIG

The first submission of our code to SIG[11], for code quality assesment, has been in week 5, the feedback can be found in appendix B. We have received an above-average score of 3/5. The only complaint in the feedback was the amount of duplication in our code. As all 6 implemented hosters use SolusVM as registration backend, the registration code of our code was similar in all implementations. We knew about this and had made the tradeoff between placing this overlapping registration code in the Hoster parent class, or in each individual subclass. We decided to use the similar code separately in subclasses because we considered not each Hoster is a SolusVM hoster, so SolusVM specific overlapping code wouldn't be covered by the responsibility of Hoster. It is true that this does result in a considerable amount of code duplication. Since the duplicated code is not completely equal but slightly different as some hosters require additional fields to be submitted one might question whether the code is "by chance" similar or similar with reason. In any case, after some further research we found the code to be even more similar than we had suspected, this resulted in implementing a class between Hoster and each individual hoster: SolusVMHoster. This class covers the overlapping part of registration code of all our current providers and offers a nice solution to the code duplication.

8 Conclusion

All in all the project has finished and provides all elements required for an autonomous anonimity providing entity to survive and to replicate itself, moving between multiple VPS providers based on their reliability. Additionally Cloudomate has been released to the public to provide automation of server registration for anyone to use. Compared to the previous group, robuster software has been created, the project is ready for real world usage, though extensive testing in real world scenarios is still left to be done.

8.1. Dicussion

We have implemented all main steps of the replication process. The project allows for reputation mining, trading reputation for Bitcoin, registering new servers with Bitcoin and replicating itself to untouched servers. The project's core parts are implemented, but real world usage on this scale brings a lot of corner cases and uncertainties with it. Extensive testing, ironing out potential bugs and some polish is still required to make the entity run smoothly for an extended lifetime. This is a time consuming task, though a solid basis has been created.

Beyond core functionality this project features an extended evolution algorithm for reinforcement of reliability in server provider choice. A command line interface based server registration program with the ability to check on status, server info, setting a root password and login in through SSH has been created. Additionally the life expectancy is used along with available server configurations the genetic algorithm to create a formula for prolonged existence of the agent network.

8.2. Recommendations

The next step in this project is implementing additional error handling and doing extensive testing. When testing with an amount of money on real servers, a couple of bugs might still occur. These bugs need to be fixed, and preferably automatically reported. A system to create an issue on the Github page of the project could be made for convenient long term debugging. There are many factors that come into play for an autonomous self-replicating entity. The test of time is required to ultimately confirm succesful operation of this project.

Bibliography

[1] Coinbase. URL https://www.coinbase.com/.

[2] Selenium. URL http://www.seleniumhq.org/.

[3] Tribler, Accessed: 2017-05-18. URL https://www.tribler.org/about.html.

[4] Dispersy, Accessed: 2017-06-23. URL https://github.com/Tribler/dispersy.

[5] anonymity tribler, Accessed: 2017-06-23. URL https://www.tribler.org/anonymity.html.

[6] Github tribler, Accessed: 2017-06-23. URL https://github.com/Tribler/tribler/wiki.

[7] Ccihosting, Accessed: 2017-06-26. URL http://www.ccihosting.com/.

[8] Docker, Accessed: 2017-06-26. URL https://www.docker.com/.

[9] Jenkins, Accessed: 2017-06-26. URL https://jenkins.io/.

[10] Kernel, Accessed: 2017-06-26. URL https://www.kernel.org/.

[11] Sig, Accessed: 2017-06-26. URL https://www.sig.eu/.

[12] Beautifulsoup, Accessed: 2017-06-26. URL https://www.crummy.com/software/BeautifulSoup/.

[13] Bitpay, Accessed: 2017-06-26. URL https://bitpay.com.

[14] Blueangelhost, Accessed: 2017-06-26. URL https://www.blueangelhost.com/.

[15] Cloudomate, Accessed: 2017-06-26. URL https://pypi.python.org/pypi/cloudomate.

[16] Crowncloud, Accessed: 2017-06-26. URL https://crowncloud.net/.

[17] Electrum, Accessed: 2017-06-26. URL https://electrum.org/#home.

[18] Legionbox, Accessed: 2017-06-26. URL https://legionbox.com/.

[19] Linevast, Accessed: 2017-06-26. URL https://linevast.de/.

[20] Mechanize, Accessed: 2017-06-26. URL http://wwwsearch.sourceforge.net/mechanize/.

[21] Monitorix, Accessed: 2017-06-26. URL http://www.monitorix.org/.

[22] Ramnode, Accessed: 2017-06-26. URL https://www.ramnode.com/.

[23] Rockhoster, Accessed: 2017-06-26. URL https://rockhoster.com/.

[24] Scrapy, Accessed: 2017-06-26. URL https://scrapy.org/.

[25] Solusvm, Accessed: 2017-06-26. URL http://solusvm.com/.

[26] N C Bakker, R v.d. Berg, and S A Boodt. Autonomous self-replicating code. *TU Delft Repositories*, June 2016. URL http://repository.tudelft.nl/islandora/object/uuid%3Aa1b443c7-8b37-4263-ae96-d38bc8b8f397?collection=education.

[27] Nicolas Christin. Peer-to-peer networks: Interdisciplinary challenges for interconnected systems. *Information Assurance and Security Ethics in Complex Systems: Interdisciplinary Perspectives: Interdisciplinary Perspectives*, page 81, 2010.

[28] Robert F Easley. Ethical issues in the music industry response to innovation and piracy. *Journal of Business Ethics*, 62(2):163–168, 2005.

A
Infosheet

Autonomous Self-replicating Code
Tribler
Presentation Date: July 3rd 2017
The core challenge of our project was to create an Internet-deployed system which can earn money, replicate itself, and which has no human control. Tribler uses exit nodes to create an anonymous network. We have made a system that can buy servers and turn them into exit nodes. During our research we analyzed work of the previous group. We were not able to use their code due to lack of robustness in their system design. During our process we switched between web scraping libraries, because the API we used did not provide us with the adequate resources we needed. We also found out that the testing of buying VPS is a challenge, because the providers banned us when we accessed their service in frequent successions. In the end our project delivered a usable product. Due to time constraints features were left out. The product can still use some polish to make it more robust.
Project team member:
Member 1:
Name: René van den Berg
Interests: Stock Market
Role: Docker, test development, PlebNet structuring
Member 2:
Name: Jaap Heijligers
Interests: Decentralized Internet, Torrents
Role: VPS provider, server testing, test development
Member 3:
Name: Mitchell Hoppenbrouwer
Interests: Torrents, Bitcoin
Role: Genetic Algorithm, VPS provider, Wallet, Graphic designer
All members on the team worked on the report.
Client:
Name: Martijn de Vos
Affiliation: Employee of Tribler.
Supervisor:
Name: Johan Pouwelse
Affiliation: Professor at Delft University, founder of Tribler.
Contact Person:
Mitchell Hoppenbrouwer: mitchell.h.lang@gmail.com
The final report for this project can be found at: http://repository.tudelft.nl

B
SIG feedack 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



[Analyse]

De code van het systeem scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de
    code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere
    score voor Duplication.

Voor Duplication wordt er gekeken naar het percentage van de code welke redundant is, oftewel
     de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen
    worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage
    redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere
    plaatsen moet gebeuren.

In dit systeem is er duplicatie te vinden tussen de verschillende subclasses van Hoster. Dit
    is gezien de architectuur vreemd: je hebt juist een parent class voor het gedeelde gedrag
    . Het is daarom belangrijk om te kijken welke code algemeen is, en welke code specifiek
    voor een bepaalde aanbieder.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de
    test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

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

C
Project Description

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 the not focus of your project. You will create software that is beyond human control and includes breathtaking features such as earning money (Bitcoin) and self-replicating code (software buys a server+spawn clone).

Earning money consists of helping others become anonymous using the Tor-like protocols developed at TUDelft and our own cybercurrency designed for this purpose, called Trustchain coins. You Python software is able to accomplish the following:

- Earn income in a form of cybercurrency (existing code, see below)
- Sell this cybercurrency on a market for Bitcoins (another ongoing project)
- Buy a server using Bitcoins fully automatically
- Login to this Linux server and install itself from the Github repository

The software also should be able to have a simplistic form of genetic evolution. Key parameters will be inherited to offspring 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 if you offer Tor exit node services for income. Bitcoins owned by TUDelft will be used to bootstrap your research.