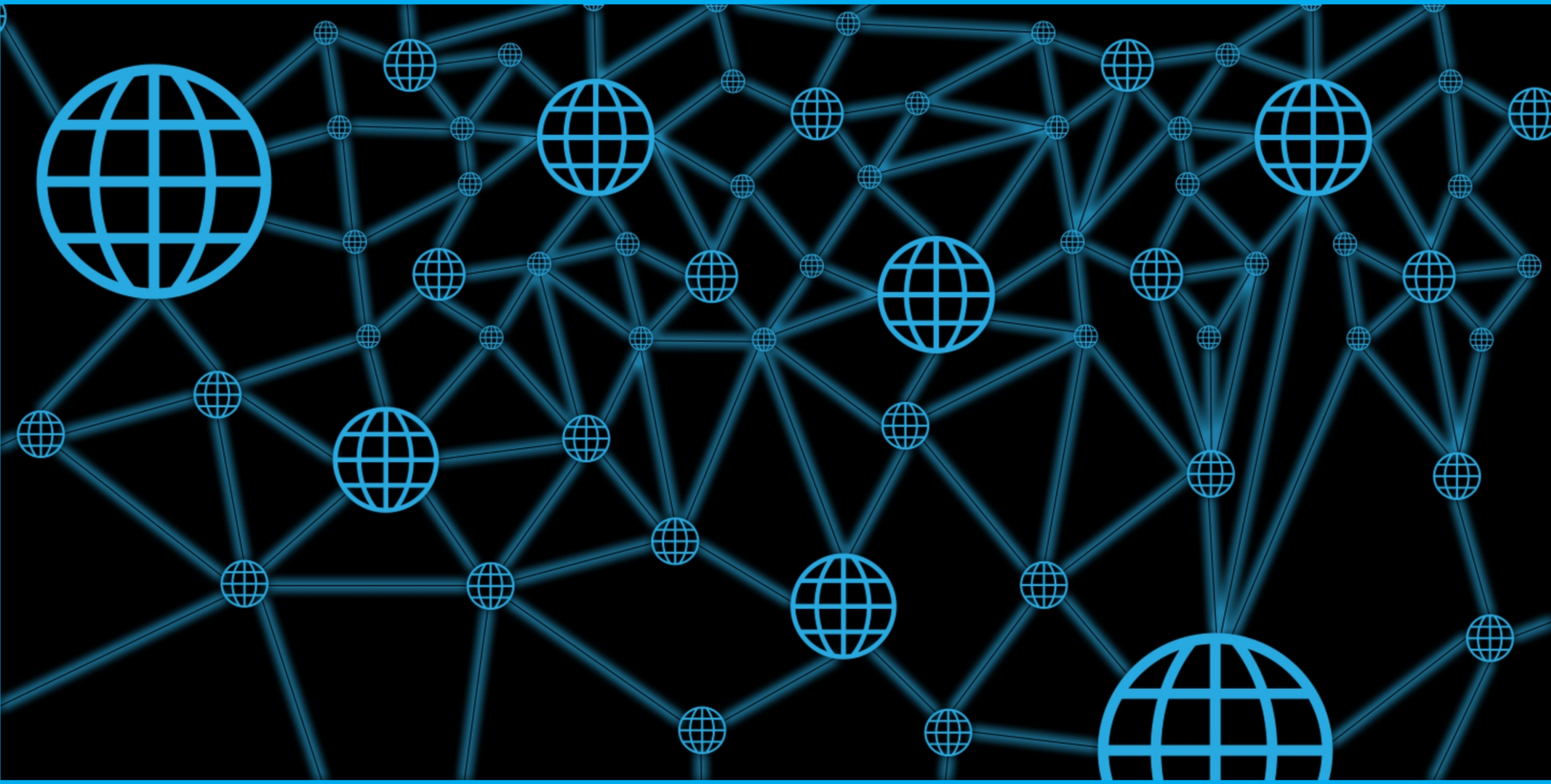
PlebNet

Working class botnet

R. van den Berg
J. Heijligers
M. Hoppenbrouwer



PlebNet
Working class botnet

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 Tuesday July 4, 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.
(other explanations)
In this document we describe our methods, designs and solutions for the problem posed by the Tribler Team.
(...)
(word of thanks)

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

Contents

1 Introduction

For this Bachelor End Project, BEP, we were asked to create an Internet-deployed system which earns money, replicate itself and has no human control. To earn money we help others to become anomymous using the Tor-like protocols developed at TU Delft, Tribler. This is done by earning reputation, which is represented by Multichain coins. We will sell our reputation for Bitcoin.
Our system will earn income in the form of cybercurrency, sell this cybercurrency for Bitcoin, use the Bitcoin to buy a server automatically and install itself on the server. The system will also have a simplistic form of genetic evolution, which will be used to determine which server it will buy.

1.1. Tribler

Tribler is a project of Delft University of Technology. It is a Peer-to-Peer file sharing system. Over the course of nine years over one million users have installed it. Tribler continuously improves on the BitTorrent protocol from 2001. Tribler has expanded it by adding more features such as, streaming from magnet links, keyword search for content, channels and reputation-management. The entire system is implemented in a distributed manner, which means it does not rely on a centralized component. [1]
Tribler uses exit nodes as an entry into the anonymous network. These exit nodes are important to support the system. Currently Tribler has a number of exit nodes located at LeaseWeb. Since all of the exit nodes are in one location it has become a centralized network. The goal of this project is to create a decentralized exit node network.

1.2. Market

One of the issues with torrent networks is leachers. Leachers are users who download at lot, but don't upload, seed, so other user can download the data. Tribler wants to limit the bandwidth the users can use if they download more than they upload. 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.
Reputation is ascertained from the data a user downloaded and uploaded. A higher upload than download will result in a positive reputation, while a lower upload than download will result in a negative reputation. Our system will earn reputation by uploading a lot of data as an exit node. Since a exit node is an entry into the Tribler network it will upload more than it will download. The reputation that is earn will be sold on the market for Bitcoin.
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. Previous work

Last year another BEP group, Tennet [2], 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.
The bot network that Tennet build 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 with 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.
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 and learn from the previous group.

1.4. Ethical Considerations

A peer-to-peer network 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 does have ethical challenges. Historically a significant portion of the data on peer-to-peer networks consists of copyrighted content.[3]. Even though it is illegal to 'pirate' copyrighted content the widespread acceptance questions whether it is unethical. The public does not view copyright infringement as a serious crime.[4]

2 Product Design

2.1. Product Requirements
This chapter will discuss 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.

2.2. Product Requirement
This section will discuss 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.

2.2.1. 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 an evolution 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 IP address
3. The agent should have an intelligent trading system
4. The agent should have different configurations
5. The agent should have 6 different VPS options

Could Have

1. The agent could have 15 different VPS options
2. The agent could publish information online
3. The agent could share knowledge with its children
4. The agent could create an issue on Github

5. The agent could update itself mid lifespan
6. The agent could predict its own life expectancy

Won't Have

1. The agent won't extend the lease on the current server
2. The agent won't update the VPS form automatically
3. The agent won't look for new VPS providers
4. The agent won't have a bitcoin miner
5. The agent won't use other currencies
6. The agent won't pair with another agent

2.2.2. 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 multichain coins. Multichain 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 multichain coins, it has to sell this for Bitcoin on the market. The Bitcoins the agent earns can then be used to sustain itself as well creating children.

Buying servers In order to have this be 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.

Evolution algortihm When the agent moves to a new server or 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. One of the important aspects of evolution is reproducing. The agent should be able to reproduce. This 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.

2.2.3. Should Have

Error notifications The agent should send error notifications to the client. If the agent runs in to 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 IP address. When the agent has bought a new server, it should send the IP address to the client. The client can keep track of how many agents are running. If this number declines the client can choose to intervene.

Trading System The agent should have an intelligent 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.

Configurations The agent should have different configurations. The agent should decide what the best Tribler configuration is. It should decide whether or not to be an end node and whether to turn the family filter off.

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.

2.2.4. 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.

Publish information The agent should publish information on its status. For example its life expectancy, current reputation, cpu usage and other such information. This way the client can see how the agents are doing.

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

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.

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.

2.2.5. 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 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.

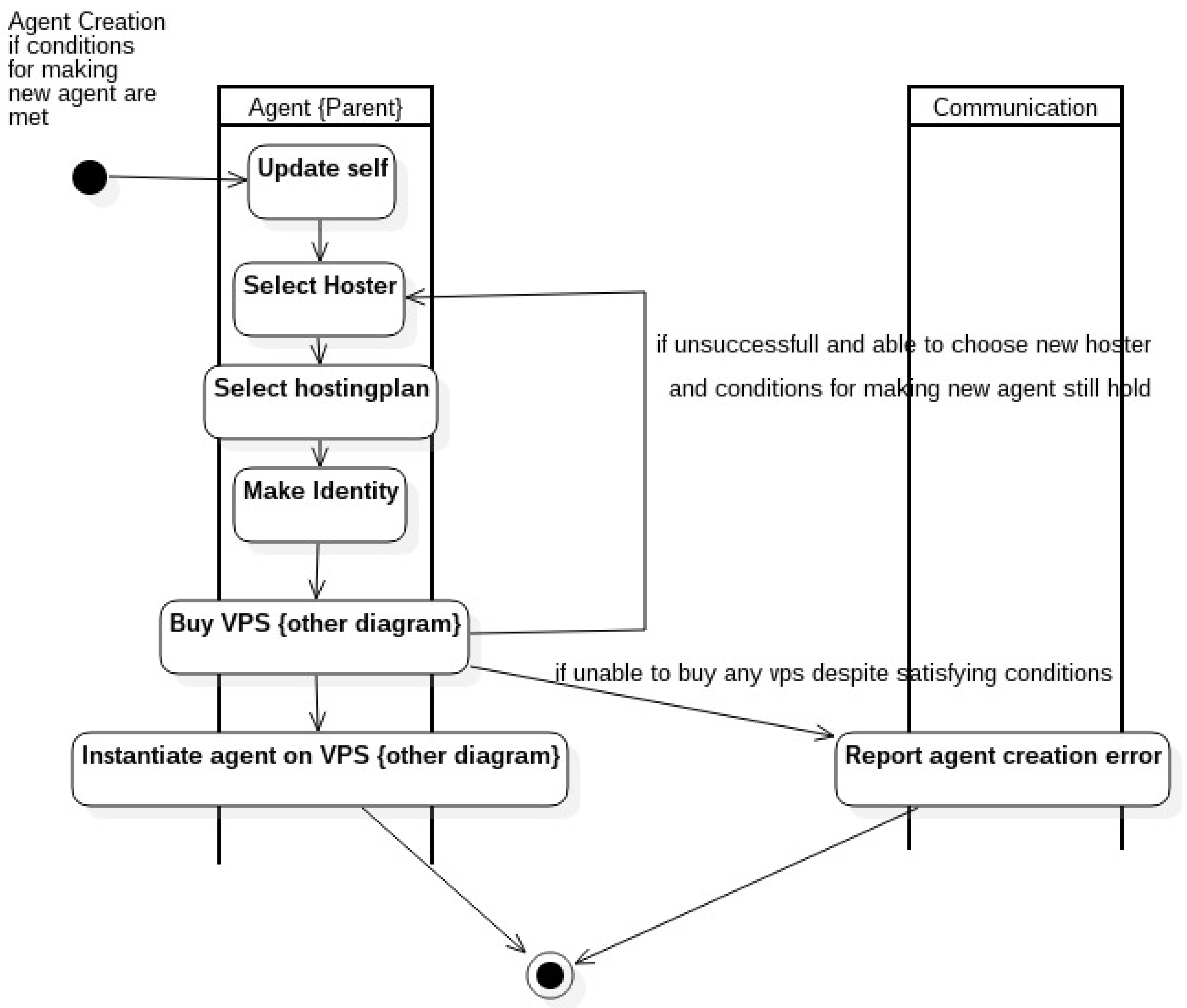
Find new VPS providers The agent won't look for new VPS providers. This would require something akin to an artificial intelligence and it is not possible to implement during this project.

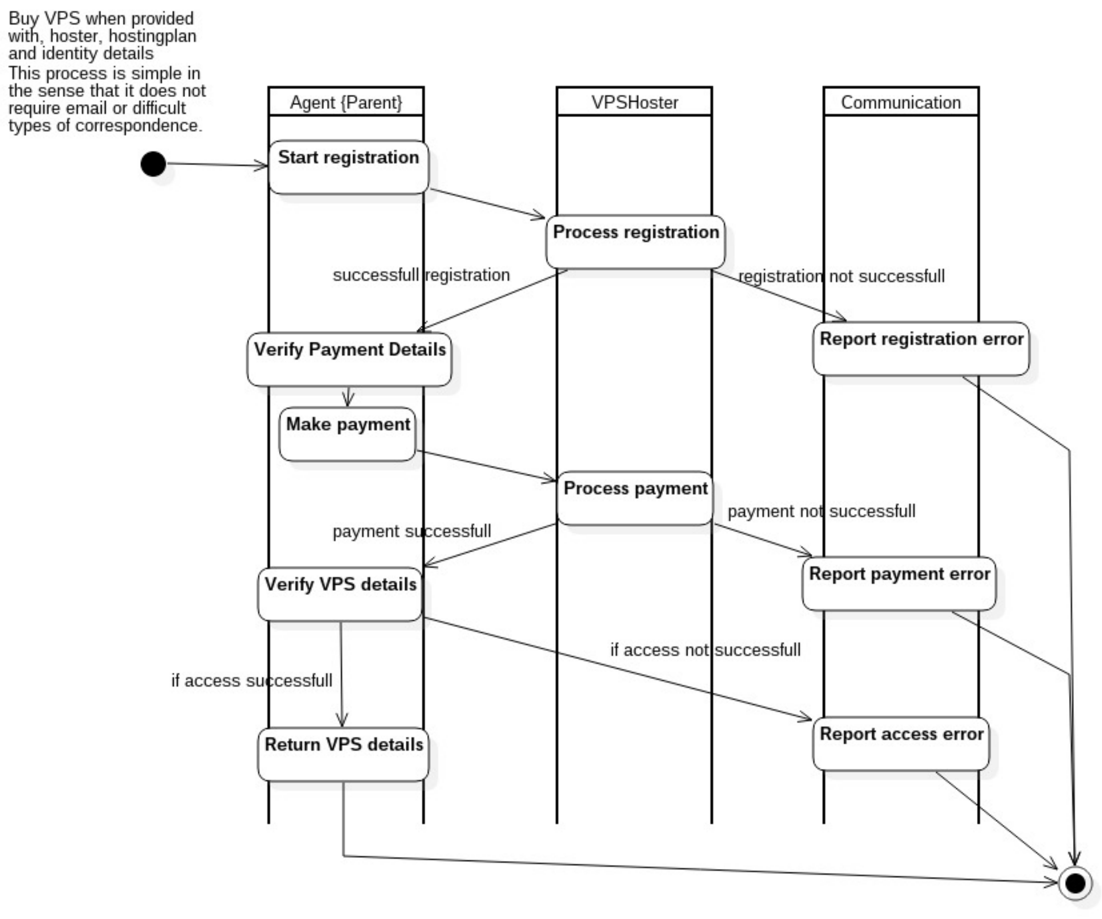
Bitcoin miner The agent won't mine for Bitcoins. The VPS providers do not have the hardware to support Bitcoin mining. It just would not be feasible to mine for them.

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

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

2.3. Agent Functions


Agent Creation if conditions for making new agent are met
Agent {Parent}
Communication
Update self
Select Hoster
Select hostingplan
Make Identity
Buy VPS {other diagram}
if unsuccessfull and able to choose new hoster
and conditions for making new agent still hold
if unable to buy any vps despite satisfying conditions
Instantiate agent on VPS {other diagram}
Report agent creation error



2.4. Functional Modules

2.5. Risks

3 Initial experiment

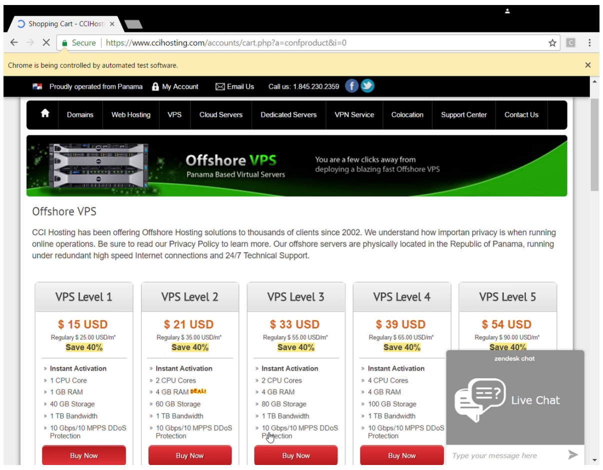
Last year there was a BEP group that worked on this project: Tennet[2]. We used this initial experiment to see what they used and what we think is useful for us to use.

3.1. Selenium

Selenium is a suite of tools that can be used to automate web processes. With Selenium you can go to website and extract information as well as fill in forms. During the process of buying VPS servers online, a number of forms will be needed to filled in. These forms include account creation and server settings. With Selenium is it possible to perform all of these tasks.

3.1.1. Example

To show how Selenium works we will show an example. In this example Selenium will try to buy a server from CCIHosting.
First Selenium opens a Google Chrome browser and goes to the URL for CCIHosting VPS plan selection page (figure 3.1). Here it chooses the cheapest option.


Shopping Cart - CCIHosti

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

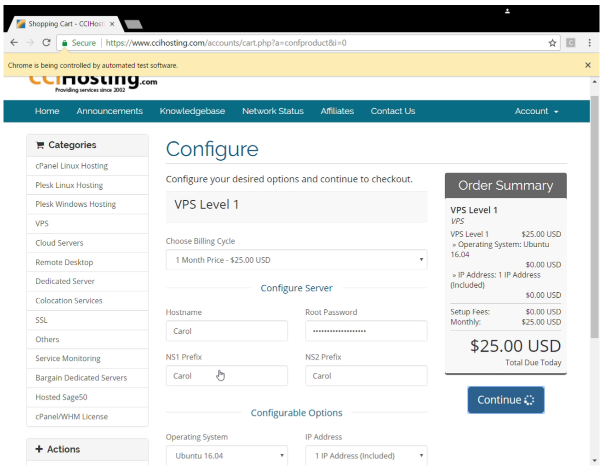
Offshore VPS

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	

Figure 3.1: VPS plan selection

After it has selected the VPS plan, Selenium goes to the configure screen (figure 3.2). Here it has to fill in a hostname, rootpassword, ns1 prefix and ns2 prefix. Then the server is added to the cart.



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

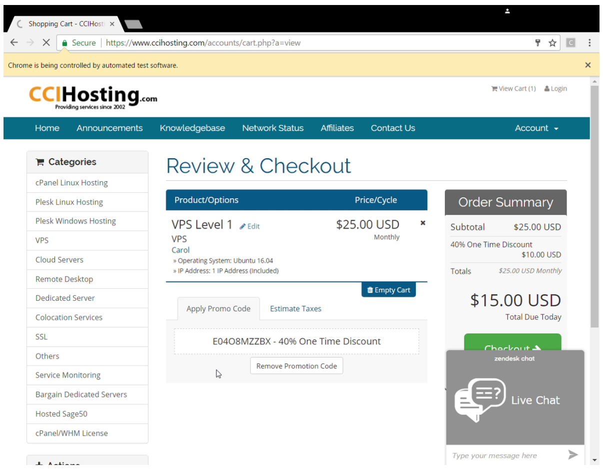
Operating System: Ubuntu 16.04
IP Address: 1 IP Address (Included)

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

Figure 3.2: configure screen

At the cart screen (figure 3.3) Selenium uses the checkout button to go the next page.



Review & Checkout

Product/Options	Price/Cycle
VPS Level 1 VPS Carol » Operating System: Ubuntu 16.04 » IP Address: 1 IP Address (Included)	$25.00 USD Monthly

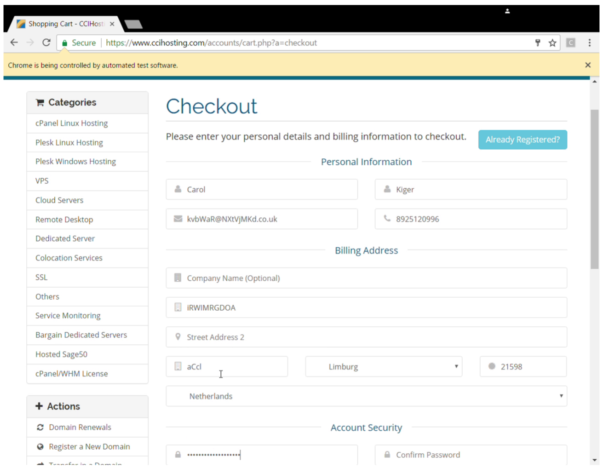
E04O8MZZBX - 40% One Time Discount

Order Summary

Subtotal $25.00 USD
40% One Time Discount $10.00 USD
Totals $25.00 USD Monthly
$15.00 USD
Total Due Today

Figure 3.3: cart

In the checkout screen (figure 3.4) Selenium fills in the personal information, such as first name, last name, email address, phone number and address.



Checkout

Please enter your personal details and billing information to checkout.

Personal Information

Carol
Kiger
kvbWaR@NXtVjMKd.co.uk
8925120996

Billing Address

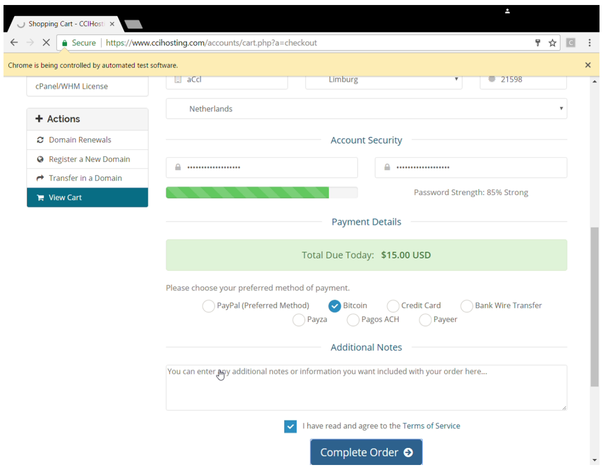
Company Name (Optional)
IRWIMRGDOA
Street Address 2
aCcI
Limburg
21598
Netherlands

Account Security

••••••••••••••••••••
Confirm Password

Figure 3.4: checkout, personal information

For the payment option (figure 3.5) Selenium chooses bitcoin and then agrees to the terms of service.


Shopping Cart - CCIHosti
Secure | https://www.ccihosting.com/accounts/cart.php?a=checkout
Chrome is being controlled by automated test software.

cPanel/WHM License

aCcl | Limburg | 21598

Netherlands

Actions
- Domain Renewals
- Register a New Domain
- Transfer in a Domain
- View Cart

Account Security

•••••••••••••••••• | ••••••••••••••••••

Password Strength: 85% Strong

Payment Details

Total Due Today: $15.00 USD

Please choose your preferred method of payment.
PayPal (Preferred Method) [✓ Bitcoin] Credit Card Bank Wire Transfer Payza Pagos ACH Payeer

Additional Notes

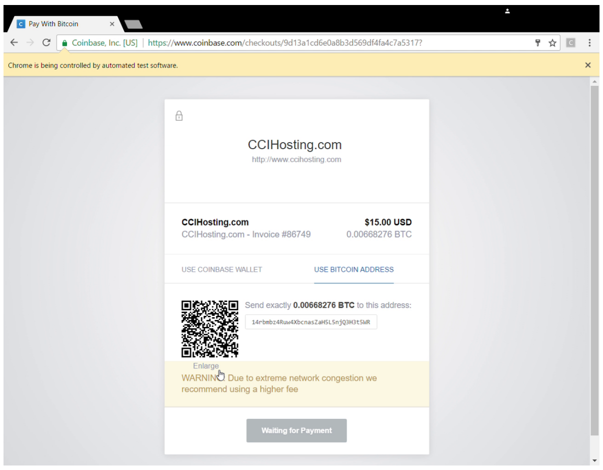
You can enter any additional notes or information you want included with your order here...

[✓] I have read and agree to the Terms of Service

Complete Order

Figure 3.5: checkout, payment option

After Selenium gets send to the coinbase site (figure 3.6) where the invoice of the server is, it reads the bitcoin amount it needs to pay and the bitcoin address


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

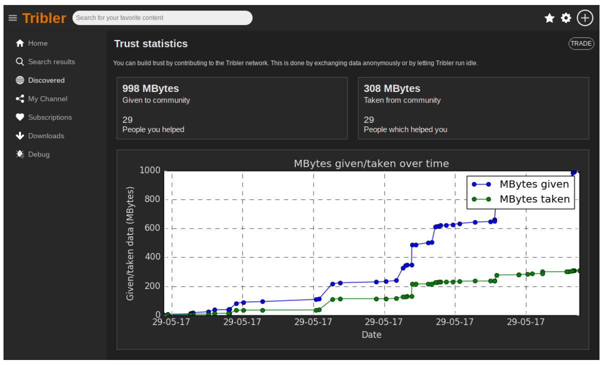
Figure 3.6: coinbase

3.1.2. Analysis

With Selenium it is possible to buy VPS servers autonomously. Selenium can go to a designated website. From there it is able to go through the site and fill in forms that are needed 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 server. However a VPS server 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. Tribler market

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 Tribler market still in development we used dummy wallets to simulate transaction reputation mining using anonymous downloading multichain to bitcoin



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

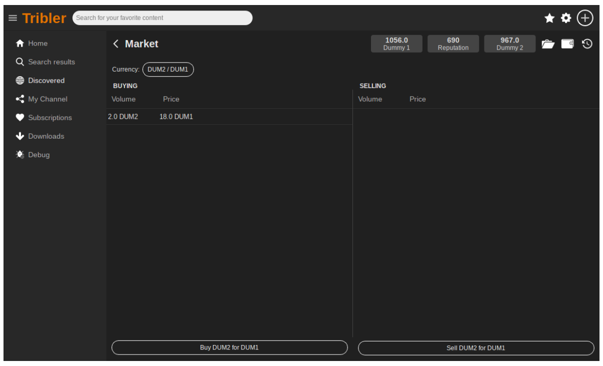
29
People you helped

308 MBytes
Taken from community

29
People which helped you

Figure 3.7: 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 3.7 the trust statistics on Tribler are shown. The statistics are separated into two blocks, the given and the taken block. 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 people helped or helping has no influence on your reputation. The only data that determines your is the amount uploaded and downloaded. Reputation is equal to the uploaded data minus the downloaded data. Using the statistics form figure 3.7 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. In the top right corner of the page is a trade button, which brings you to the Tribler market.



1056.0 Dummy 1 | 690 Reputation | 967.0 Dummy 2

Currency: DUM2 / DUM1

BUYING

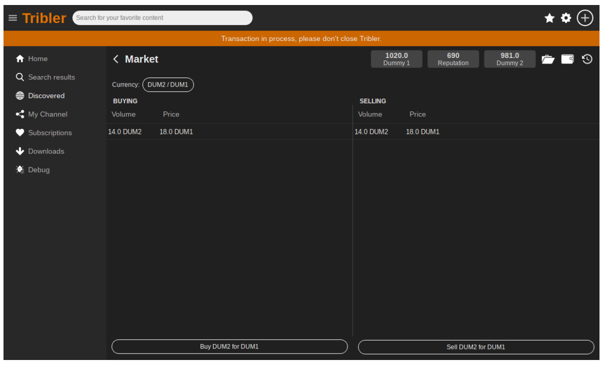
Volume	Price
2.0 DUM2	18.0 DUM1

SELLING

Volume	Price

Figure 3.8: the Tribler market

In the Tribler market (figure 3.8) reputation can be sold for Bitcoin. During our experiment we were able to use dummy wallets with 2 dummy currencies, dummy 1 and dummy 2. The market shows your wallets as well as your reputation. You can change your currency, during our experiment this meant changing between dummy 1, dummy 2 and multichaincoins (reputation). In the market you can either make a bid or an offer. The transaction offers consists of a volume and a price. The bids and offers are shown on their respective side.



Transaction in process, please don't close Tribler.

BUYING

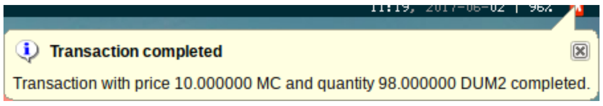
Volume	Price
14.0 DUM2	18.0 DUM1

SELLING

Volume	Price
14.0 DUM2	18.0 DUM1

Figure 3.9: a transaction in progress

If an offer and a bid match, a transaction is started. This can be seen in figure 3.9. 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 12 dummy 2 and someone else wants to buy 5 reputation for 12 dummy 2. In this case a transaction is made and the offer is then lowered to 5 reputation. One of the issues with this that we reported to the Tribler market development team is that the price does not change. The remaining 5 reputation are still being sold for 12 dummy



Transaction completed

Transaction with price 10.000000 MC and quantity 98.000000 DUM2 completed.

Figure 3.10: a completed transaction

3.3. Dispersy

Distributed Permission System NAT-firewalls Decentralized

3.4. Conclusion experiment

make a pip no selenium not just useful for us previous group had one project with different responsibilities we wanted to seperate it

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.

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 opensource we not only aim to provide this functionality to the general public after the Bachelor End Project finishes, but that others will 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.

(?....include picture of actors present...?)

4.2. Module Requirements

Figure 4.1: List of Functional Requirements

- Use commandline to access functions
- View supported hosting providers
- View supported vps
- 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

pip why

4.4. Webscraping

options for webscraping make small table, comparison and list whats important in choosing. scrapy is for massive parallel scraping options ophalen account aanmaken

4.5. Hosting Providers

hoster selection and table of properties similarities between providers, but also dissimilar since lots of possible ways to handle registration. emails. clientarea? tos obstacles(or maybe in intro)? first provider blocked from ramnode and change in testing procedures include parts of conversation with ramnode summarized and mention of ToS

4.6. payments

no renewal and why. bitpay is largest payment provider for bitcoin? coinbase also comes across quite a bit Provide overview of hosts and functionality. (maybe put provider list in introduction)

4.7. Bitcoin wallets

options for wallets, why electrum, payments

4.8. Volgend hoofdstuk

Extending and hardening cloudomate to provide more providers as

5 Hardening Cloudomate

5.1. Problems with Scrapy

Scrapy is a high level scraping framework that has a well defined way of using it. 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 then 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, in theory, 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 for 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 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 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 convientently 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 send to the server as POST request when the submit method is called.

5.4. SolusVM

SolusVM is VPS management system, it's software that 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 via a number of by the VPS provider chosen payment options. SolusVM handles the state of the payment and the 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 uses SolusVM for their registration and management. We haven't implemented a VPS provider that doesn't use SolusVM as their registration backbone. For our project, this is a convenient coincidence as the registration progress from the perspective of our code 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>
```

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
```

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

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

Since the number of arguments that have to be passed to purchase 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, coming from 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.

5.6.2. The options purchase gap

There are two different commands for extracting the VPS options from the provider's main website and purchasing this option. There had to be found a robust way to be able to select the right option through the purchase command without confusion. The options 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 purchase 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 registering process can start from this URL without danger 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.

6 Testing

6.1. Jenkins

6.2. SIG

Bibliography

[1] About tribler. URL `https://www.tribler.org/about.html`. Accessed: 2017-05-18.

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

[3] 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.

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