Dollynator

Final Report for TU Delft's Software Project
CSE2000

Giorgio Acquati, Tommaso Tofacchi, Roberta Gismondi,
Giacomo Mazzola, Daan Goossens

EEMCS
TU Delft
April 2020

Preface

This report was written by a group of five students at Technische Universiteit Delft (TU Delft), each of them completing the second year of their Bachelor of Science in Computer Science and Engineering. The report aims to present and describe the development and work done by the team on Dollynator, an autonomous self-replicating botnet of Tribler exit nodes. The work was part of the Software Project (CSE2000), iteration of 2020, in collaboration with TU Delft Blockchain Lab.

This report was written assuming the reader has a basic understanding of the difference between centralised and decentralised distributed system, along with basic knowledge on the principles of Reinforcement Learning (in particular Q-Learning). An overview of the previous efforts made on the project and its past development might also help the reader to have a clearer understanding of the subject presented. A more conceptual and high level description of the system is provided in Chapter 5, while Chapter 6 contains detailed descriptions of the system implementation. Readers who are interested in the ethical consequences of our work may consult Chapter 8 for further information regarding the aforementioned implications.

We would like to thank our TU Delft Coach, T.A.R. (Thomas) Overklift Vaupel Klein, and our Teacher Assistant Julian Van Dijk for their advice and feedback during the development. We would also thank our Technical Writing teacher, drs. M. (Mariëtte) Bliekendaal for her thorough supervision during the writing process. Finally, we would like to express our gratitude to M.A. (Martijn) de Vos, MSc and Dr. Ir. J. A. (Johan) Pouwelse for granting us the opportunity to contribute to the project.

Delft, 10 June 2020

Giorgio Acquati
Roberta Gismondi
Daan Goossens
Giacomo Mazzola
Tommaso Tofacchi

Summary

Tribler is an open source *Peer-to-Peer* file sharing client for the *BitTorrent* protocol, created as a research project of *Delft University of Technology*. *Tribler* includes its own overlay to *BitTorrent*, to provide a *Tor*-like onion routing network and aims to provide its users with anonymity and security with end-to-end encryption.

Among other projects carried on by the *Tribler* team, there's *Dollynator*, which aims to create a Darwinian reinforcement learning system based on self-replication. *Dollynator* is a system composed of several autonomous agents which run *Tribler* exit nodes. Each node can earn reputation in the form of *megabyte tokens*, which are sold on a decentralized *Tribler* market in exchange for *Bitcoin* currency. Each agent then replicates itself and buys *VPS*[1] instances to host the replicated nodes, effectively creating a self-replicating *botnet*. In order to automatically buy *VPS* instances, *Dollynator* makes use of *Cloudomate*, an unpermissioned open computer API developed as part of the *Tribler* project. Thanks to *Cloudomate*, *Dollynator* is able to automatically buy hosting services and expand itself on the internet.

[1] Virtual Private Server: virtual machine sold as a service by a *cloud* hosting service.

Contents

1 Introduction

In today's digital world, we rely on the internet as a primary source of information. Moreover, all sorts of complex systems are connected together, constituting what can be considered one of the largest projects in the history of humankind. However, it is often forgotten that Internet providers and government agencies are in control of this service and are able to restrict access to the network to its users at any moment. Therefore, the necessity for secure, anonymous communication on the internet has arisen over the past years. ***Tribler*** aims to create a decentralized system for exchange of data, and provides an anonymous and secure connection to a network of more than 2 million users [1][2]. In order to achieve this goal the system provides the community with multiple entry-points, that is Tribler exit nodes. For this reason Tribler developed ***Dollynator***: a *self-replicating* exit-node that aims to create a botnet in autonomous expansion. Nonetheless, as per now, Dollynator offers poor decision-making when it comes to replicate itself and decide which Virtual Private Servers (VPSs) to buy for its offspring. Currently, Dollynator uses a Q-Learning reinforcement algorithm that quickly converges to a single, optimal VPS provider: albeit it may seem the most obvious and effective approach, it exposes the entire network to a critical single point of failure situation, as all of the nodes may end their lives if the single VPS provider on which they are all installed suddenly becomes unavailable. Moreover, information of VPSs are simply passed from each Tribler node to its replicated offspring, which brings an important loss of knowledge of the whole network on what the optimal VPS-choice strategy might be.

To solve the outlined issues, we will be following a two-step approach. First we will focus on improving the current VPS-selection algorithm, such that it takes into account variables about the entirety of the network. We will build on the implemented Q-Learning algorithm by exploiting the concepts of *Evolutionary Game Theory* and *Swarm Intelligences*, allowing every node in the network to bring its own contribution into finding the most effective global replication strategy and avoid pitfalls caused by greedy approaches. In order to do so, nodes should be able to communicate with each other, thus leading to our second goal. To obtain agents collaboration, we will implement a *Gossip Algorithm*, which effectively exploit the distributed nature of Tribler's nodes and implements information exchange ("gossiping") amongst agents.

The report will present a more detailed analysis of the problem, including previous efforts to tackle it, in depth research on its nature and feasibility study, in Chapter 2. Chapter 3 will be reserved to list and explain the requirements gathered in agreement with the client. An overview of the preliminary work to be done in order tackle the core problem can be found in Chapter 4, while Chapter 5 and 6 explain the design of the solution we presented and the technical details of its implementation. Chapter 7 gives an overview of some extra work done with regards to Dollynator's market strategies. Eventually, a reflection about the ethical implications of our work is included in Chapter 8. Conclusions are

drawn in Chapter 9.

2 Designing a botnet of self-replicating nodes

During our collaboration with *Delft Blockchain Lab* we will focus our efforts on two main aspects of the project, namely the chance to exchange information through a decentralised communication protocol and the support for collective learning during the process of buying a VPS for replication purposes.

2.1 Make competing agents collaborate

Our botnet, just as any other example of *peer-to-peer system*, is a ***massively distributed service***. As stated by McKenzie Alexander, such a shift in scale does not come without the necessity to analyse important characteristics of large-scale distributed systems and changes in computing paradigms [3].

For example, *centralised management* is an unfeasible solution for large DSs. The presence of a central node which is able to localise data and monitor the system, maintaining a consistent and global view of it, would become a bottleneck for the system performance and a threat to its fault tolerance. Furthermore, systems of such scale are *highly dynamic*, either due to nodes leaving, joining or failing. In order to address the need of a new paradigm to deal with this premises, a gossip-based communication is introduced in [3]. ***Gossiping*** (or ***epidemic***) protocols have been around for decades now and they have shown to have many desirable properties for data dissemination, fast convergence, load sharing, robustness and resilience to failures. As stated in the paper, both traditional and not protocols adhere to the same basic gossiping framework.
Each node of the system maintains a *partial view* of the environment. Interactions between peers are periodic and *pair wise exchange of data* among peers that is organised as follows: every node selects a partner to gossip with among all its acquaintances in the network and it selects the information to be exchanged. The partner proceeds to the same steps, resulting in a *bidirectional* exchange between partner nodes.

To date, ***gossip learning*** (decentralised peer-to-peer machine learning protocols based on gossiping) is considered to be the state-of-the-art approach when it comes to train models across wide DSs. This machine learning protocol has been proved to be both scalable and efficient; yet no remarkable application in the industrial field is known. In [4] many questions on its performances in real world scenarios are answered. The goal of the study, as stated in the paper, is to validate the protocols in real-world conditions. To do that, three of the assumptions on the system that are likely to be violated in realistic environments are lifted, namely:

- a fully distributed data model
- unrestricted network topology
- homogeneous processing and communication speed

Nonetheless, the protocol is shown to possibly be extended to handle not fully distributed data and that its performance is not affected by the sample

size distribution across the nodes.
The results also proved that it can be used in networks with restricted topologies, however convergence could be extremely slow. Lastly, the protocol can also handle different communication speeds across nodes, as long as their speed distribution is independent from the data stored in the node.

[5] investigate a distributed multi-agent reinforcement setup in a distributed system, where agents follow a distributed Q-Learning technique (***QD-Learning***) and collaborate by means of a local processing and mutual information exchange to converge to the overall system-like optimal solution. The proved success of this solution suggests, once again, the validity of a gossip protocol application to the learning process of the algorithm responsible for the VPS selection in our botnet replication task.

2.2 Evolutionary based reinforcement learning approach

Evolutionary Game Theory is an application of traditional, mathematical theory of games to evolutionary prone, biological contexts, proposed as a consequence of how strategic aspects of evolution are affected by frequency dependent fitness functions. According to [6], there are two main approaches to EGT. The first one, derived by the work of Smith and Price, identifies as its principal analysis tool the concept of evolutionary stable strategy (ESS). The second approach constructs a dynamic model by which the frequency of the strategies changes in the population and aims to analyse the evolutionary dynamics that ensue that assumption. As it is easily inferred by the problem description, the latter approach is more of interest to us. While evolutionary Game Theory has been used to give insight with regards to multiple aspects of human behaviour, it has become of increased interest to computer scientists as well, due to the many possible applications of its principles and concepts to computationally intelligent models for optimisation purposes.

As stated in [7], an Evolutionary Game Theory approach can be taken towards five main research agendas of ***multi-agent learning***, including algorithms describing how agents learn in the context of other learners and how they should cooperate in order to achieve distributed control of dynamic systems. In particular, the properties mentioned in [8] of ***multi-agent systems*** (MAS), such as our botnet, seem to correspond well with EGT properties for two main reasons. First, MASs consist of independent agents, each trying to accomplish a certain goal while not being aware of other agents' intentions nor the exact global state of the environment (in other words, with *limited viewpoint*). Secondly, MASs are *dynamic systems* where agents change their behaviour over time as a consequence of other agents' behaviours. EGT provides a framework in which consequences of these assumptions can be easily analysed.
Therefore, an evolutionary game theory approach to PSO (***particle swarm optimisation***) was proposed in [9]. PSO is a swarm intelligence algorithm that can be used to depict the sociological behaviour of a group of people. The algorithm variant presented in the paper aims to introduce evolutionary game theory in the traditional PSO formulation, proposing an evolutionary game

based particle swarm optimisation algorithm (EGPSO) that uses replicator dynamics to analyse the learning of the agents in the model. Simulations proved the algorithm efficiency, suggesting once more that an EGT integration could be of wide benefit to our purpose.

Nonetheless, the assumption of a *unique*, global best option to aim to is, in the case of our botnet, a possible pitfall of the model. In fact, due to fault tolerance requirements, it is of great importance that some variety in the VPS provider is taken into account while choosing the best trade-off option. An algorithm that converges to the choice of the same single VPS offer for every node in the system could lead to inauspicious consequences if a server provider shutdown occurs for whatever reason. As far as multiple optima are concerned, [10] presents a way to handle ***multi-objective optimisation*** problems using *multi-swarm cooperative particle optimisers* (MC-MOPSO). The algorithm organises the set of particles in a master-slave swarm architecture. Each slave swarm is designed to find all the non-dominated optima of the multi-objective problem, then the master swarm generates a set of non-dominated solutions starting from the results of the slaves, in order to eventually find the optimal solution to a multi-objective problem; this and other shrewdness presented in [10] produce a PSO option which turns out to be highly competitive to solving multi-objective optimisation problems.

2.3 Feasibility study

The need for an autonomous, self-replicating botnet comes from Tribler's intention to protect its users' privacy. To access the Tribler network you need a Tribler exit-node. As stated in [11], an exit-node is a publicly visible entryway into the anonymous network; being the connection between anonymity, notoriety leaves this entity vulnerable.

Dollynator and Cloudomate cooperate in order to provide Tribler users with a fortified and decentralized network of exit-nodes.

Tribler community counts roughly 231k unique users. This information alone already represents a solid reason for our team to contribute to an overall improvement of Dollynator, the software which is responsible for the network reach and entry-point provision.

The changes that we aim to apply to the existing code base consist of the integration of new protocols (namely Gossiping protocols for distributed learning) and the modification of existing algorithms (such as a revision of the computational intelligence responsible for the replication dynamics). The first point of improvement, namely the introduction to the network of a form of communication following the gossiping paradigm, is reasonable to undertake as it satisfies the need for a way to reliably and consistently share information across the nodes, both for fault tolerance and decision-making purposes. The protocol is shown to have good performances on distributed systems with characteristics that are similar to those of our botnet [4], thus making the choice of gossiping a feasible solution. The second operation in our agenda has multiple possible implementation options. On one hand, we evaluated the possibility of improv-

ing the existing Q-Learning algorithm. On the other hand, we researched more suitable options, such as Swarm Intelligence models. Either way, being the two alternatives both solid and reliable solutions for decision-making tasks [5] [9], they both represent suitable approaches to our core problem.

More generically, every aspect of our go-to strategy has been validated by thorough research and past industrial employment. Moreover, it does not comport further financial investment in hardware technology or infrastructure.

As per now, software maintenance and preliminary work on the present-day system are in order before we can start tackling the core problem; still, taken into account the current state of the software and the full time commitment of each member of the development team, we can conclude that the time span of ten weeks is appropriate for the estimated workload.

2.4 Risk analysis

Due to the nature of Tribler and Dollynator, our plan of development of the project is exposed to risks on different levels, summarized and further explained as follows:

- *external risks*, which affect the project directly, but are eventually involving issues occurring outside of Dollynator's scope.
- *implementation risks*, that may arise when progressing on one or more multiple requirements.
- *team-related risks*, concerning our basis of knowledge on what we are going to develop.

First of all, issues may arise as a result of failures in Cloudomate. With Cloudomate being the only viable option to buy VPS services for the exit-nodes, its incorrect operation would prevent us from examining any improvement to Dollynator in a real-life scenario. At the current state, Cloudomate is not guaranteed to function with the previously implemented VPS providers, thus our need to check – and eventually fix – its correct behaviour. Due to time constraints, our goal is to fully re-implement just one provider; however, as stated in the requirements section, one of the Must Haves concerns the refinement of the VPS selection algorithm. In order to deploy the software out of the testing scope and check its behaviour when choosing the optimal VPS plan in a production environment, access to multiple VPS providers is required. Such a feature could not be granted even if all of the currently supported providers were to be fixed: the aforementioned providers may, for instance, change their registration web pages and nullify the efforts. To prevent such a risk, we will set up a testing environment which will enable us to verify the correctness of our algorithm's implementation independently of similar situations.

A second pitfall of the project could emerge during the Gossip learning implementation. Although we have found a variety of sources about the topic

which would grant feasibility in applying this model under each single Dollynator's constraint, we have not encountered any reading proposing an application of Gossip learning while all of said constraints are enforced at the same time. Albeit unlikely, the possibility of failing to deliver a correct implementation of the technique due to the current organization of Tribler and Dollynator – either because of technical-related issues or security-related concerns – is to be considered.

Lastly, a factor of risk is given by the lack of general programming experience in the field of distributed systems. No member of the team has previously coded focusing on this aspect of the project, although we are all acquainted with the concepts related to it. Due to this and to the ease of access to highly informative sources on the matter, we are confident in stating that such a risk will be minimized throughout the project development. A similar argument could be raised for other specific topics as well – laws of finance when dealing with implementing new market strategies, for instance -, but in all of these cases the same consideration can be applied.

3 Requirements

After a thorough analysis of the pre-existing code-base and the previous efforts on the Dollynator project, taking into account the recommendations and remarks of our predecessors and consulting the client we established a set of requirements to be fulfilled as part of our contribution to the project and by the end of our experience. The requirements consist of both functional and non-functional specification for our system. The functional requirements are presented in MoSCoW style first and further explained in a later sub-section, the non-functional requirement are listed in the last sub-section of this chapter.

3.1 Functional requirements

The process of requirement analysis, carried out along with the client and supervised by both the TA and TU Coach, resulted in the following list of functional requirements.

An overview of the specifications for our system is given below as a MoSCow list; the requirements are divided into *Must Have*, *Should Have*, *Could Have* and *Won't Have*. Further explanation of each of the requirement in this list can be found in later sub-sections.

3.1.1 MoSCoW

- **Must Have**:
  - Cloudomate must support at least one VPS service.
  - Nodes must be able to share information through gossiping.
  - Nodes must be able to perform reinforcement learning on the information gathered from the network, and exploit such information for VPS provider choice.
  - Dollynator must have support for the latest version of Tribler.
  - Dollynator must be upgraded from Python 2.7 to 3.6.
- **Should Have**:
  - Agents should be able to transfer Bitcoins before dying.
  - The agents' network should be resilient to permanent nodes failures.
- **Could Have**:
  - Cloudomate could support multiple VPS providers.
  - Agents could have access to new market strategies (moving average / deep learning).
  - Agents could apply a Q-learning algorithm for choosing VPS, based on multiple service features.

  - Agents could apply a Q-learning algorithm for choosing the market strategy.
  - Dollynator could have a visualisation tool for the botnet.

- **Won't Have**:
  - Cloudomate will not implement new features.
  - Dollynator won't have an algorithm to choose what VPN service to purchase.

3.1.2 Must have

- **Cloudomate must support at least one VPS service**: Since Cloudomate has not been updated in a while, all of the VPS providers are not compatible with it anymore. Therefore, we need to update Cloudomate to restore compatibility with VPS providers. Our goal is to get Cloudomate to support at least one VPS providers, which will enable us to run Dollynator in a real-world scenario.
- **Nodes must be able to share information through gossiping:** In order to reach better cooperation between agents, we want to implement a gossip algorithm to share important information between agents. This will enable the agents to make a more informed decisions throughout their life cycle (e.g. VPS service choice for replication).
- **Nodes must be able to perform reinforcement learning on the information gathered from the network, and exploit such information for VPS provider choice:** as stated in the previous requirement, we want to create a network of collaborating agents. Once the nodes are able to communicate and share information, we want to exploit such information by applying reinforcement learning on VPS service choice.
- **Dollynator must have support for the latest version of Tribler:** in the current state, Dollynator uses an old version of the Tribler protocol, which is not compatible with the current release. Therefore, we need to update Dollynator in order to have the botnet run Tribler exit nodes.
- **Dollynator must be upgraded from Python 2.7 to 3.6:** Since Python 2.7 reached its EOL[2] deadline, we want to upgrade Dollynator to Python 3.6 in order to make it more maintainable in the future.

3.1.3 Should have

- **Agents should be able to transfer Bitcoins before dying**: When a node's life cycle comes to an end, all of the Bitcoin currency that it has earned needs to be transferred to other nodes in order to prevent its lost.

[2]End Of Life

- **The agents' network should be resilient to permanent node failures**: since an agent could be permanently shut down at any time by its VPS provider, agents should share the private key to their Bitcoin wallet in order to prevent loosing access to it.

3.1.4 Could have

- **Cloudomate could support multiple VPS providers:** adding support for multiple VPS services would make Dollynator less dependent on any of the providers, resulting in a more resilient botnet. Having multiple VPS services to choose from would also enable us to test the VPS choosing algorithm in a real world scenario.
- **Agents could have access to new market strategies (moving average / deep learning)**: Currently, the market strategies applied by the nodes for selling reputation tokens are very basic and are not very efficient when establishing the price for said tokens. We would like to improve these strategies by implementing different approaches which could increase the revenue generated by the nodes and therefore increase their chance of reproduction.
- **Agents could apply a Q-learning algorithm for choosing VPS, based on multiple service features**: Currently, each node reproduces using the cheapest VPS option available, but not taking into account the different features(CPU cores/memory/bandwidth) offered by different VPS services, which could potentially make a node perform better. We would like to implement a Q-Learning algorithm that takes these features into account as well.
- **Agents could apply a reinforcement learning algorithm for choosing the market strategy**: If market strategies are added, there are more options for the agents to choose from, so reinforcement learning can be used to learn to use the optimal market strategy.
- **Dollynator could have a visualisation tool for the botnet**: We would like to implement a tool that creates visual representation of the state of the network or parts of it. This tool would be useful to gain insights about the network for both development and marketing purposes.

3.1.5 Won't have

- **Cloudomate will not implement new features:** Our goal is to bring Cloudomate to a functioning state, but not to implement new features. The focus of our project is Dollynator, which determines what improvements we need to make to Cloudomate.
- **Dollynator won't have an algorithm to choose what VPN service to purchase:** Different VPN services offer different benefits and implementing an algorithm to establish which one would be the best option

would be beneficial. However, we are not going to focus on this aspect of the project which could make for a good starting point for future improvements.

3.2 Non-functional requirements

The requirements analysis produced a list of non-functional specifications for our system. The following list contains both system-related features and norms regarding the development process.

- The agents will run on VPS running Ubuntu.
- Dollynator shall have at least 70% line test coverage.
- The team should apply "pull based development".
- Pull requests need at least two approvals, excluding the contributors of the code under review.
- Code needs to be commented: each method needs a comment that explain the behaviour of the method

4 Preliminary work

Dollynator received its last *GitHub* contribution in January 2019. Before further developing any other aspect of it, the project needs to be reviewed in order to function in nowadays world: dependencies to other libraries/projects (*Cloudomate*, for instance) need to be updated and, more in general, the entire codebase requires fixes according to the changes in the environment occurred in the past year.

4.1 Updating to latest version of Tribler

The purpose of our network is to run a botnet of Tribler exit nodes. In order to allow the agents of the network to be used as exit-nodes, Dollynator strictly relies on the Triber submodule. Compared to older releases of the software, Tribler latest version was subjected to heavy refactoring operations. For this reason, both the software dependency and its submodules import were to be adjusted.

In particular, Tribler reference in PlebNet was updated to point to the devel branch of its repository. Furthermore, to allow PlebNet to effectively import and exploit Tribler modules, the respective paths to those submodules were included in the PYTHONPATH variable through the install script. Also the PYTHONPATH environment in the twisted module, that is accountable for running Tribler in the background, was updated.

4.2 Fixing Cloudomate

Another relevant dependency of the Dollynator system is *Cloudomate*. Cloudomate is the software responsible for smooth and automated VPS purchases. Its services are consumed by Dollynator both during the decision process (to retrieve VPS providers and options) and during the replication process (the VPS to replicate onto is bought using Cloudomate). Due to this essential role in the economy of our system, part of our contribution to the project was to make sure that Cloudomate successfully integrates at least one VPS provider.

The first step to achieve this goal was to run the Cloudomate test suite provided by our predecessors. Tests regarding the process of VPS purchasing were of two types, according to their testing purposes. Part of them aimed to check whether the option regarding the providers' offers and services could have been retrieved using the software, while the rest was responsible to test that the system was indeed able to successfully buy one of these options in an automated fashion.

As it turned out, only one of the VPS providers integrated by the previous group was still able to successfully pass all these tests, namely *QHoster*.

Nonetheless, to optimise the efforts, the *Linevast* provider was the first provider whose integration was analysed and fixed as it is the one whose interface changed the least over time (changes were estimated using waybackmachine to retrieve the state of the website by the time Cloudomate was lastly

developed). As Linevast's gateway for bit-coin payments changed from *BitPay* to *Coinbase* a new way of conducting automated payments was needed. To this purpose, *Selenium* was employed. *Selenium* is a tool that allows automated browsing; in the case of Cloudomate it was used to develop an automated Firefox browser. The browser was used to proceed to the payment process and to retrieve payment details.

In order to test that Cloudomate was able to successfully run in a real-world scenario we used part of our budget to conduct outer-world experiments. Unfortunately, the tests suggested that Linevast wasn't a dependable service to rely on. QHoster, on the other side, was integrated to Cloudomate. Although the system is able to successfully buy a QHoster VPS instance, the available purchase options for the provider all present kernel incompatibilities with the latest version of Tribler, for this reason, it is currently impossible to run an exit nodes on the purchased server.

4.3 Upgrading to Python 3.6

The preexisting code-base for Dollynator was developed in *Python 2.7*. As Python 2 approaches its EOL[3], the need for an upgrade to a later version of the language was in order. For this reason, part of our contribution to the project was to modify the code baseline to support *Python 3.6*.

In order to adapt the existing Python 2 code to be compatible with a Python 3 interpreter, the *futurize* script from python-future package was run.
The *futurize* script passes Python 2 code through all the appropriate fixers to turn it into Python 3 code. The script is advised to be run in two stages: stage one is for "safe" changes that modernise the code but do not break Python 2 compatibility while stage 2 is to complete the process [12]. Only stage one was run on our project, while many adjustments were made manually.

In order to guarantee completely functional Python 3 support changes were made both to the Python code of the project itself and its dependency-management related components. The scripts responsible for dependency installation and requirements fulfilment were updated to fetch the latest (and thus Python 3 compatible) version of the packets.

The preliminary work made to upgrade the code-base to Python 3 allowed us to safely proceed developing the software using a reliable and stable version of the programming language.

[3] End Of Life

5 Product design

Our goal is to provide *Dollynator* with a renewed and better algorithm for decision making purposes, based on a collective learning model where each and every node in the botnet is able to fetch and exploit information from the entire network. It order to achieve this goal, we designed the product in two main components: a protocol, based on gossip learning, that aims to conveniently share information between agents and a new, distributed, reinforcement learning algorithm, which is able to consume the aforementioned information and result in an effective decision-making process.

5.1 Choosing a VPS provider for replication

One of the core components of our system is the algorithm responsible for purchasing the Virtual Private Server where a node will replicate itself onto. The process of choosing a service provider is conducted using Reinforcement Learning techniques. Before our contribution to the project, a node would base its decisions on information and previous experience regarding the node's life cycle only, without any contribution from external agents or botnet components. Our main goal was to modify the current algorithm to include the ability of learning in a "collective" fashion, i.e to gather relevant information from other nodes in the botnet and exploit it to pick a VPS to purchase.

In order to do so, we employed a collective variant of Q-Learning, namely QD-Learning. QD-Learning is a distributed version of Q-Learning, in which agents of a network collaborate by means of local processing and mutual exchange over a communication network to achieve a shared goal. Q-Tables are updated, by the time of a new purchase, both locally and by means of a merging process of tables from the network. The new, updated value is a weighted sum of the local and remote information, where the weights of the two components change over time to give increasingly higher relevance to the former source of knowledge.
Remote Q-Tables are retrieved through gossiping and exploiting the communication protocol described in the next section.

The update processes of Q-values and environment tables were also slightly modified. By the time of a purchase, the entire column regarding the action of the purchased VPS is updated: regardless of the current provider, the action of buying a VPS is either penalised - therefore discouraged - if the purchase fails, or rewarded if the purchase is successful. This approach is to a certain extent different from the one which was previously adopted. The former version of the Q-Learning technique only penalized the transition from a specific current-state to the failing action. The algorithm was also modified to positively reward the current VPS at the end of every life cycle. The positive update value in the environment table is computed as a function of MB Tokens and the cost of the option that it is relying on.

5.2 Communication between nodes

The communication between nodes is one of the most relevant aspects of our contribution to *Dollynator*. The baseline for our design was a communication system based on the *Internet Relay Chat (**IRC**)* protocol. The IRC protocol is for use with text based conferencing. An IRC network is defined by a group of servers connected to each other and the only configuration allowed is that of a spanning tree where each server acts as a central node for the rest of the network it sees. The protocol provides no mean for two clients to directly exchange information and the communication between clients is relayed by the server(s) [13]. Our intention was to substitute such a centralised system with a *peer-to-peer* network where every node is dynamically provided with the necessary information to directly communicate with every other node in the botnet, thus resulting in a fully connected network topology. The complete network allows every node to easily exchange information with the rest of the botnet and constitutes a basis for the gossiping protocol.

The information about the other nodes of the network (including their IP and their current state) is gathered from the network itself. When a node replicates it passes to its child all the knowledge it has about the botnet. This knowledge is shared in the form of a *contact-list*, where a "contact" contains all the necessary information to directly reach a specific node. The parent contact-list will, therefore, contain the contacts relative to all the nodes inside the botnet it is aware of. Moreover, at the moment of replication the parent also takes care of informing its acquaintances about the new born node and share with them its contact. When the latter is received, the nodes proceed informing the elements of their contact-list about the new contact received, and so on. When a message to one of the contacts is not delivered, the receiver is detected as potentially down. This failure triggers a ping procedure: the sender systematically tries to contact the failing server; after a customizable time span, if none of the communication attempts succeeded, the receiver is considered down and its contact is deleted.

The communication between nodes is encrypted and every message exchanged (both regarding network awareness and collective learning) is signed and authenticated.

6 Implementation

The technical details of the product implementation are listed below. The reinforcement learning algorithm was modified and updated to include support for *QD-Learning* (distributed Q-Learning) whilst the communication protocol was implemented based on *Berkeley Socket API* and it exploits the basic principles of *gossiping* for information sharing across the network.

6.1 Reinforcement learning for VPS choice

The process of VPS selection is conducted exploiting information coming from the network in the form of ***Reinforcement Learning***. Reinforcement Learning was also the go to strategy before our contribution to Dollynator. The RL technique used was *Q-Learning* and the information necessary to efficiently buy a new and convenient VPS were only passed from parents to children. The algorithm that was developed during this iteration of the Software Project aimed to exploit information from every other node of the botnet and thus learn how to make decisions in a *collective* fashion. Thus, the changes made to the preexisting code-base can be classified in two categories: changes that affect the way the *Environment* and the *Q-Tables* are locally updated and changes that aim to allow support for collective learning.

The first significant change to the algorithm regards the initialisation value of the Environment components. In the previous version of the software, the value was not constant and derived based on the price of a purchase option and its bandwidth. This detail was modified to initialise every value in the environment to the same, *constant*, quantity.

The second change regards positive and negative reward after an attempt of purchasing. In the past, when a node managed to successfully buy and replicate, all the transitions leading to the current state were updated positively; while in case of failure, only the transition between the current state and the chosen failed state was penalised. This was changed to reward and penalise all the transitions leading to respectively a successful or failing state, and to benefit all the transitions leading to the current state in case of fortunate replication. The update value of the latter is computed as a function of the number of MB tokens that the agent was able to produce per cost of the current VPS option; this value is normalised using data on past exit-node performance gathered from previous reports[11].
The changes made in order to support collective learning in our algorithm aimed to exploit a gossiping protocol to exchange information across the network in form of Q-Tables. This distributed alternative to the traditional learning was implemented in the form of ***QD-Learning*** [5]. By the time of a replication process a node exchange information about its current knowledge with a number of other nodes. The in going Q-Tables (coming from the rest of the botnet) are then merged to the local one by means of a weighted sum. The weights of the

sum (that are initially set to 0.8 for the local information and 0.2 for the outside knowledge) are repeatedly updated to reflect the relevance of the two sources of information as a function of the botnet size. For this reason, the local value (α) is systematically decreased and the remote value (β) is increased instead. This process continues up until α and β are swapped (they are now 0.2 and 0.8 respectively), this equilibrium guarantees that the local information will always be relevant in the decision process of a node.

6.2 Gossip algorithm

A ***socket*** is one end-point of a two-way inter-process communication link and thus the basis of the communication protocol responsible for inter-node messaging in the botnet. Python's socket module provides an interface to the *Berkeley Sockets* API.

The core objects of the implemented design are, in fact, a *MessageSender* and a *MessageReceiver* which are responsible to manage the end-to-end socket communication between two nodes. The two objects respectively send encrypted information to a specific node and receive and decrypt messages, with the *MessageReceiver* also responsible to function as a simple message-broker.
In fact, an observer design pattern is implemented to notify parts of the system with different prerogatives about incoming messages of their specific interest. The messaging service implements channel based messaging, which allows multiple components of the system to subscribe to different virtual channels. Upon receiving a message, the *MessageReceiver* is responsible for notifying the consumers registered to the channel to which the message belongs. This architecture allows great flexibility, since any new set of components can allocate new communication channels on which communicating, without affecting any other part of the system in any way. As per now, a node's *MessageReceiver* has two main consumers, namely the *AddressBook* and the *LearningConsumer*.
The role of the latter is to fetch information for learning purposes and consume it performing reinforcement learning in a collective fashion.

The *AddressBook*'s role, instead, is to keep track of other nodes' state and contact information. The *AddressBook* main task is, therefore, to make sure to store a complete and reliable list of *contacts*. A node's contact contains a unique ID, its host name and the port on which the *MessageReceiver* runs. The contact also contains the RSA public key used to encrypt messages sent to the node.
In order to update its contact-list the *AddressBook* exchanges messages containing network information, hence it subscribes to the *network* channel. The *AddressBook* is also responsible for sending messages to other nodes, since it also keeps track of failed communication with other nodes. When a message is not correctly delivered (an error occurred or an ACK message is missing) the *AddressBook* starts periodically pinging the recipient of the message. If the sender *AddressBook* is unable to contact the receiver after several attempts, it

marks the receiver as "permanently down" by deleting the receiver's contact, effectively disabling communication with it.

As previously mentioned, each message exchanged using the developed protocol is encrypted and signed, so that only the designed receiver is able to read the message's content and to verify the signature of the sender. To achieve this goal, a mixture of ***RSA*** (*asymmetric*) and ***AES*** (*symmetric*) encryption is employed, in a similar fashion to how *PGP* is designed. In fact, the RSA asymmetric cryptography algorithm is used to exchange a symmetric AES key between the sender and the receiver. The AES symmetric key is then used to encrypt and decrypt the actual massage payload. Moreover, RSA is employed to sign the content of the messages. These security measures guarantee our protocol to be functional and dependable.

7 Extras: a new market strategy

One of the possible point of improvement of the Dollynator system was the introduction of some new, more reliable strategies to deal with the bit-coin market. In order to achieve this goal, we introduced crossover to the pre-existing simple moving-average strategy that Dollynator was implementing.

7.1 Crossovers and moving averages

The ***moving average*** is a simple technical analysis tool to smooth out price data by creating a constantly updated average price. The average is taken over a specific period of time and it aims to detect and deal with *market fluctuations*.

Dollynator already implemented a MA strategy, namely the *simple moving average* strategy. A five-day simple moving average (SMA) adds up the five most recent daily closing prices and divides it by five to create a new average each day. The algorithms then compares them in order to establish whether there is a possibly increasing or decreasing trend in the market. Our contribution to this model consisted of two main parts, namely to include an *exponential* moving average to the average calculation (where more recent data affects more the outcome of the function) and to implement *crossovers*.

Crossovers are one of the main moving average strategies. The implementation we opted for, in particular, is to apply two moving averages to a chart: one longer and one shorter. When the shorter-term MA crosses above the longer-term MA, it's a signal that the agent should wait to sell for bitcoins, as it indicates that the trend is shifting up. Meanwhile, when the shorter-term MA crosses below the longer-term MA, it's a sell signal, as it indicates that the trend is shifting down.
This helps Dollynator to effectively deal with the bit-coin market and to optimise its bit-coin income in an automated way [14].

7.2 Moving average implementation

In order to include crossovers in Dollynator moving average strategy some refactoring of the previous code was in order. The code-base included an entire module which aimed to both compute the average, compare them and interact with the market. This approached was error prone as it implied very low cohesion in the code.

For this reason, the code-base was modified to include a MovingAverage template class, which is in charge of the moving average computations according to an exponential model and given a variable time-span. The rest of the computations, such as averages comparison and decision making, is taken care of by a CrossoversMovingAverages class that inherits from MovingAverage.

This way we incremented the cohesion of the classes of the subsystem and improved maintainability and testability of the code.

8 Discussion on ethical implications

As a consequence of our commitment to the Tribler team, a reflection on the ethical consequences of our work is in order. After a detailed analysis of the possible moral implication of this project, we narrowed down our ethical thoughts to two main topics, both strictly correlated to Tribler aforementioned intention to provide its users with anonymity and privacy.

Anonymity is defined as the state of remaining unknown to most other people. In other words, it is the impossibility of being identified in a given context. This characteristic can be beneficial or even necessary to individuals in a variety of occasions and for this reason anonymity itself is acknowledged by the *'Declaration on freedom of communication on the Internet'* or *'Report of the Special Rapporteur on the promotion and protection of the right to freedom of opinion and expression'* [15].
Considering this, one could state not only that the usage of Tor-like tools is legal, but that it is also an ethical choice, a valid decision to preserve the well-being of the users of the internet. Nonetheless, it is worth specifying that the morality of anonymity is *context dependent*, hence it would be misleading to consider it an intrinsic value of human-kind.
Tor-like systems can indeed allow and encourage a immoral behaviour among its users.
Anonymity on the internet notoriously stimulates despicable activities such as copyright infringement, cyberbullying and the spread of morally compromised contents. In fact, if an action is deemed illegal and unethical, perpetrating it anonymously should also be considered illegal and unethical. For this reason it is better to define anonymity as an extrinsic value: a medium to defend - for example - the freedom of expression, the freedom of association, the right to privacy and all the values that ensue the latter.

Privacy is indeed an important topic in this discussion as well. It can be argued that there is a "right to be left alone" based on a principle of "inviolate personality", and it is therefore difficult to conceive of the notions of privacy and discussions about data protection as separate from the way computers, the Internet, mobile computing and the many applications of these basic technologies have evolved [16]. Two main kind of privacy can be identified, namely *constitutional* (or decisional) privacy and *tort* (or informational) privacy. Although both rights are unarguably important, from now on we will refer to privacy as the latter, which is concerned with the interest of individuals in exercising control over access to information about themselves. Along with an attempt to define privacy, in [17] the concept of right to privacy is introduced. The right to privacy is not my right to control access to me, it is my right that others be deprived of that access." Having privacy is not the same thing as having a right to privacy. I can have either without the other [17]. Nonetheless, privacy is a right that all people should have, however unfortunately it happens often that people's privacy gets violated. Two striking examples of privacy violations, one

by a government entity and the other by a private company, are testified and reported by respectively [18] and [19]. The former one talks about the China's government's behaviour towards the population during the Corona virus crisis, while the latter one is about the big scandal that involved Facebook and Cambridge Analytica.
These articles are key resources to understand why privacy is a fundamental step to reach freedom. More related to our work, people should have the chance to freely browse the internet and fetch contents from it. Interference by entities that prevent users of the internet from retrieving information and/or doing so in a secure and enjoyable way should be discouraged and avoided as much as possible. For this reason, privacy is important since it protects people from annoying custom advertisement or, in worse scenarios, blackmailing and government restrictions that often threaten ones' freedom of expression and opinion. Tribler helps to guarantee people's privacy thanks to the anonymity of its users, provided through the usage of Tor technology and by the adoption of a Peer-to-Peer model, where the absence of a central node prevents any server from being able to control network traffic. However, analogously to what has been said before with regards to anonymity, this kind of advantage is context dependent too. In fact, since nobody can track down the user of an illicit request, a network such as Tribler could give the possibility to its users to share in an easier way content protected by copyright, confidential information, pedopornography material, etc.

Although, as much as we would like to avert such an occurrence, this is clearly a controversial situation since as per now the only way to prevent this from happening is to control each node's traffic, violating their privacy and rejecting the concept of a decentralised network.

In conclusion, all the values we talked about are extrinsic, content-dependent values, which are to be considered essential to people's well-being, but can harm other people if used as a medium to adopt an immoral behaviour.

As threatening as it appears, relying on Tribler users' ethical conduct is as per now unavoidable. As with many other social dynamics, the premise of every member of a society behaving in their interest without diverging form a common moral scheme is an essential one.

Absence of privacy is, in many ways, a dangerous condition that can deprive individuals of inalienable rights and inauspiciously affect their well being; for this reason, and despite the aforementioned risks, we have chosen to design for this value.

9 Conclusion

The aim of this report was to describe the developing process of Dollynator, an autonomous self-replicating bot of Tribler exit-nodes. The main goal of the project was to include support for collective learning in the process of VPS purchasing. This goal was achieved by means of two main modification: a gossiping protocol for node communication in the botnet was developed and the Reinforcement Learning algorithm responsible for VPS purchasing was modified to adhere to the standards of QD-Learning (Distributed Q-Learning).

The advantages and disadvantages of these designs were evaluated and analysed in comparison to other possible solutions and eventually adapted to satisfy the specific needs of our system. Along with that, preliminary work was made to upgrade the base-line to a real-world deployable version of the code.

During the software development many difficulties were encountered, mainly related to Dollynator strict dependencies to other software, especially Cloudomate and Tribler. Being the two software currently under development as well, and due to a lack of documentation on the current state of the systems, we encountered some unpredictable issues with their integration with Dollynator. As far as Tribler is concerned, its upgrade to the most recent version caused an impossibility to successfully run the botnet nodes as Tribler exit-nodes. Tribler is, as per now, suffering from a bug that prevents a correct instatiation of a bitcoin wallet. This is because the *bitcoinlib* library responsible for the bitcoin-wallet management is currently being updated to a later version and its integration is not final and functional yet. As far as Must Have number one is concerned, the task was accomplished but not entirely functional yet. Our system is, in fact, able to successfully purchase a VPS from QHoster provider. Nonetheless, kernel incompatibilities with the latest Tribler version prevents a smooth execution of the software. Overall, we consider the system to be in a satisfactory state as every other essential requirement was achieved, including the Should Haves and one of the Could Haves in our list.

The solution adopted is solid and consistent but its real-world deployment is prevented by a hard dependency on Cloudomate, a part of the system which lacks maintainability and, as per now, real-world applicability. For this reason, it is not easy to establish the quality of Dollynator performance out of a testing environment. Our recommendation is to improve the quality of Cloudomate software and its functionality as it would allow Dollynator's decision process to be deployed, evaluated and fine tuned in the out-side world.

The work-flow and team-performance was characterized by a steep learning curve due to the complexity of the system under analysis. Dollynator is made up of multiple, different parts and it is important to take into account a significant amount of time to get acquainted with its implementation and functioning before diving into further developing.

References

[1] Tribler team. *Tribler*, 2020. Accessed on: Apr. 28, 2020. [Online]. Available: `https://github.com/Tribler/tribler/blob/devel/README.rst`.

[2] Tribler team. *Tribler - Privacy using our Tor-inspired onion routing*, 2020. Accessed on: Apr. 28, 2020. [Online]. Available: `https://www.tribler.org/about.html`.

[3] E. Riviere and S. Voulgaris. "Gossip-based networking for internet-scale distributed systems". *Lecture Notes in Business Information Processing*, 78:253–284, jan 2011.

[4] L. Giaretta and S. Girdzijauskas. "Gossip learning: Off the beaten path". *2019 IEEE International Conference on Big Data (Big Data)*, pages 1117–1124, dec 2019.

[5] S. Kar, J. Moura, and H. V. Poor. "QD-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus + innovations". *IEEE Transactions on Signal Processing*, 61, apr 2012.

[6] J. Alexander McKenzie. *Evolutionary Game Theory*. The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, 2019.

[7] K. Tuyls and S. Parsons. "What evolutionary game theory tells us about multiagent learning". *Artificial Intelligence*, 171:406–416, may 2007.

[8] K. P. Sycara. "Multiagent systems". *AI Magazine*, 19(2):79, jun 1998.

[9] L. Wei-Bing and W. Xian-Jia. "An evolutionary game based particle swarm optimization algorithm". *Journal of Computational and Applied Mathematics*, 214(1):30 – 35, 2008. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0377042707000799`.

[10] Z. Yong, G. Dun-wei, and D. Zhong-hai. "Handling multi-objective optimization problems with a multi-swarm cooperative particle swarm optimizer". *Expert Systems with Applications*, 38(11):13933 – 13941, 2011. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0957417411007275`.

[11] R. van den Berg, J. Heijligers, and M. Hoppenbrouwer. "Plebnet, botnet for good". *Delft University of Technology, Student bachelor thesis*, jul 2017. [Online]. Available: `https://repository.tudelft.nl/islandora/object/uuid%3A08841cf0-8ddf-4354-9c99-bc7c270d67fd?collection=education`.

[12] Python Charmers Pty Ltd. "Futurize: Py2 to py2/3", 2013-2016. Accessed on: Jun. 7, 2020. [Online]. Available: `https://python-future.org/futurize.html`.

[13] C. Kalt. "RFC 2810 - internet relay chat: Architecture", 2000. Accessed on: May, 28. 2020. [Online]. Available: https://tools.ietf.org/html/rfc2810#section-3.

[14] C. Mitchell. "How to use a moving average to buy stocks", 2020. Accessed on: Jun. 19, 2020. [Online]. Available: https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp.

[15] E. Çalışkan, T. Minárik, and A.M. Osula. "Technical and legal overview of the tor anonymity network". *NATO Cooperative Cyber Defence Centre of Excellence (the Centre)*, 2015.

[16] J. van den Hoven, M. P. W. Blaauw, and M. Warnier. "Privacy and information technology". In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2020 edition, 2020.

[17] J. H. Reiman. "Driving to the panopticon: A philosophical exploration of the risks to privacy posed by the highway technology of the future". *Robotics and Autonomous Systems, Santa Clara High Tech. L.J*, pages 97–103, jan 1995. [Online]. Available: http://digitalcommons.law.scu.edu/chtlj/vol11/iss1/5.

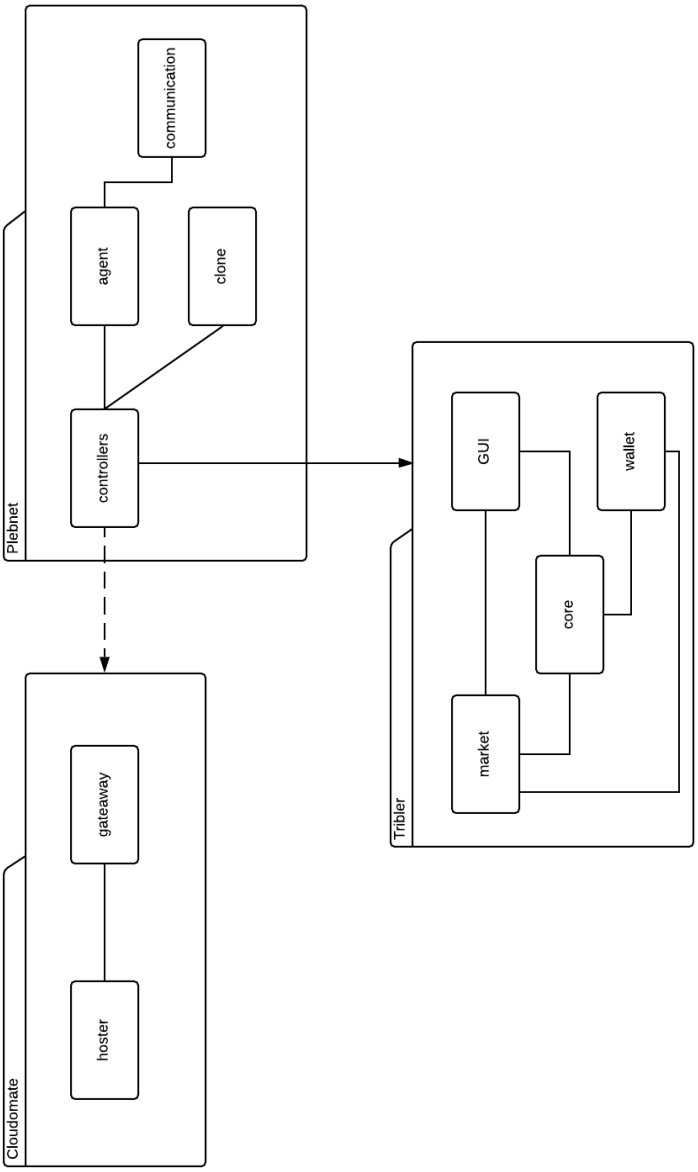
[18] The Guardian. "'More scary than coronavirus': South korea's health alerts expose private lives", 2020. [Online]. Available: https://www.theguardian.com/world/2020/mar/06/more-scary-tan-coronavirus-south-koreas-health-alerts-expose-private-lives.

[19] The Guardian. "Facebook to be fined 5bn dollars for cambridge analytica privacy violations", 2019. [Online]. Available: "https://www.theguardian.com/technology/2019/jul/2/facebook-fine-ftc-privacy-violations.

Appendix A - Product Feedback

In this appendix we will describe the feedback received by the SIG (Software Improvement Group).
We contacted them in week 4.5 in order to ask for a product review, but unfortunately they answered that, due to the Corona virus situation, they didn't have enough time and men power to do it; so they suggested us to use a static tool instead named 'BetterCodeHub' developed by the SIG itself.
The tool gave our project an overall score of 8/10, which is pretty good. However, after discussing with the client, we decided not to take it into consideration because static tools can be very inaccurate grading big projects like ours.

Appendix B - System's Component Diagram



Appendix C - Original Project Discription

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

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

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

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

Appendix D - Project Skills

Roberta Gismondi

During this year iteration of the Software Project (CSE2000) our team was picked to work for TU Deft Blockchain Lab and contribute to the Tribler/Dollynator project. Tribler is a tor-like peer-to-peer system that aims to provide its users with anonymous access to a network for content sharing purposes. In order to do so, Tribler makes use of Tribler exit-nodes, which are entry-points to the tor-net. In order to furnish a reliable and highly available suppliance of network entry-points Dollynator was developed. Dollynator is a self-replicating autonomous botnet of exit-nodes. The system exploit Reinforcement Learning techniques to pick and purchase Virtual Private Servers to be run as Tribler exit-nodes instances. Our contribution to the project mainly focused on ensuring a decentralised architecture for the botnet and to improve the algorithm responsible for VPS purchasing. Furthermore, the main concern of our team was to enable the latter algorithm to exploit information gathered from all the nodes of the network in order to improve the decision-making process.

In order to achieve such goals we decided to provide the system with a new, decentralised communication protocol for message exchange among nodes based on gossiping and to modify the pre-existing learning algorithm to adhere to a collective-learning model. Along with that, modifications to other Dollynator hard dependency systems were in order to guarantee a dependable deployment in real-world environment.

The team consists of five second year students, studying Computer Science and Engineering at TU Delft. The development process was organised such that each of us could initially focus on one of the three aforementioned sub-problems. Two of us implemented a reliable, decentralised messaging protocol, two of us put their efforts into modifying the current learning algorithm to use such protocol and consume information from the entire network during the learning process and one of us focused on fixing the dependency-related issues of the project. Nonetheless, we constantly kept a flexible division of tasks, with everyone willing to help with different problematics when necessary. The documentation-related matters were handled by a thorough approach consisting of division of the assignments (based on the weekly individual code-related workload) and peer-review.

During the project our team did not encounter major task-related conflicts, the division of the tasks was discussed thoroughly and established according to everyone's interests, knowledge, skills and ability. Everyone came form a different variant of our studies and have different expertise, yet everyone was willing to put effort into learning about new concepts and technologies. For this reason it was easy for us to find a compromise to accommodate everyone's need, ensuring a positive and formative experience during which every component of the team was given the chance to improve him/her-self both personally and

professionally. Being our team a well balanced group of students, consisting of trustworthy and competent members, and being our personal and professional relationships already stable before the beginning of the project, the decision-making process was easily held in a positive and constructive environment where each of us was able to research about the problem first, express their opinion afterwards and occasionally change their mind according to what had emerged during the discussion.

As we established our goals and expressed thoughts about our expectations about the project before-hand, we were able to avoid conflict of interests and misunderstandings during the development process. The division of the work in sub-problems and thus sub-groups made possible for us to reduce the number of intra-group conflicts.

Each decision was first discussed and deeply analysed in the context of the sub-group (between two or three people) and then presented to the entire team along with the list of pros and cons evaluated before-hand. Discussing matters in a smaller group first facilitated the communication and the fact that the dynamics between group members were already familiar (due to our pre-existing personal and professional relationship) helped us to promptly manage the occasional tension between teammates. The specific situation everyone is forced to face during the lockdown implied an additional amount of difficulties and potential problems. The impossibility of meet and discuss matter and the need to only communicate through internet could have lead to a more likely chance of under-performance of one or multiple members of the group. The organisation of our work into sub-problems tackled by a couple of people reduced the risk of social loafing as this is a phenomenon strictly correlated to the size of a group responsible for a specific task and could have been enhanced by the current world-wide situation.

Fortunately, the nature of most conflicts raised during our team-project was healthy; the nature of our work excluded the chance of a conflict for resources all together (as the work did not need any specific hardware nor had particular technical requirements), conflicts over power and competition were mitigated by the mutual esteem of group members, and the personal relationships between group members was one of friendly affection and reciprocal support before the start of the team-project. The dynamics of the team lead us to deal with the occasional conflict using a collaborative approach as much as possible. Some of us were less prone to assertiveness but willing to cooperate, resulting in an often accommodating behaviour towards the ones more assertive. The nature of this behaviour could be traced back to both an apparently wider knowledge of the topic by some of the group members and a natural inclination.

Furthermore, the current impossibility to have an actual confrontation (and the chance to more easily avoid discussing problematics) probably encouraged the aforementioned behaviour.

Nonetheless, when a conflict emerged, the team put as much effort as possible to allow everyone to express their own opinion, asking questions and encourag-

ing everyone to make a clear point. Avoidance was highly discouraged as every perceived problem was immediately discussed and never dismissed in the interest of time and quality of our work. Overall, each of us was able to balance out their cooperativeness and assertiveness to guarantee that each of the small conflicts occurred during the process were solved through compromise and occasionally full collaboration. This guaranteed a positive working environment, where each of the group member could grow personally and professionally. The quality of our team-work also reflected on the quality of our product and the pace to which it was developed.

Overall, the process loss of our team-project was a satisfactory percentage of out potential performance, this lead us to enjoy our collaboration and to benefit from the experience of working together and along with a third party (a client), developing a complex system that contributed to our personal and professional growth.

Giacomo Mazzola

The Software Project is getting to an end, therefore it is appropriate to make a reflection about it, retracing the path we have followed during these ten weeks.

It has been a challenging project, full of difficulties and unexpected problems; in particular we have had a rough start due to a lack of information provided by the client. In fact, he gave us a lot of freedom in designing the product, giving us very few requirements. This turned out to be helpful in a later stage, but made us struggling in the beginning because we did not have enough knowledge about the exiting project to make proper decisions. Moreover, the struggling was increased by the fact that we had to fix the current (broken) complicated and not well documented existing project, in particular Cloudomate.
After overcoming these difficulties we found our pace and we started to work as we were required to do. We did such a good job that we managed to bring to the midterm meeting a first functioning demo of the messaging protocol and the reinforcement learning algorithm.
The following weeks have passed very quickly, however we managed to finish our must and should haves and also to implement a could have.

After giving a general summary of the project, a discussion about intragroup organization and relations is required.
Despite the many encountered difficulties, I think that we managed to complete the project in a successful way thanks to our group organisation and devotion to it. From the beginning we divided the requirements into two main parts, the gossiping related parts and the reinforcement learning related ones. We did this because implementing the project from both parts simultaneously gave us the ability to show every week many different improvements to the client and the TA, and also, more important, since the two parts interact with each other, implementing them at the same time was the better way to make a proper design

and to adjust it whenever we found a problem in one of them.
Overall, I had a good relationship with the other members of the group; the only remark I want to point out is that I would have preferred to have more communication, review and support between each other.

As for the organization of the project itself, I think it was really well done. I liked that fact that we were constantly followed by a TA that helped us out in some situations and the coach, that with his experience gave us useful tips throughout the project. I found very useful the TW and the responsible CS components of the course. They are fundamental skills for an engineer and inserting them in a project is the best way to practise and learn them. However, I think that we were overloaded by the extra homeworks of these components, especially in the first two weeks of the project. The coordinators of the Software Project must consider that a week in not enough to get along with a huge and complicated project such as ours, therefore I would make less strict deadlines for the next years, especially in the first three weeks.

One additional difficulty that has to be discussed is the situation caused by the Corona virus. Due to the lock down, we were forced to face a unusual situation that caused us many troubles. First of all, we were forced to work from home without physically see each other. Everybody knows that physical meetings are easier to take and more time efficient then online meetings and that techniques like pair programming can be very useful in certain situations; unfortunately the virus didn't let us the possibility to opt for this techniques. Furthermore, we had, and we still have, an increased workload since the postponed exams were rescheduled to an overlapping period with the project. For these reason, we were not able to spend the required amount of time on the project.
Considering the given reasons, I would expected the coordinators of the project to adjust the workload and the requirements in such a way that let us be able to have a life instead of spending 60 hours per week working for TU Delft and, in the end, perform worse than usual because we had too many things to work on at the same time. I however understand that this is a situation that involves many parties and none of them want to give in to find a trade off that is better for the students.

Last topic I want to discuss is the educational side of the project.
I think that the project contributed in an important way to increase our knowledge and skills. We learned much about Python, Linux, distributed systems, bitcoin technology...
However, what makes me glad the most about this experience is the experience itself. We worked for a real client, in a real-world environment with professional people and professional tools. We also practised various techniques and procedures that are used in a real jobs, such as the literature study, the meeting with real clients and many more. We also practised our abilities to face and solve arising problems in the project and with the client.

To conclude, I must say that this was a very positive experience and, even though this was not my first experience of real job in the field of Computer Science and Engineering, it gave me a very valuable fund of experience.

Daan Goossens

I worked on a project called Dollynator, with my project team members: Tommaso Tofacchi, Giacomo Mazzola, Roberta Gismondi, Giorgio Acquati. I got into the group via Giacomo when I heard that they needed one last member in their group and I was basically the new guy in the group, the others all new each other already. I was the only Dutch guy in group of Italians, but I didn't feel there was any language barrier, because we all spoke English pretty well. I definitely went through the 5 stages of group development, except I didn't really experience the conflict stage so much, because we all reached a common goal what we wanted to improve on the Dollynator project. I couldn't have asked for better team members and had a good experience during this project.

Dollynators name comes from the combination of Dolly the sheep (the first cloned mammal) and the artificial intelligence of the Terminator movie. To understand what we improved on Dollynator, you first have to understand what Dollynator does. The lifecycle of Dollynator agent is to run as an Tribler exit-node on a VPS, earning mb-tokens (the currency in Tribler earned when running as an exit-node). The agents trades the mb-tokens on a decentralized Tribler market for bitcoin. When the agent has earned enough bitcoin, it buys another VPS where it self-replicates onto, and the cycle continues. The goal of Dollynator is to create a network of agents that run as Tribler exit-nodes on a VPS. Currently there was a reinforcement learning algorithm implemented for selecting the next VPS provider to buy and replicate onto. Our goal in this project was to get the old code working again first, because the code hasn't been touched in over 2 years and wasn't fully functioning anymore and after that improve this reinforcement learning algortihm, by implementing a gossip algorithm and implementing a reinforcement learning algorithm that works with gossipping. The gossip algorithm, is a state of the art algorithm, that hasn't been used yet in many real world applications. The gossip algorithm is an algorithm in which agents share their experiences with other nearby agents in the network. This way collective learning can be applied in a decentralized network of Dollynator nodes.

We didn't really have much task-related conflicts in the project. The only instance where there was a task-related conflict was during the implementation phase of the project, which was about how to implement the reinforcement learning algorithm in my sub-group that worked on that problem. It was a short lasting and the healthy kind of conflict. They were all about how to go about solving the problem at hand, and then we discuss the different solutions and come to a singular solution we all agreed with. I think this is a natural task-related conflict that should happen in any project for it to function prop-

erly. What helped is that we all agreed in the beginning what to improve about the project and we all had the same goal, namely improve the reinforcement learning algorithm by implementing a gossip algorithm. I also feel there was good (transparent) communication which especially made task-related conflicts get resolved quickly.

We didn't have any intragoup conflicts during the course of the project. All my project team members were really nice to work with, especially compared to other group projects I worked on, where there was a lot of social loafing going on, and some people were always late during meetings, or they never communicated. On a theoretical level there can be because of multiple reasons for having no intragroup conflicts.

Firstly, our group had a high entitativity, partly because our group for the most part already knew each other already, but also because as a group we decided our goal early on in the project.

Secondly, the group was transparent. We had regular meetings updating each other what we have done. And when the group was split up into subgroups, I kept everyone in my subgroup up to date what I was doing and they kept me up to date what they were doing. The communication was in my opinion the key part for making this group project work. This also made sure there wasn't any social loafing in our project.

Thirdly, we divided the project pretty well into everyones individual parts. This made it so everyone was responsible for something in the project. This also aided in the lack of social loafing in our group.

Lastly, we didn't have any of the 5 roots of intragoup conflicts (conflict and competition, conflict over resources, conflict over power, task and process conflict, personal conflicts) to begin with, this was caused because everyone had the same collective goal what to improve on the Dollynator project and we all were motivated. This also made sure there was good collaboration in the group. All in all this was a nice group to work with with a healthy working environment.

There weren't any unhealthy conflicts in the team, only some healthy conflicts, namely the task-related conflicts. Those were conflict were about how everyone had a different idea how to implement something. But those were dealt with explaining to each other why we wanted to implement something that way and convince each other of our own implementation. After discussing we always agreed to one solution which was the best for the project. As mentioned before we didn't have many conflicts, because the work environment was good, so there were only healthy conflicts, which are in my opinion necessary for a good project team. Even if there would be an unhealthy conflict, they would probably not get out of hand, because there was good communication and the conflict would go away quickly. Everyone also was motived, so there wasn't much social loafing going on in the team. This was, because in the end everyone had something specific they worked on, so everyone has their own contribution to the project and in the end everyone has done about the same amount of work. Everyone also trusted each other on the work they did.

In conclusion, I have good experiences working in this project team, with little conflicts, of which there was no unhealthy conflicts.

Giorgio Acquati

Our project team consists of five members, alphabetically listed as follows: Giorgio Acquati, Roberta Gismondi, Daan Goossens, Giacomo Mazzola and Tommaso Tofacchi. The five of us were assigned the task of continuing the development of Dollynator, a botnet of self-replicating agents that run on the Tribler distributed network. When we took over, the project was already well structured, and our main objective was to continue the development as started by the previous group that worked on it in order to bring the botnet to a state in which it is able to run in the real world, and not just in a simulated environment. Since the project was already in an advanced stage when we took over, our team had to spend a considerable amount of time getting acquainted with it. In fact, given the complex nature of the system that we were dealing with, understanding how it is structured and its behaviors posed a great challenge to us members of the group. During this initial period, we analyzed each component of the system under the perspective of understanding how one could modify its behavior or enhancing the features it provides. Moreover, we needed to reach a deep level of knowledge about the system before we would be able to negotiate functional requirements with our client. Fortunately, our client took over the role of an advisor more than a counterpart of negotiation. In fact, they limited themselves to pointing out to some rather general areas in which they wished their system were improved. While facilitating the negotiation phase, this certainly gave us the feeling of lacking guidance. After roughly three weeks spent on doing research on the matter, we finally were able to clearly state what our goals for this project was and to negotiate the requirements with our client.

As soon as our functional requirements were set, we immediately started working on implementing improvements to the system. At this point, our main concern was that we had spent a lot of time doing prep-work, and that we feared that we would not have enough time to satisfy the requirements that we had set. After all, a relatively long time had passed during which we had not written a single line of code. However, the time spent planning soon proved to be worth it. We immediately split our team of five in 2/3 sub-teams, each assigned with different complementary tasks. Personally, I was able to realize how splitting the work in smaller teams made us work in a very efficient way. We went on using this approach throughout the entire project, and I believe that it showed both its cons as it showed its pros. On the bright side, I believe that the team put good trust in individuals in solving problems related to a single task. This resulted in an unforeseen high efficiency, to the point where we completed our Must-Have requirements in just a few weeks. Moreover, this approach proved to be also highly effective at keeping the team members' level of stress low, since each team member was always concerned about small, atomic tasks. Not once during the implementation phase I was warried that we would not able to finish our project in time. However, looking back at that period

I believe that we moderately exceeded in splitting tasks. We were aware that communication between members of the group was set to play an important role, especially during the quarantine time during which we could not physically see each other. In fact, we always schedule 2/3 meetings every week, to make sure that all the sub-teams were always aligned and aware of the progress made by other members. However, these meetings had the tendency to be very short and I believe that we made the mistake of miscommunicating how challenging some of the tasks that we faced were. Moreover, I believe that in some cases we failed to communicate the fallacies of some of the solutions that we implanted to complete the more challenging parts of the tasks assigned. While not compromising the overall team progress, this miscommunication generated considerable difficulties in integrating different parts of the systems together to create the final demo for the project. To be fair, some of the more challenging problems faced in the integration part were not caused by miscommunication between team members, but rather to incompatibility of some components of our work with latest versions of the Tribler network. In fact, Tribler is an ongoing project that our system is based on, and it was modified during our work. In retrospective, I believe that the time spent planning was both sufficient and necessary to a satisfying outcome of the project. Moreover, our team was very efficient at splitting the work balance, but we slightly came in short regarding communication between team members. Overall, I am quite satisfied by how the project went from a technical point of view.

Regarding personal interactions between members of our group, I do not believe that any issue worth noting was encountered. In fact, we came from the advantageous situation of knowing each other very well before taking on the project. In my opinion, this almost completely cancelled many of the challenging aspects of cooperation, since we did not need to get acquainted with each other during the initial period of the project. This greatly facilitated communication and cooperation between members of the group. One thing worth noting is that four out of the five team members are of Italian nationality, while one of us is Dutch. I believe that Daan, our Dutch team member, would agree with me in saying that we made sure to never speak in our native language whenever he was present. From previous experience, I can tell how frustrating it can be working with people who discriminate you by speaking in a language that you do not understand, and I believe that we all put in the necessary amount of effort to avoid this situation.

As stated in the previous sections, I believe that this project group made for an overall positive experience. Thanks to previous team projects, we were aware of the challenges that working in group poses, and I believe that we successfully implemented some strategies to prevent conflicts within the team. I believe that knowing each other before the beginning of the project really helped eliminating the most common causes of miscomprehensions and non-task related conflicts. Regarding task-related conflicts, I believe that the only time during which we faced anything resembling a conflict was during the integration phase. These conflicts were mainly solved by openly discussing the issues with all the members of the team present. Moreover, after agreeing on a common solution,

we adopted the technique of "pair programming", which provided a way of closely collaborating.

Tommaso Tofacchi

I am Tommaso Tofacchi and I am part of the Dollynator's project team for 2020's Software Project iteration. Over the weeks that spanned the duration of the course, Dollynator has proven to be an as interesting as challenging project to tackle on a variety of aspects.
Firstly, the type of work which we were required to accomplish did not all fall under the same category, but could at least be seen as of two branches. Since we had to deal with a project that had remained untouched for about one and a half years, we immediately had to focus on bringing Dollynator back to a usable state, and did so by updating all of its interactions with external dependencies (another hot topic on the matter, as once again previously and adequately discussed) that had drastically changed in the meantime. Moving onwards, Dollynator's "restart and maintenance" kind of project emerged from the need to rewrite the codebase itself to make use of the latest Python Long Term Support version - currently 3.6 -, as the entire software was written in the by-now EOL Python 2.7. On the other hand, during the past ten-week period we had the opportunity to concentrate ourselves on Dollynator's new features implementation, covering a typology of project that is now closer to actual *innovation* compared to the previous, albeit still required *fixing-and-upgrading* work.
As the project is evidently composed by a high number of tasks, cooperation and efficiency as a group was needed more than ever. Working on Dollynator I collaborated in a group environment that was – by different means – already interconnected before the beginning of the 4th quarter. Giorgio and Giacomo, for instance, have been my roommates since the beginning of the 2019/2020 academic year; Roberta has been a friend of ours from the first days at TU Delft and Daan was Giacomo's project-mate for many of their assignments. We all knew that we were motivated student who would have all strived to achieve the best possible result given a situation and would not just be settled with accomplishing the bare minimum. Our collaboration for the duration of the Software Project ended up reflecting the expectations that came to our minds when teaming up in first place, which finally leads to an analysis on how effective our teamworking turned out to be.

During our first week together, we discussed a basic contract of cooperation to be signed by every member of the group. Other than establishing general rules regarding attendance to meetings and working methodology (for instance, Sprint-based iterations and adhering to Agile principles), we defined internal team roles for all team members. These roles, which concerned aspects of teamworking of a higher level than Dollynator's tasks fulfillment (i.e.: delegating a Scrum master, a Git repository maintainer, a secretary and deadline supervisors), were nonetheless welcomed by every member and remained functional for entirety of the project.

When dealing with proper development and its tasks subdivision, we tended to split in two groups of people, in order to tackle two diverse macro-areas of requirements. After effectively dividing the workload in this manner for the first weeks, we decided to stick to it, since we were satisfied with both the quality and quantity of our production. As a result, for what concerns new-features implementation Giorgio and Roberta focused on the gossiping algorithm, whilst Giacomo, Daan and I dealt with the reinforcement learning part. As the development proceeded, so did the need of merging the two groups' works together in a way that could allow for integration tests and general trial of Dollynator in a as close as possible scenario; facing the "merging of features" issue was surprisingly easy, since one or two person per group could join together and bring their acquired expertise to rapidly solve it.
In addition to this, all of the major decisions were taken with common agreement by consulting every member during a meeting. These strength points enabled us to avoid task-related issues throughout the entirety of the project, as anyone could have a constant overview of the project's development and express his/her opinion before any decision were to be made. Having this approach favoured the integration of every team element as a fundamental piece for Dollynator, as well as the correct spread of team mentality in which all of our individualities could positively emerge to finally benefit the project itself.

As a result, our intragroup relationships did not encounter negative bumps on the road. The role that each one of us covered in the team resulted in meetings that would often see suggestions on a main, commonly-shared solution emerge and be discussed. Such instances can be considered healthy discussions, as they all were task-related and relative to different approaches, rather than targeting power or personal issues.
The absence of unhealthy arguments has to be brought back to the previously analysed "individual in a team integration". Effective tasks management and reduction of encountered issues to the bare minimum is linked with every member's always high engagement and responsibility in the project. Having such team structure required constant effort from all of us, as at any time of development there was no *weak link* that could decide to suddenly back off without consequences on the others' work. This directly leads to active participation during meetings and avoidance of passive participation, which - for instance - makes it difficult for me to reflect on finding the classic figure of a *leader* amongst us.

Finally, every bit added up until this point makes it evident that no intervention technique was needed to be applied. Our natures and personalities (which, once again, were known to the others to a certain extent before deciding to team-up) blended extremely well right from the start of the project and continued to do so the more we got together. The proactive discussions that we had helped us giving ourselves the needed guidelines that we may not always find in the words of our client, and enriched every member with lines of thought that otherwise one would not have considered.
In fact, I believe that the suggestions for intervention techniques were actually well exploited in our team by making use of the fundamentals at the basis of

them (see the previous paragraphs), in order to prevent harmful behaviours to appear. I am pleased and very satisfied with how we have emerged as a group and, even more, as individuals that put their maximum effort for their own and their teammates success.

Appendix E - Info-sheet

- ***Title of the project***: Creating a Botnet of Self-Replicating Autonomous Organisms
- ***Name of the client organization***: Delft Blockchain Lab
- ***Team members***: Giorgio Acquati, Roberta Gismondi, Tommaso Tofacchi, Daan Goossens, Giacomo Mazzola
- ***Description:***
  - ***Challenge:*** the challenge was to bring forward the development of Dollynator, a botnet of self-replicating agents. These agents run Tribler exit nodes, which earns them Bitcoin currency. The earned currency is then spent to buy new Virtual Private Servers instances onto which the agents replicate, making the botnet expand.
  - ***Product:*** the created product consists of a botnet of agents that can share information in order to increase their chances of replicating. Specifically, agents make use of gossiping communication to share information about available VPS options onto which they could replicate. By sharing this information, the agents are able of collectively establishing which option is more convenient to use for replication.
  - ***Outlook:*** the project is mostly related to research efforts rather than real world needs, therefore it is hard to establish whether the system will be used by the client. The system will probably not be used by the client in its current state, but will (hopefully) probably pose as a solid foundation for implementing real-world deployable solutions based on it.