ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Noah Studach

Automatization of Internet Path Transparency Measurements

Semester Thesis SA-2018-28
June 2018 to October 2018

Tutor: Prof. Laurent Vanbever
Supervisor: Dr. Mirja Kühlewind
Supervisor: Brian Trammell
Supervisor: Tobias Bühler

Abstract

The growth of the Internet is reliant on faster and more scalable Internet transport protocols. Deploying new transport protocols is highly dependent on a transparent end-to-end path between any two servers on the Internet. This transparency can be challenged by outdated or misconfigured middleboxes. Misconfigured middleboxes may modify or drop packets they are designed to receive. Since middleboxes provide crucial network functionality, they are utilized very often. Path transparency can only be determined with active path measurements. PATHspider is a tool for such measurements but only provides one vantage point. Using multiple vantage points gives a better view of the situation and repeating the measurements periodically allows to monitor the progression over time. This calls for automation and we present you Sugar, a tool that lets you take active path transparency measurements with PATHspider from multiple vantage points with the help of a cloud infrastructure provider. Additionally, the results are directly uploaded to the Path Transparency Observatory from the MAMI project.

Contents

Chapter 1

Introduction

1.1 Motivation

Since its creation, the Internet has grown massively. Not only in size, but also in complexity. Today the Internet has over one billion hosts [6] and consists of multiple interconnected networks also known as AS's, that in turn can contain other networks within. Each AS manages its own infrastructure. Most of the time with the help of so-called middleboxes since they provide crucial in-network functionality. They not only keep large networks manageable and economically viable but are also essential in providing network security. Any device that transforms, inspects, filters or manipulates parts of the Internet Protocol for other reasons than packet forwarding can be called a middlebox. Firewalls and network address translators, NAT, are examples of middleboxes [18, 19]. However, the use of middleboxes can create problems if they are misconfigured or outdated. Because every network owner configures their middleboxes themselves, certain protocols or protocol features might be blocked by such a network. These middleboxes challenge the transparency of any path they are part of. An internet path is transparent if the protocol header arrives at its destination without any changes not part of packet forwarding. When rolling out new protocol features or even completely new protocols, path transparency is often a requirement for them to function properly [22].
Path transparency can only be determined with an active measurement, since the placement and functionality of middleboxes are unknown. To measure the transparency for a new protocol, packets with two different protocols, one new and one proven to work, have to be sent. Comparing their replies results in the path transparency. For this reason the Measurement and Architecture for a middleboxed Internet, MAMI [13], project created PATHspider a tool for active path transparency measurements [15]. The path a packet takes is dependent on time and the location on the sender and receiver. The representation of the measured transparency across the Internet should be as accurate as possible and therefore multiple measurements from various vantage points are needed. Infrastructure changes over time can be monitored by repeating those measurements in a periodic fashion. Automating this task will not only make it faster for all people currently taking measurements but also makes it more accessible to others. It is necessary to process the raw measurement data to draw meaningful conclusions. Therefore, the data should be provided to the MAMI Path Transparency Observatory, where it is added to a larger data set [14].

1.2 Goal

In this thesis the goal is to have a system in place that automatically takes internet path transparency measurements. Previous work has already created such a system called Spiderweb, but due to recent changes, it is now out of date. Yet, the system properties set in Spiderweb still apply for this work [28]:

- Multiple available measurement points
- Cost effective

- Compatible with PATHspider
- Automation: No human intervention required after the initial configuration
- Flexible regarding future extensions
- Automatically analyze measurements

We create Sugar, the system to replace Spiderweb. Like in Spiderweb, PATHspider performs the measurements while the MAMI PTO [14] serves as a file server to store and analyze the results. Therefore the goal is to automate taking PATHspider measurements. This includes but is not limited to:

1. Provide an up-to-date list of jobs for PATHspider
2. Execute PATHspider from multiple vantage points
3. Provide the results to the MAMI PTO
4. Keep the cost minimal

1.3 Overview

Chapter 2 presents different software and architectures, that are incorporated into the design of Sugar. For each piece, the purpose in Sugar and the way of interaction is explained. Next, Chapter 3 maps out the key challenges for the software design. Then the individual software components are presented and explained in more detail. Chapter 4 compares the implemented approach with the previously set desired system properties and presents a short benchmark of the final solution regarding time, resources and output. Chapter 5 concludes the thesis by giving a short summary of the outcome and an outlook to future extensions.

Chapter 2

Software Components

2.1 PATHspider

PATHspider is a tool that measures internet path transparency and will be used by Sugar to do the actual measurements. It performs an A/B comparison between two different transport protocols or different protocol extensions on the path to a given host. From this controlled experiment one can observe protocol-dependent connectivity problems or a differential treatment. The transparency is always measured for one specific protocol or protocol extension. The A/B testing is done using as baseline connection, usually TCP with no extensions and an experimental connection with a different protocol or additional extensions. PATHspider supports a number of plugins, such as Explicit Congestion Notification *ECN*, TCP Maximum Segment Size *MSS* or TCP Fast Open *TFO* but is also extendable with 3rd party plugins.

2.1.1 Data Format

Both input and output of PATHspider uses the newline delimited JSON, also called *ndjson*, format to encode the host information and observed path behavior. An input file may contain many hosts, where each host information is encoded in *ndjson* format on a separate line. Each line can, therefore, be referred to as a job.

- Each job needs at least an IP address in a *dip* field, but more information might be required depending on the plugin in use.
- The output of each job contains the original job, a computed path and a set of conditions observed for the path. An output is also called an observation.

The *ECN* plugin for example has a *connectivity*, a *negotiation* and *ipmark* condition. Each condition has a state for example connectivity has the following states:

- **works:** Both connections succeeded
- **broken:** Baseline connection succeeded where experimental connection failed
- **offline:** Both connections failed
- **transient:** Baseline connection failed where experimental connection succeeded

Each state of a condition of a plugin is notated as follows *plugin.condition.state*. This allows to quickly analyze the output regarding connectivity to get a feedback on the success of the measurements. If many connectivity states are offline there might be a problem with the server performing those measurements. All the conditions of the integrated plugins can be viewed in the PATHspider documentation [15].

2.1.2 Usage

PATHspider can be run via the command line by using *pspdr* followed by a command. At the time of writing the supported commands are *analyze, filter, measure, metadata, observe,* and *test*.

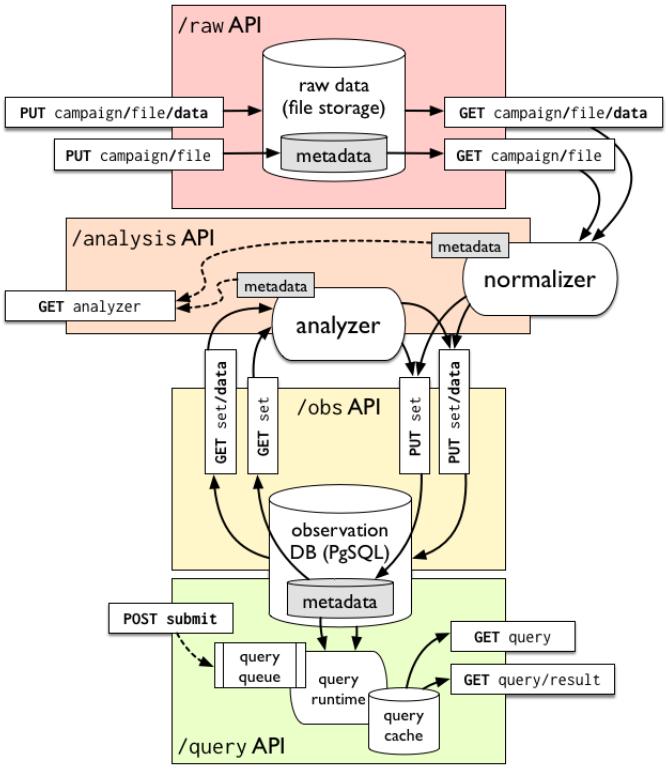


Figure 2.1: The PTO Architecture [26]

2.2 Hellfire and CANID

Meaningful measurements require on an up-to-date list of PATHspider jobs. Sugar uses Hellfire with a canid server to resolve the Amazone Alexa top 1 million hosts list [1] to provide a list of jobs. According to Section 2.1, PATHspider jobs must be formatted as JSON and contain at least a IP address. To convert one of the lists of topmost contacted host such as the Alexa top 1 million list to fit PATHspider we use Hellfire [12]. Hellfire is a tool to resolve such a list and get most recent IP addresses, domain and ranking. Hellfire can catch additional information by adding a daemon process such as the Caching Additional Network Information Daemon or CANID [27]. Both Hellfire and CANID are written in the programming language go [16].

2.3 Path Transparency Observatory

The path transparency observatory or PTO, in short, is a database from the MAMI project with the purpose of storing and analyzing PATHspider observations. The PTO consist of two data stores, the raw data storage, and the observation database. Any data that is uploaded is stored on the raw data storage, processed by the normalizer and analyzer, and ends up in the observation database where the more meaningful interconnected information is extracted. The PTO architecture is shown in Figure 2.1. While the observations and queries of the observation data are usually publicly available, the raw data is generally only accessible by the owner of the data. A raw data file is always associated with a file containing its metadata.

2.3.1 API

Access to the PTO is established over a RESTful API, which is a web-based and receives commands over an HTTP message. The RESTful API supports five methods: *POST*, *GET*, *PUT*, *PATCH* and *DELETE*. The *POST* method is used to create a new resource and upon a successful creation, it returns a link to the resource. A resource is read via the *GET* method, which returns the resource. *PUT* and *PATCH* are used to modify a resource where *PUT* replaces and

PATCH only modifies it. There is also the option to delete a resource using the *DELETE* method. Additional options can be added to each method such as extra header information containing authentication data or the content type of the related resource [5].

2.4 DigitalOcean

DigitalOcean, Inc. is a cloud infrastructure provider with data centers worldwide [17] and provides the remote servers to run PATHspider. DigitalOcean is used in Sugar to create many vantage points by creating multiple virtual machines in different locations. With DigitalOcean you can create virtual machines or droplets, that is how DigitalOcean calls them, on any of their servers distributed all around the globe. Each droplet is configurable regarding the operating system, hardware requirements, ssh access keys and so forth. The cost scales the more hardware is allocated. DigitalOcean uses a RESTful API similar to the PTO discussed in Section 2.3.1 [11].

2.5 SaltStack

SaltStack is an open source infrastructure management system. With SaltCloud, a part of SaltStack, virtual machines can be created and destroyed on a previously configured cloud provider. SaltCloud can also execute commands upon creation such as installing and executing programs. In Salt, such a set of instruction is called a State. A State can also run Python code. Their Reactor module gives the ability to execute any State when triggered from starting or finishing another State or an explicit call.

2.6 Spiderweb

Spiderweb is a first attempt to automate internet path transparency measurements. Spiderweb uses the Salt management system, see Section 2.5, to set up and command multiple servers. The servers perform the measurements and upload the data according to a predefined configuration. Spiderweb does most of the things requested in Section 1.2 but is not working due to recent changes on Digital Ocean, PATHspider, the PTO and SaltStack. To get Spiderweb working again, first, the interaction of Spiderweb and SaltStack must be adapted to a newer version of SaltStack. Next, the Digital Ocean configuration within SaltStack needs to be updated. Furthermore, the commands starting PATHspider need to change such that the newest PATHspider version can be used. The PTO RESTful API also needs changes to work properly or can be directly integrated into PATHspider itself.

2.7 Salt and Sugar

Since basically, every interface Spiderweb interacts with changed, updating Spiderweb needs nearly as much effort as starting from the scratch. Additionally, Spiderweb depends on SaltStack to function, which is not reliable enough. In theory, SaltStack is a sufficient standalone solution and handles the creation and destruction of servers, the execution of PATHspider and by adding some Python code, also the upload to the PTO. Creating this *Salt-Solution* failed, due to issues with the reactor. The problem is not solved yet, even with the help of the Salt Slack Workspace [10] and the Salt GitHub [2]. Due to the fact that on their GitHub repository new bug reports are opened nearly every day, SaltStack currently seems unreliable for long-term use. Therefore, we create a new management system only using Python. Chapter 3 shows the design of this system, Sugar, in detail.

Chapter 3

Software Design

3.1 Key Challenges

Before ever writing a single line of code it is always a good idea to evaluate the requirements and key challenges of the problem. Such an evaluation helps in designing a work plan and also marks the milestones of the project.
The quality of an internet path transparency measurements is equal to its representation of the current state of the Internet. The more vantage points are incorporated in the experiment and the more different addresses are probed the better is the measured state represented in the observed data. This leads to the first key challenge of using more than one machine to run the measurements on and the second that a measurement will take a long time to complete, about one to two days. In any case, the original desire of an automated measurement process is still the core of this project. It involves installing PATHspider, all its requirements and initiating the actual measurement but also uploading the results to the MAMI PTO. Any failures occurring during the runtime should be reported automatically for convenience. Obviously, all those challenges should be solved with minimal cost in mind. This leads to the following key challenges observed:

1. Multiple vantage points
2. Long run time
3. Automated setup, measurement, and upload
4. Report failures
5. Minimize cost

3.2 Software Parts

To fulfill the key challenges the high-level software design consist of five main parts:

1. Prepare PATHspider jobs
2. Create and destroy virtual machines
3. Set up measurement environment
4. Measure
5. Upload data

The first part is to prepare a list of up-to-date jobs, which are fed into PATHspider. The second part is the creation of virtual machines in different vantage points on the Internet. The third part sets up the measurement environment, such that PATHspider is ready to be executed. This includes installing PATHspider and all its dependencies but also fetching the input files. Then the measurement is taken on each machine. As a next part, the results from the measurements are uploaded to a file server, in this case, it is the PTO from Section 2.3. When all the previous parts are completed the virtual machines are destroyed.

3.3 PATHspider Extension

PATHspider is extended by an *upload* command, uploading a specified output to the PTO. Adding an *upload* command is desirable as it allows easier upload even for manual use. According to Section 2.3 the PTO requires that the metadata and the measurement results are sent to the server via an HTTP POST request. Such a POST request is easily sent with the python module pycurl [23]. Using similar modules such as requests [24] is also an option, but since pycurl is already part of PATHspider it does not add to its list of dependencies. Fist, Sugar extracts the metadata from the raw observation data. The metadata extraction is already part of PATHspider and accessible by the *metadata* command. We added the feature to enter custom metadata tags to this command and use it to enter the droplets location into the metadata. Next, Sugar sends the metadata to the PTO. The PTO specifies the location for the data by replying with an URL. This link is then used to upload the compressed measurement data.
All the tasks the PATHspider upload command needs to fulfill are listed below:

1. Create metadata with *metadata* command
2. Compress measurement data
3. Prevent overwriting existing files
4. Upload metadata
5. Receive URL and upload data file

3.4 Flowcharts

Section 3.5 contains flowcharts to better illustrate the inner working of the program code. Figure 3.1 helps reading them, by showing each flowcharts building block and its function. A double-sided arrow indicates, that the process calls another process and blocks until it responds.

Figure 3.1: The building blocks and their function as used in the flowcharts.

3.5 Sugar

Sugar is the connection between all the software and architectures presented in Chapter 2. It creates droplets, sets up the necessary environment and executes tasks on each droplet. The basic concept is displayed in Figure 3.2.

1. Sugar contacts DigitalOcean which creates the droplets
2. Sugar copies the required files to each droplet and starts PATHspider
3. The droplets send their observations to the PTO

Figure 3.3 shows a graphical representation of the high-level program structure. The process *Create Droplet* is found in Figure 3.5a in more detail. The block *Initiate Installation* is displayed in Figure 3.6a. The remote processes *Installation* and *Tasks* are presented in Figure 3.5b and 3.6b respectively. *Hellfire*, *Fetch* and *Split* are responsible for preparing the jobs, *Create* and *Destroy Droplets* interacts with *DigitalOcean* to allocate and release cloud resources, while the rest handles PATHspider measurement and data upload to the PTO.
We first explain both core parts of Sugar. The remote disconnected execution of any script in Section 3.5.1 and the interaction with DigitalOcean in Section 3.5.2. Next, in Section 3.5.3, we show the environment set up process to run Sugar and PATHspider. Section 3.5.4 explains how Sugar creates the job lists in a similar fashion as it handles PATHspider but using Hellfire instead.

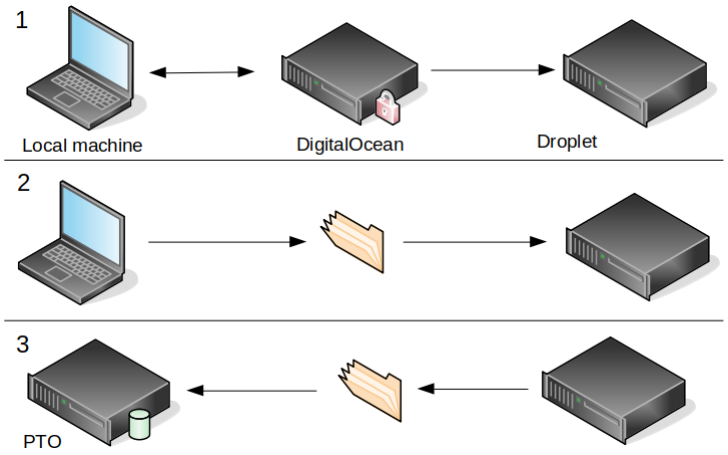


Figure 3.2: The basic concept of Sugar

Finally, Sugar has to manage all those tasks accordingly and Section 3.5.5 presents how this is achieved. Detailed explanations on installing, configuring and using Sugar are located in Appendix A. The source code is available on GitHub [4].

3.5.1 Remote Execution

For Sugar, the main challenge is the long time a path transparency measurement of reasonable size takes to execute. Under this circumstances, connections between servers are not reliable and a solution that is reasonably robust against disconnects is needed. Sugar is built such that after the environment is set up, the remote servers execute the tasks of measuring, uploading data, and destroying themselves without a connection to the local machine. This approach does not block the local machine during measurements and leaves it free for other use. Figure 3.4 shows how Sugar resolves this challenge. The remotely located script is started via an SSH connection using *setsid*. This stops the process from being terminating after SSH disconnects [20]. Sugar connects to each remote host in parallel using the Python module ParallelSSH [21]. Before we run the script Sugar reads the task information and IP address of each droplet, which is stored in the local configuration file. Then Sugar updates the tasks in each remote configuration file, so the remote instance knows what tasks to execute.

3.5.2 Droplet Management

Sugar uses remote virtual machines from DigitalOcean, a cloud service provider, to execute PATHspider from different locations throughout the Internet. Sugar creates and destroys these so-called droplets via DigitalOcean's RESTful web API. Figure 3.5a shows the program flow of creating a set of droplets. All droplet have the same specification, which is stored in the local configuration file in JSON format. Sugar serially creates one droplet in each region specified in the configuration. Sugar first reads the configuration and for each region, it generates a name and sends the specification with the name and the region to DigitalOcean via a POST request. DigitalOcean allocates the requested resource and answers with the droplets specification that also contains an ID. Sugar waits for the droplet to boot up, before refreshing the droplet specifications via the API. From this specification, Sugar extracts the IP address and saves Name, IP and ID in the configuration file for future use. If the Droplet is not ready after three attempts, Sugar deletes that droplet.

When a droplet is no longer in use it is destroyed. This keeps the cost low since DigitalOcean only bills you for the time the droplet exists. Sugar sends DELETE request to the API with the droplets ID contained within the URL. This process runs remotely if destroy is a selected task or locally when the IP address extraction fails multiple times.

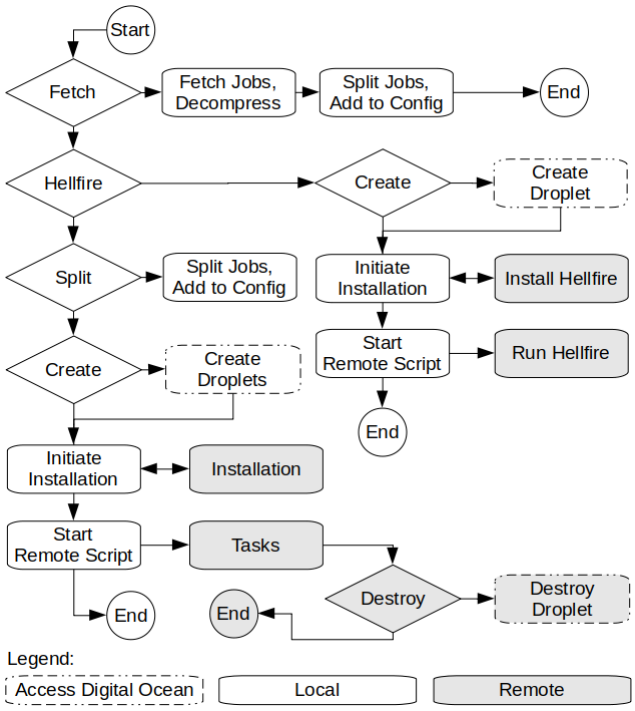


Figure 3.3: An abstract view of Sugars programming logic.

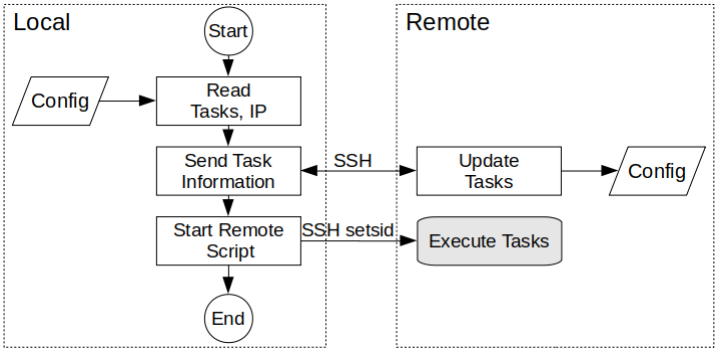
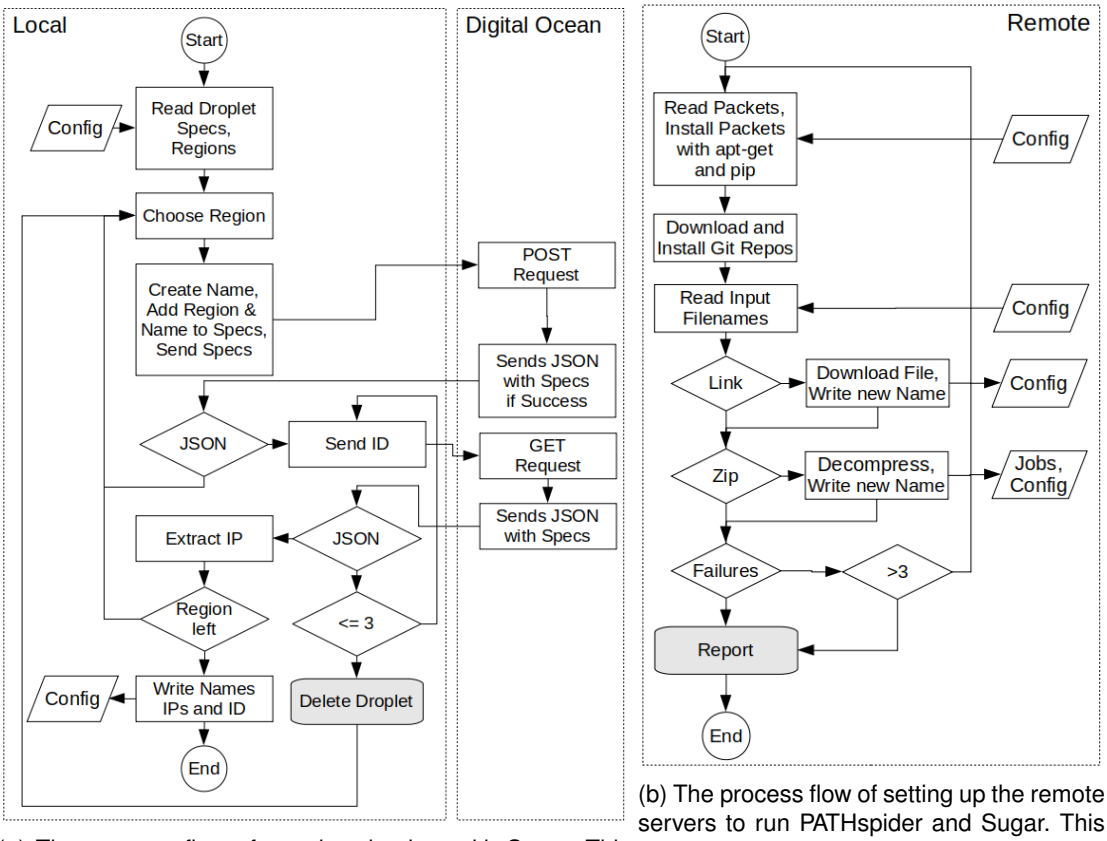


Figure 3.4: Sugar is run on all droplets in parallel. The command **setsid** stops terminating the process on disconnect.

3.5.3 Environment Setup

Not only PATHspider requires other programs to run but also Sugar relies on certain Python modules to execute its function. PATHspider and Sugar are both written in Python, which usually comes preinstalled on most UNIX systems. Sugar uses Debian 9 as the Operating System of its droplet. Debian 9 is chosen because it uses little amount of memory [9] and contains Python 3.5. Since Debian 9 comes with Python 3.5 we use the *subprocess* module to run Bash commands and install UNIX packets with *apt-get*. With the UNIX packets installed we have access to pip and therefore to all Python modules not in Python's standard library [8]. Some packets are not available to *apt-get*, such as PATHspider and *python-libtrace* [7]. Those are downloaded with *git* and installed via the command line. Additionally, PATHspider requires jobs to probe. The management system of Sugar presented in Section 3.5.5 compresses and copies the jobs to the remote hosts. Sugar also supports downloadable URL as input filenames. Therefore in a next step Sugar downloads any filenames that are URLs and decompresses all compressed files. Sugar retries any failed step up to two times before it stops and posts the progress on slack.



(a) The program flow of creating droplets with Sugar. This process runs in series for each droplet created.

(b) The process flow of setting up the remote servers to run PATHspider and Sugar. This process runs on all remote servers simultaneously.

Figure 3.5: The flowcharts of the *Create Droplets* and *Install* process.

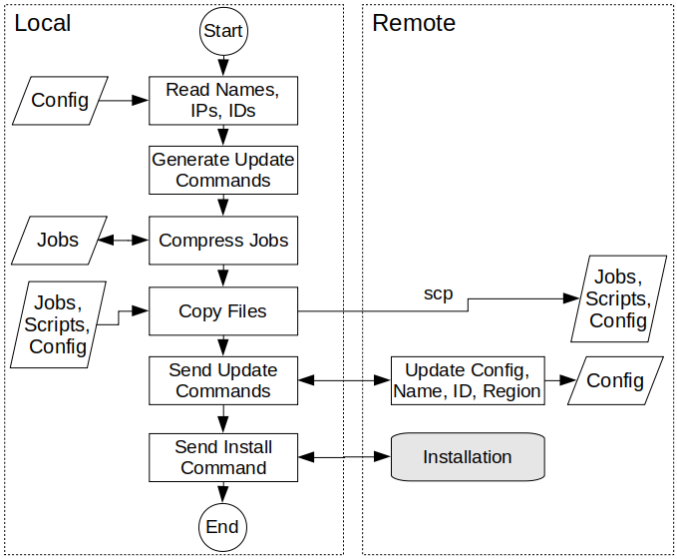
3.5.4 Preparing Input

With the help of Hellfire and CANID, Sugar delivers an up-to-date job list as input for PATHspider. Before running Hellfire, Sugar first creates a droplet and then sets up the environment for Hellfire by installing go and all other required packages. When Hellfire finishes, it informs the user via Slack that the jobs are ready. Via the - *-fetch* option Sugar downloads the jobs from the remote host, splits them into smaller chunks and adds their names to the configuration file. The splitting is required due to a bug occurring when using large input files with PATHspider [25]. By splitting them into multiple parts job file size is reduced to a point where even more workers can be used. This results in an overall shorter runtime but the output is fragmented into several parts. This makes reporting via slack and file handling less clean. Since the whole process of uploading and processing in the PTO is automated, this drawback is justified. The program flow of setting up the environment and running Hellfire is very close to the flows presented in Section 3.5.1 and 3.5.3 and therefore not explicitly illustrated. The one distinct difference between Hellfire and PATHspider is that Hellfire is written in Go, PATHspider in Python 3. Go has trouble finding its PATH when it is run over a remote connection. Luckily the subprocess module from Python allows specifying the environmental variables for each call.

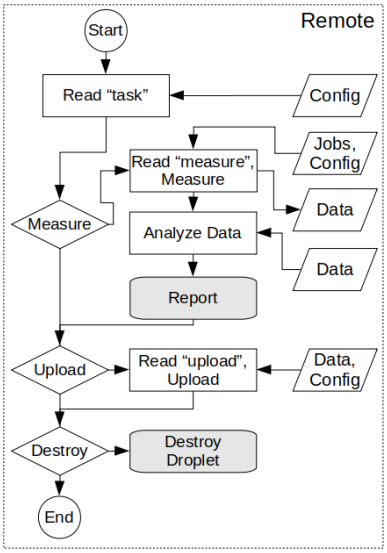
3.5.5 Management System

The management system combines the previously presented parts of Sugar into one working system. It is composed of two part, one running locally and one remotely. The locale instance is responsible for creating the droplets as seen in Section 3.5.2, copying all required files to each droplet and initiating the remote execution of Sugar. Whenever droplets are created Sugar copies part of its code to the droplets. It updates each remote configuration file with the droplets name, ID and region identifier before it initiates the setup of the remote environment. This process is displayed in Figure 3.6a. The remote part of Sugar is illustrated in Figure 3.6b and

manages the execution of the tasks on each droplet. The process is only displayed for PATHspider tasks. When running Hellfire, Sugar starts CANID and Hellfire, then waits for Hellfire to finish before informing the user over slack about the completion of the task.



(a) Local Sugar Instance prepares each droplet.



(b) Remote Sugar Instance managing PATHspider tasks.

Figure 3.6: This flowchart illustrates the local and remote parts of the Sugar management system.

Chapter 4

Discussion

4.1 Design

4.1.1 Drawbacks

During the short time of updating the configuration file and the still relatively short time of setting up the environment, the connection is kept alive. But for tasks that require more time such as executing PATHspider or Hellfire the connection is closed and the local instance of Sugar cannot determine when or if the remote execution stops. To solve this drawback, the remote script runs PATHspider and all its associated tasks independently and reports back to the user over slack. Only when preparing the PATHspider jobs the drawback becomes noticeable, since Sugar does not know if or when Hellfire finishes and can therefore not get the jobs list automatically. By adding the - *-fetch* option we minimize the effect but do not achieve complete automation. We accept this knowing that for each wave of measurements the jobs are only fetched once.

4.1.2 Different Cloud Service Providers

Currently, Sugar is only compliant with the cloud service provider DigitalOcean presented in Section 2.4. In Figure 3.3 presenting Sugar, all the interactions with DigitalOcean are marked. The switch to another cloud service provider with a similar RESTful API design is achievable with little change to the code of the program. On the contrary, if a cloud service provider with a completely different architecture and interface is chosen, adapting Sugar is not really applicable. Sugar uploads files on startup to the remote servers and also downloads and uploads files during the remote execution. Adapting Sugar to an infrastructure it is not designed for, such as the mobile testbed Monroe for example, where the node running the measurement is not accessible during runtime and the traffic of each node is limited, is probably more effort than to start from the beginning again.

4.2 Performance

4.2.1 Setup

We created nine droplets with two virtual cores and two gigabytes of memory. We deployed one on each server in the region Amsterdam, Frankfurt, London, Toronto, Singapore, Bangalore, and San Francisco and two on different servers in the region New York. We ran the PATHspider plugins *ECN*, *MSS*, and *DSCP* and used 60 workers. For convenience, the droplets were destroyed after each plugin finished uploading and rebuilt for the next set of measurements. One of the used configuration files is in Appendix A.4. The joblist was created on September 27th and the measurements ran between September 27th and October 5th.

4.2.2 Results

The *missing* column is roughly equivalent to the *pathspider.not_observed* condition. The condition *pathspider.not_observed* appears if the PATHspider observer does not observe any flow. Using too many workers overloads the observer and it cannot observe all the traffic. There were no observations missed with the *MSS* plugin instead the amount of observed offline connections is exceptionally high. Analyzing the data within the PTO gives a better understanding of the captured data. In this analysis, we focus on the percentage of missed observations since this is the only changeable variable. From the number of missing observations it is clear that fewer workers are to be used for future measurements. Any other observed condition is the measurement. But a high percentage of *offline* conditions observed can be an indication that the droplet does not work properly. This is however not under our control.

4.2.3 Duration

The duration of the measurements for each plugin and droplet is listed in Tables 4.1. The data making up the Tables 4.1 was extracted from Slack using a script [3] parsing the channel history. Therefore the order in which the droplets appear in the Table represents the order of appearance in the chat. This loosely correlates with the speed they processed the first input file. We observe that the duration does not seem to correlate with the initial speed. Further the speed does not depend on the amount of *offline* or *pathspider.not_observed* conditions. We conclude that the duration is highly dependent on the plugin used and the number of jobs but not on the droplet region.

4.2.4 Reliability

Sugar is reliable. The only failed large measurement since the fixing the problem of PATHspider allocating non-existent memory was with the *DSCP* plugin. Its analysis is shown in Table 4.1c. The failed droplets are identifiable on their number of total observations and are *sgp1*, *fra1*, and *nyc1*. The failure occurred in PATHspider during the measurements. Only future use gives enough context to determine the long-term reliability of Sugar.

4.2.5 Known Bugs

At the time of writing the automated execution of Hellfire as described in Section 3.5.4 does not work. After Sugar installs go, hellfire and CANID, it first runs CANID using the *Popen* function from the module subprocess in Python [8]. This runs canid as a child process without blocking the Sugar instance. Next, Hellfire is started using the *call* function of subprocess. This blocks Sugar until Hellfire finishes. After Hellfire finishes the user is informed via Slack. The script behaves as expected if started within an SSH connection. However, if the script is initiated from the local machine using the Parallel-SSH module [21], the execution of *call* kills the CANID process. This instantly terminates Hellfire because it does not find the specified canid process. Removing the *Popen* command from the script and starting CANID manually over SSH results in the script running as expected. When removing the *call* command from the script and starting CANID via the local Sugar instance, the script runs as expected, keeping the CANID process alive. At the moment the Hellfire process is started manually, the CANID process is terminated. Until the issue is resolved the script must be started via an SSH connection prepending the *setsid* flag to stop termination after the disconnect.

Region	nyc1	ams3	tor1	fra1
Duration	3	12	7	13
Online	897436	533896	749908	461100
Offline	639	871	78	1581
Missing	137659	500967	285748	573053
Total	1035734	1035734	1035734	1035734
Region	lon1	sgp1	nyc3	blr1
Duration	16	11	6	15
Online	336458	585450	778686	410934
Offline	7619	17	857	21
Missing	691657	450267	256191	624779
Total	1035734	1035734	1035734	1035734

(a) *MSS* Results. From Sept. 27th to Sept 28th.

Region	tor1	nyc3	sgp1	sfo2	blr1	ams3	lon1
Duration	21	21	23	21	23	19	21
Broken	1362	1489	1242	1484	1543	1946	1342
Offline	403457	404953	630600	457917	576354	245471	468366
Transient	1891	2222	2013	2024	2412	3288	2209
works	502980	627068	401879	574308	455425	785026	563816
Total	909690	1035732	1035734	1035733	1035734	1035731	1035733

(b) *ECN* Results. From Sept. 27th to Sept. 29th.

Region	sfo2	nyc3	lon1	nyc1	tor1
Duration	44	47	44	30	46
Broken	3	27	8	0	19
Offline	11	19	12	2	9
Works	777730	412283	780074	145232	668028
Missing	257990	623405	255640	7071	367676
Total	1035734	1035734	1035734	152305	1035732
Region	fra1	sgp1	ams3	blr1	
Duration	37	31	44	47	
Broken	2	2	2	15	
Offline	7	7	8	22	
Works	176001	32687	874533	457968	
Missing	125936	119610	161188	577729	
Total	301946	152306	1035731	1035734	

(c) *DSCP* Results. From Oct. 3rd to Oct. 5th.

Table 4.1: Analysis of measurement results provided over Slack.Duration rounded, in hours.

Chapter 5

Summary and Future Work

5.1 Summary

With a growing Internet and the desire for new better internet protocols, there is a need to measure internet path transparency. Current programs only execute one set of measurements. Multiple periodic measurements increase the quality of path transparency observations. Executing multiple measurements in parallel from different remote locations calls for automation. We created Sugar to fill this gap. A Python-based tool that creates and sets up remote servers via the cloud service provider DigitalOcean. It runs PATHspider simultaneously on all servers and uploads the output to the MAMI PTO. We satisfy the key challenges of having multiple vantage points by creating droplets in remote locations. A remote instance of Sugar runs disconnected from the local machine not caring about long runtimes. We automated the setup, measurement and upload process for PATHspider measurements with Sugar and a new command, *upload*, for PATHspider.Failures are reported over Slack, keeping the user up to date. Sugar destroys droplet after the upload, minimizing the cost. We have run a few measurements and their analysis has shown that we probably used too many workers and observed a large number on *pathspider.not_observed* observations. With exception of a single known bug Sugar runs and one known PATHspider failure Sugar runs reliably and we call it a success.

5.2 Future Work

Sugar achieves the task it was created for but like most things, it can be improved. The most crucial point is fixing the known bug presented in Section 4.2.5. The analyzer which checks PATHspider's output for observation defined in the plugin documentation can be improved by first making it generically usable for all plugins existing and yet to come. Additionally, if it distinguishes IPv4 and IPv6 traffic problem with a droplet might be diagnosed better. Another point for improvement is the reporting system. Slack works great but reading the output of each input file for each droplet individually is unnecessarily tedious. The script used to analyze the output presented in Section 4.2.2 helps but the reporting can be made more friendly towards a parser or sum all the numbers directly during runtime and provide a summary before terminating.

Appendix A

Sugar

A.1 Installation

Sugar is constructed as a Python module and available through GitHub [4]. To download Sugar use the git command or manually from GitHub. Sugar is installed by running *setup.py* with the argument *install*. To download and install Sugar run the following commands in the command window.

```
$ sudo git clone https://github.com/nstudach/sugar.git
$ cd sugar/
$ sudo python3 setup.py install
```

A.2 Configuration

Sugar must be configured with the according API authentication tokens for Slack, the PTO, DigitalOcean. The PTO token is only required to upload data to the PTO. With sugar comes an example configuration file and it is recommended to use it as a template. The part of the *example_config.json* file from Sugar, where the tokens are put is shown below.

```
{
    "slack":{
        "token": "Slack Chatroom Token",
        "channel": "#<<Slack Channel name>>"
    },
    "upload": {
        "campaign": "<<PTO Campaign name>>",
        "token": "<<PTO Token>>"
    },
   "provider":{
        "headers":{
            "Content-Type": "application/json",
            "Authorization": "Bearer <<DO API Token>>"
        },
        "regions": ["ams3", "blr1", "fra1", "lon1", "nyc1",
                    "nyc3", "sfo2", "sgp1", "tor1"]
    }, ...
```

The task Sugar executes are also configurable. All tasks are either set *true* to activate or *false* to disable them. The verbosity of the Slack messages posted is increased with *debug* set. To generate an up-to-date job list set *hellfire*. This disables all the other tasks except *debug*. With *hellfire* Sugar only uses the first region specified. The options *create* triggers Sugar to create droplets as presented in Section 3.5.2. The *install* option installs the PATHspider environment seen in Section 3.5.3 and 3.5.5. With *measure* a PATHspider measurement is started, *upload* uploads the output to the PTO and *destroy* destroys the droplets. Figure 3.6b from Section

3.5.5 illustrates them in more detail. Listed below is the section of the configuration file where the tasks are set.

```
...
    "task":{
        "debug": false,
        "hellfire": false,
        "install": true,
        "measure": true,
        "upload": false,
        "destroy": false
    }, ...
```

A.3 Usage

To use Sugar type *sugar* in the command window. The option *-h* gives information about available commands and their arguments:

```
usage: sugar [-h] [--plugin plugin] [--config file-location]
             [--key file-location] [--fetch filename n]
             [--split filename n]

Manage automated pathspider measurements

optional arguments:
  -h, --help            show this help message and exit
  --plugin plugin       Pathspider plugin to use
  --config file-location
                        Path to config file. Default is
                        configs/config.json
  --key file-location   Path to ssh authentication key
  --fetch filename n    Downloads input, executes --split
  --split filename n    Splits input into n sized parts, adds
                        them to config file
```

Sugar is launched using the command *sugar* followed by one of the optional arguments. The argument - *-config* provides Sugar the path a the configuration file. The arguments - *-plugin* and - *-key* overwrite the corresponding field in the configuration file. The - *-split* argument is used on local files. It splits the provided file of PATHspider jobs into *n*-sized parts and writes them into the specified configuration file. The - *-fetch* argument is discussed in Section 3.5.4. It fetches the job list from a remote droplet and executes - *-split*.
Whenever Sugar creates new droplets for PATHspider it adds them to the configuration file under *setup* as *host info*. If it creates a droplet for Hellfire it is saved under *hellfire* as *host info*. In case Sugar reads a configuration file with the respective *host info* set, it asks before overwriting the information. Choosing not to overwrite aborts the execution in case of *create*. In the case of *hellfire* it copies the scripts and executes the installation again.

A.4 Configuration File

An example of a configuration file used for measurements with the *DSCP* plugin. Only measurement relevant entries are shown.

```
{
    "droplet":{
        "size": "s-2vcpu-2gb",
        "image": "debian-9-x64",
        "ssh_keys": [<<removed>>],
        "backups": false,
        "ipv6": true,
        "user_data": null,
        "private_networking": null,
        "volumes": null
    },
    "provider":{
        "headers":{
            "Content-Type": "application/json",
            "Authorization": "Bearer <<removed>>"
        },
        "regions": ["ams3", "blr1", "fra1", "lon1", "nyc1",
                    "nyc3", "sfo2", "sgp1", "tor1"]
    },
    "measure":{
        "plugin": "dscp",
        "inputfile": [
           "https://www.dropbox.com/s/5oaoztja92uddc6/tpSept_1?dl=1",
           "https://www.dropbox.com/s/rcanl8p35cymy5d/tpSept_2?dl=1",
           "https://www.dropbox.com/s/1ncij9yzenttwgu/tpSept_3?dl=1",
           "https://www.dropbox.com/s/qn5k8tqeldnr38b/tpSept_4?dl=1",
           "https://www.dropbox.com/s/8cbta2ylviyqvso/tpSept_5?dl=1",
           "https://www.dropbox.com/s/8m7f734qt4ndtjg/tpSept_6?dl=1",
           "https://www.dropbox.com/s/x6j416ynuuymlzw/tpSept_7?dl=1",
           "https://www.dropbox.com/s/42p7k6ypfa35ecq/tpSept_8?dl=1"
            ],
        "outputfile": [],
        "workers": "60"
    },
    "task":{
        "debug": false,
        "hellfire": false,
        "install": true,
        "measure": true,
        "upload": true,
        "destroy": true}
}
```

Bibliography

[1] Alexa top 1 million global sites.
https://www.alexa.com/topsites.
Accessed: 06.10.2018.

[2] Git repository: Saltstack.
https://github.com/saltstack.
Accessed: 06.10.2018.

[3] Git repository: Slack-extractor.
https://github.com/nstudach/slack-extractor.
Accessed: 09.10.2018.

[4] Git repository: Sugar.
https://github.com/nstudach/sugar.
Accessed: 06.10.2018.

[5] Learn rest: A restful tutorial.
https://www.restapitutorial.com.
Accessed: 06.10.2018.

[6] Number of worldwide internet hosts in the domain name system.
https:/statista.com/statistics/264473/number-of-internet-hosts-in-the-domain-name-system/.
Accessed: 06.10.2018.

[7] Python module: Python-libtrace.
https://github.com/nevil-brownlee/python-libtrace.
Accessed: 06.10.2018.

[8] The python standard library.
https://docs.python.org/3.5/library/.
Accessed: 06.10.2018.

[9] Reddit: Linux distros ram consumption comparison.
https://www.reddit.com/r/linux/comments/5l39tz/linux_distros_ram_consumption_comparison_updated/.
Accessed: 06.10.2018.

[10] Salt slack workspace.
https://saltstackcommunity.herokuapp.com/.
Accessed: 06.10.2018.

[11] Website: Digitalocean.
https://www.digitalocean.com/.
Accessed: 06.10.2018.

[12] Website: Hellfire.
https://pathspider.net/hellfire/.
Accessed: 06.10.2018.

[13] Website: Mami project.
https://mami-project.eu/.
Accessed: 06.10.2018.

[14] Website: Path transparency observatory.
https://v3.pto.mami-project.eu/.
Accessed: 06.10.2018.

[15] Website: Pathspider.
https://pathspider.net.
Accessed: 06.10.2018.

[16] Website: The go programming language.
https://golang.org/.
Accessed: 06.10.2018.

[17] Wikipedia: Digitalocean.
https://en.wikipedia.org/wiki/DigitalOcean.
Accessed: 06.10.2018.

[18] Wikipedia: Middleboxes.
https://en.wikipedia.org/wiki/Middlebox.
Accessed: 06.10.2018.

[19] S. W. Brim and B. E. Carpenter. Middleboxes: Taxonomy and Issues. RFC 3234, Feb. 2002.

[20] G. Khera. Website: 5 ways to keep remote ssh sessions and processes running after disconnection.
https://tecmint.com/keep-remote-ssh-sessions-running-after-disconnection/.
Accessed: 06.10.2018.

[21] P. Kittenis. Python module: Parallel ssh.
https://pypi.org/project/parallel-ssh/.
Accessed: 06.10.2018.

[22] I. R. Learmonth, B. Trammell, M. Kuhlewind, and G. Fairhurst. Pathspider: A tool for active measurement of path transparency. In *Proceedings of the 2016 Applied Networking Research Workshop*, ANRW '16, pages 62–64, New York, NY, USA, 2016. ACM.

[23] O. Pudeyev. Python module: Pycurl.
https://pypi.org/project/pycurl/.
Accessed: 06.10.2018.

[24] K. Reitz. Python module: Requests.
http://docs.python-requests.org/en/master/.
Accessed: 06.10.2018.

[25] N. Studach. Git issue: Memory allocation.
https://github.com/mami-project/pathspider/issues/242.
Accessed: 06.10.2018.

[26] B. Trammell. Blog: An observatory for path transparency measurement.
https://blog.apnic.net/2017/10/23/observatory-path-transparency-measurement/.
Accessed: 06.10.2018.

[27] B. Trammell. Git repository: Canid.
https://github.com/britram/canid.
Accessed: 06.10.2018.

[28] P. D. Vaere. Continuous measurement of internet path transparency, Jan. 2017.