



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Piet De Vaere

Continuous Measurement of Internet Path Transparency

Semester Thesis SA-2016-68
October 2016 to January 2017

Tutor: Prof. Dr. Laurent Vanbever
Supervisor: Dr. Mirja Khlewind
Supervisor: Brian Trammell

Abstract

In this thesis, a framework for the continuous measurement of internet path transparency is developed and tested. This framework allows for the scheduled and automated orchestration of internet measurements using cloud computing infrastructure. Furthermore, integration with the *Path Transparency Observatory* (PTO) — a service providing automated measurement analysis — is provided. To test the measurement system, a study of internet path transparency for *Explicit Congestion Notification* (ECN) was performed. Two noteworthy results from this study are that ECN path dependency varies greatly over time, and that there is a strong link between ECN path dependency and nationwide internet censorship.

Contents

1	Introduction	5
1.1	Motivation	5
1.2	Goals	6
1.3	Overview	7
2	Background	9
2.1	PATHspider	9
2.2	Path Transparency Observatory (PTO)	10
2.3	SaltStack	10
2.4	Alexa Top Million	11
3	Design and Implementation	13
3.1	General Considerations	13
3.2	Architecture Selection	13
3.3	Campaign Configuration	15
3.4	PATHspider Modifications	16
3.5	SaltStack Implementation	17
3.5.1	SaltStack Module	17
3.5.2	SaltStack State Logic	17
4	First Application: ECN	21
4.1	Introduction and Background	21
4.1.1	Explicit Congestion Notification (ECN)	21
4.1.2	PATHspider ECN plugin	21
4.2	Implementation	22
4.2.1	PATHspider Modifications	22
4.2.2	Analyzer Modules	23
4.2.3	SpiderWeb Configuration	24
4.3	Results	27
4.3.1	State of ECN Path Transparency	27
4.3.2	ECN and Internet Censorship	29
5	Conclusion	33
6	Future Work	35
	Bibliography	37
A	List of Acronyms	39
B	Implementation Information	41
C	Results of the ECN Study	45

Chapter 1

Introduction

1.1 Motivation

Just like about everything else in this world, the internet started small. In its beginning, it consisted of only a handful of computers, managed by a close group of researchers. Routing tables were configured by hand, links were well known, and the number of protocols was limited. In fact, in its early days, the entire *Advanced Research Projects Agency Network* (ARPANET) — the technical foundation of the internet — could easily be drawn in a simple diagram, as is shown in Figure 1.1.

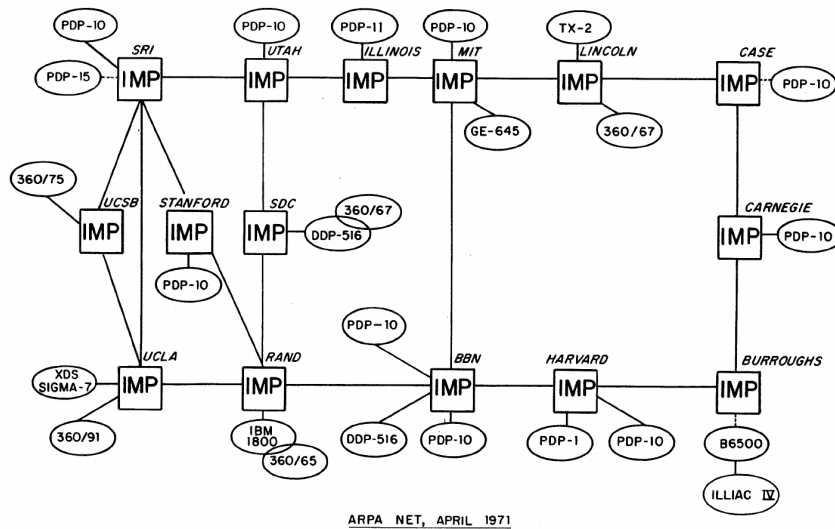


Figure 1.1: Logical map of ARPANET in April 1971. Ovals represent hosts. Squares represent interface message processors, the predecessor of the modern router. Image from the ARPANET completion report [1].

Much has changed since these early days. It is estimated that, at the time of this writing, 3.5 billion people are using the internet [2]. As a consequence of this massive scale-up, the internet has grown beyond the complexity that an individual human can understand. However, not only its large scale makes the internet so complex. Other factors, for example, the introduction of *middleboxes*, have also added to its intricacy. Middleboxes are loosely defined as any device that

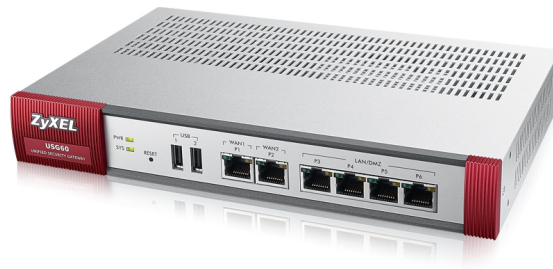


Figure 1.2: A Zyxel USG60 firewall, a typical example of a middlebox. Image from <http://zyxel.ch>

manipulates *Internet Protocol* (IP) packets for purposes different from packet forwarding [3]. For example, a firewall as shown in Figure 1.2 qualifies as a middlebox.

Furthermore, because of its decentralized nature, the internet is not homogeneous. For example, some internet middleboxes treat certain protocols differently than others. This can be intentional (e.g. for security purposes), or unintentional (e.g. because of misconfiguration or out of date software). In any case, this greatly increases the difficulty of rolling out a new protocol, protocol version or protocol extension. After all, it could be that there are middleboxes impeding the proper functioning of this new protocol.

Unfortunately, there is no better way to determine if this is the case than to perform large scale internet measurements. Therefore, the *measurement and architecture for a middleboxed internet* (MAMI) project has developed PATHspider [4], an active measurement tool that is designed to measure if datagrams get mangled while travelling over a network. When this is the case, it is said that the path the datagram was routed over is not *transparent*. To measure path transparency using PATHspider requires coordinated measurements from multiple locations. Furthermore, as internet path transparency is dynamic, periodic measurements have to be performed to keep track of how it evolves. When performed manually, setting up and running these measurements can be a tedious task.

This thesis contributes to the work the MAMI project has been doing by designing a system for automated, periodic, PATHspider measurements. Thus allowing for the continuous measurement of internet path transparency.

1.2 Goals

The main goal of this thesis is to facilitate the continuous measurement of internet path transparency. In order to accomplish this, an automated measurement framework will be designed. This framework should have the following properties:

1. It should be possible to perform measurements from a large variety of measurement points.
2. Measurement campaigns should be cost effective.
3. The framework should be compatible with PATHspider.
4. Measurements should be automated: after the initial configuration of a periodic measurement campaign, no human intervention should be necessary.
5. The framework should be flexible, and designed for future expandability.
6. The measurements should be automatically analysed.

1.3 Overview

This thesis consists of two main parts. First, a platform for automated PATHspider measurements is designed and implemented. Next, a demo study on internet path transparency for *Explicit Congestion Notification* (ECN) is conducted on this platform.

The remainder of this document is structured as follows. First, Chapter 2 provides background information on the most important components used in this work. Second, Chapter 3 describes the design of the measurement framework. Third, Chapter 4 documents the demo application of this framework. Finally, conclusions and future work are reported in Chapters 5 and 6, respectively.

Most code produced for this work is publicly released and can be accessed on <https://github.com/mami-project/>.

Chapter 2

Background

The work presented in this thesis uses a number of pre-existing components. The most important of these components are introduced in this chapter. When the reader is already familiar with some of these components, the corresponding sections can be skipped without problem.

2.1 PATHspider

PATHspider is a tool for measuring internet path transparency [4]. That is, it can be used to determine if the internet is transparent for a certain protocol, protocol extension, or protocol version. Support for different types of measurements is provided through user writeable plugins.

When provided with a list of measurement targets, PATHspider will attempt to connect to each of the specified hosts twice. First, a baseline is established by connecting to the host using a well known and universally supported protocol. Next, a second connection attempt is made using the protocol under test. This is illustrated in Figure 2.1. Based on the results of both connections, it can be determined if the protocol under test is fully functional or not.

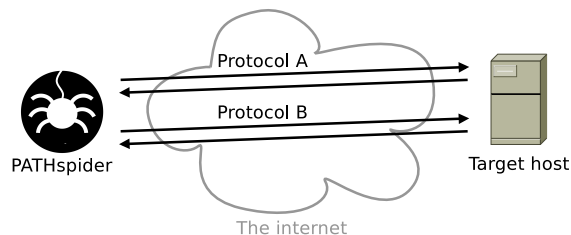


Figure 2.1: Schematic representation of a PATHspider measurement. First, PATHspider attempts to connect to the host using protocol A. Next, a connection using protocol B is attempted.

However, when the protocol under test fails, a single PATHspider instance can not determine whether the problem lies with the target host, or if something on the network path is obstructing the connection. Therefore, multiple measurements should be made, each from a different location on the internet, as shown in Figure 2.2. When some, but not all, of these measurements show that the protocol under test is functional, that must mean that the problem lies on the network. After all, it is confirmed that connectivity to the target host using the protocol under test is possible. Thus, by combining the results from these measurements, it can be determined if certain paths are impeding protocol functionality. When this is the case, we say that the protocol under tests exhibits *path dependency*.

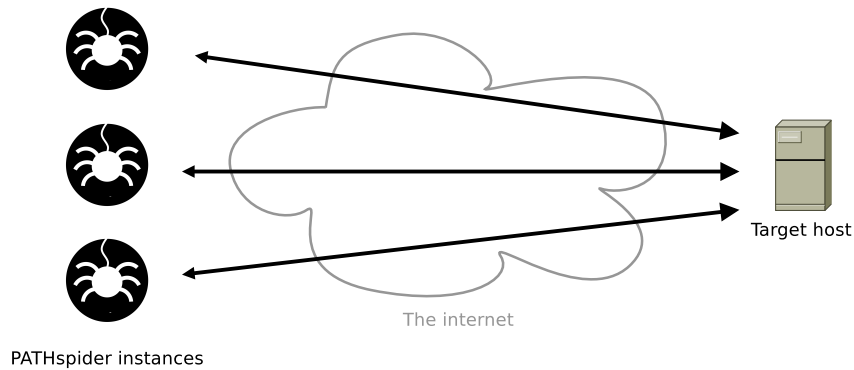


Figure 2.2: Schematic representation of a distributed PATHspider measurement. By performing PATHspider measurements from different locations on the internet, protocol path dependency can be determined.

2.2 Path Transparency Observatory (PTO)

In order to determine if there is path dependency, the results from multiple PATHspider measurements have to be combined and analysed. This is done by the *Path Transparency Observatory* (PTO) [5]. The PTO provides a number of services:

The upload service provides a RESTful *application programming interface* (API) that can be used to upload (raw) measurement data to the PTO. The upload service also allows for metadata to be uploaded alongside the measurement data.

The analyzer engine monitors the uploaded measurement data, and executes *analyzer modules* when new, relevant, data is available. These analysers are provided by the users of the PTO, and can be arbitrary Python scripts. They generate output in the form of *observations*, which are statements about conditions that have been observed on network paths.

The observation database stores all observations generated by analysers. Analysers can also query the database.

The web front end provides a user friendly interface to query the observation database. It is intended to be the main way of extracting data from the PTO. At the time of this writing, the web front end is under active development, and only an alpha version is available.

2.3 SaltStack

SaltStack, or Salt for short, is an open source remote execution and configuration management system [6]. It is targeted mainly at the management of cloud based computing infrastructure. A typical SaltStack setup consists of two types of nodes: a single *master* node, and multiple *minion* nodes. The master is usually an always up component. When a minion starts, it will connect to the master, and a bidirectional communication channel is established. The master can send *commands* to all connected minions. The minions execute these commands, and report back to the master using *events*. This is illustrated in Figure 2.3.

Besides this basic functionality, Saltstack provides many additional features. These include:

- A reactive state management system, allowing the master to automatically respond to events.

- A cloud provisioning module, allowing the master to create and destroy of minions.
- A secure file server, allowing the minions to retrieve files from the master.
- Both client and server side variable stores.

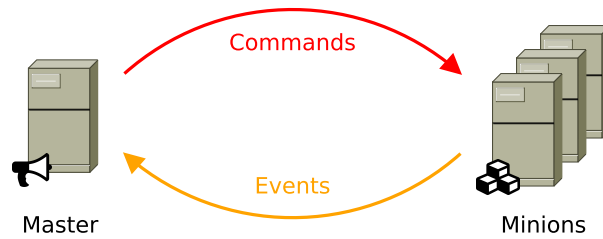


Figure 2.3: Schematic representation of a typical SaltStack infrastructure setup. A single, always up, master is connected to multiple minions. The master sends commands to the minions, and the minions generate events.

2.4 Alexa Top Million

The *Alexa top million* is an ordered list of the top million most popular websites. The list is sorted according to the Alexa traffic metrics [7], and has long been the *de facto* standard target list for internet measurement.

Although the list is made freely available by the Alexa Internet company, it has recently been announced that it will be replaced by a paid service in the near future. The *MAMI Public Target List* [8] strives to provide an open alternative to the measurement community, and can be used for future measurement campaigns.

Chapter 3

Design and Implementation

3.1 General Considerations

Based on the first two requirements put forward in Section 1.2, it was decided to base the measurement framework on cloud computing. This does not only allow for measuring from a variety of different locations (Requirement 1), but is also cost effective (Requirement 2). The later is true because measurement nodes can be created when they are needed, and destroyed once the measurement is complete. This means that the user is only billed for the measurement infrastructure while the measurements are taking place. In order to be compatible with PATHspider (Requirement 3), Linux based servers should be used.

The automatic analysis of the measurements (Requirement 6), can be accomplished by using the *Path Transparency Observatory* (PTO) (See Section 2.2). As the PTO was designed specifically for this purpose, this was a natural decision.

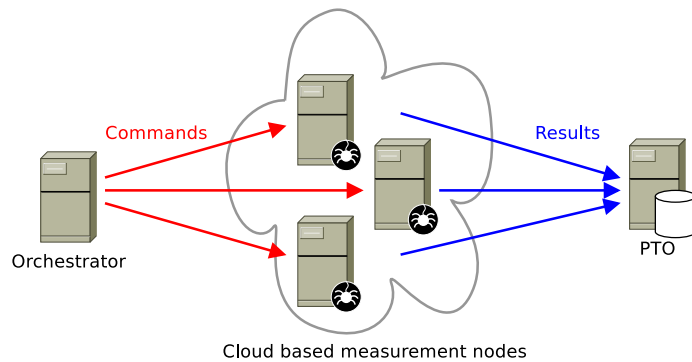
For the two remaining requirements — automated measurements and flexibility —, there is no readily available solution. Therefore, one will have to be designed. The requirement to have automated periodic measurements (Requirement 4), suggests that an orchestration component will be needed. As this is not commonly provided as a service, this component will have to be implemented on a self managed server. As this server should be able to start measurements at any time, it must be always up. The last remaining requirement is flexibility (Requirement 5). Because this does not easily translate into a single design decision, it will be used as a decision criterion throughout the entire design process.

3.2 Architecture Selection

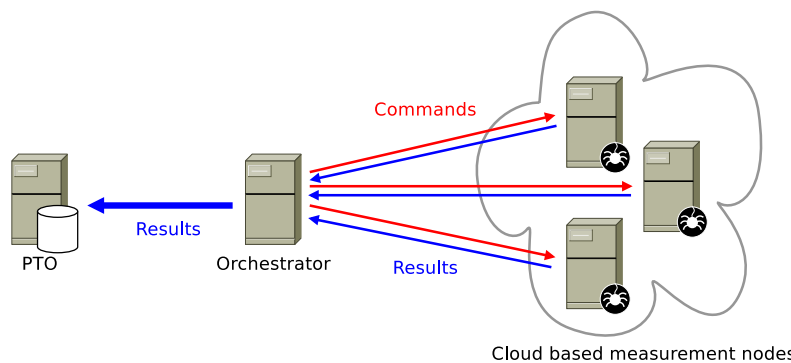
From the previous section, it follows that at least three components will be needed: an always up orchestration server, cloud based measurement nodes and the PTO. These components can be organised in a number different ways, the two most sensible of which are shown in Figure 3.1.

The setup in Figure 3.1(a) is the simplest, because it eliminates the need for result handling on the orchestrator. Furthermore, uploading the results directly to the PTO also puts less load on the orchestrator. On the other hand, by letting all results pass through the orchestrator — as in Figure 3.1(b) —, it becomes possible to do a first analysis on the data before passing it on to the PTO.

Because it is the philosophy behind the PTO that as much raw data as possible should be stored, and because it adds the least amount of complexity, the architecture in Figure 3.1(a) will be used.



(a) The orchestrator sends measurement commands to the measurement nodes. The measurement nodes send their results directly to the PTO.



(b) The orchestrator sends measurement commands to the measurement nodes. It also receives the measurement results, and forwards these to the PTO.

Figure 3.1: Schematic representations of two possible measurement architectures.

This can be done without compromising in flexibility, as the results are uploaded to the PTO using a RESTful API. Thus, by running a command compatible API on the orchestrator, and pointing the measurements nodes to it, the architecture from Figure 3.1(a) can effectively be transformed into the architecture in Figure 3.1(b).

Now that a decision has been made on the general architectural structure that will be used, the measurement system can be developed in more detail. To keep costs low, the cloud based measurement nodes should only be online while a measurement is taking place. Thus, the orchestrator should not only be able to send commands to the measurement nodes, but should also be able to *create* them. Because creating and setting up cloud servers is a complex process, it was decided to use SaltStack, a cloud management framework (see Section 2.3). This provides a number of advantages:

- Cloud server creation is largely simplified. Creating and connecting to a cloud server can be done with a single command.
- Cloud server creation is largely abstracted. SaltStack has out-of-the-box support for more than twenty cloud providers, and after initial configuration, all provider specific settings are abstracted away.
- A secure and bidirectional communication channel is automatically created between the orchestrator and the measurement nodes.
- Integration with a large number of tools (e.g. Slack [9]) is provided.
- Configuration can be done using a high level and modular system, eliminating the need for

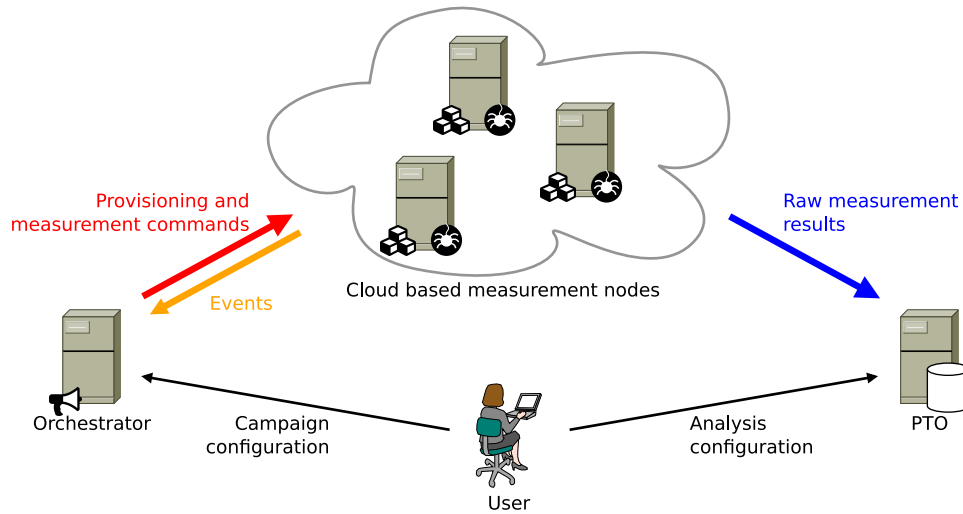


Figure 3.2: Schematic representations of the complete measurement architecture.

long and unwieldy (shell) scripts.

Because the orchestrator will create and configure all measurement nodes, there are only two elements of the measurement infrastructure the user should interact with. These are the orchestrator (to configure measurement campaigns), and the PTO (to configure measurement analysers).

Adding the SaltStack and the user to the architecture in Figure 3.1(a), results in the final architecture shown in Figure 3.2.

3.3 Campaign Configuration

Path transparency measurements are typically organized in campaigns. Each campaign consists of the following:

A PATHspider configuration determining the kind of measurement that will be made.

A target list specifying which communication endpoints will be tested.

A set of measurement locations determining where the measurements will take place from.

A time schedule defining when the measurements runs will take place.

To allow the user to efficiently configure and schedule measurement campaigns, *SpiderWeb* was created. SpiderWeb is a tool that can read out campaign configuration files (see below), and can instruct SaltStack to perform a measurement run. By using a time-based job scheduler (e.g. Cron [10]), periodic measurements can be performed.

A campaign configuration file should be *JavaScript object notation* (JSON) formatted, and should contain the properties specified in Table B.2. Listing 3.1 shows an example configuration file. What follows is a line by line explanation of this example.

The first property sets the name of the measurement campaign.

```

1  {
2    "campaign": "modern-times",

```

Next the argument string that will be passed to PATHspider is specified. Before this string is actually passed on to PATHspider, it will be processed by the PATHspider execution module (see Section 3.5.1). There, the Python `format()` function will be applied to it. This will substitute all bracketed expressions by the value stored under that key in the SaltStack local variable store. All properties of the campaign configuration file are also made available in the local variable store, so they can be used as well. In this example, `{campaign}` represents the campaign name specified on line 2, `{id}` represents the hostname of the measurement node and `{cloud_location}` the physical location of the measurement node.

```
3     "pathspider_args": "--pto-campaign {campaign} --pto-url
      'https://observatory.com/hdfs' --pto-api-key 1337
      --pto-filename {id} --pto-location {cloud_location} ecn",
```

On line 4, the location of the PATHspider input file is specified. This is the location of the file on the orchestrator. This file will automatically be copied to all measurement nodes.

```
4     "input_file": "/target_list.csv",
```

The `minions` property specifies what measurement nodes should be used. The key of every entry is a SaltStack cloud profile (cloud profiles are a mechanism used to predefine cloud servers), and the value is the number of measurement nodes to be spawned. In this example, 3 nodes will be spawned of each profile. Each of these nodes will independently measure every endpoint in the input file.

```
5     "minions":
6         {"do-nyc2-2048": 3,
7          "do-sfo1-2048": 3,
8          "do-ams3-2048": 3
9         },
```

The final property specifies what measurement nodes should do when the measurement is completed, and the results uploaded to the PTO. The current options are `destroy` and `None`. The former will cause the node to issue a request to be destroyed, the later will do nothing.

```
10    "when_done": "destroy"
11 }
```

3.4 PATHspider Modifications

When a measurement is completed, the measurement nodes must upload their results to the PTO. Because the ability to upload to the PTO is useful for a broad range of PATHspider measurements, it was decided to add this functionality to PATHspider itself. In order to accomplish this, PATHspider was modified so that it does not only writes measurement results to the regular output file, but also writes the results to a temporary file that is compressed using `bzip2`. Once the measurement is completed, PATHspider will upload this file to the PTO using its RESTful API. A separate, `bzip2` compressed file is used, because PATHspider outputs can be gibibytes in size, but are also highly compressible. Space savings of 95 % are not uncommon. Together with every upload, PATHspider also uploads metadata. This metadata includes: the measurement tool used, the PATHspider version, the dataformat, the measurement campaign, the location of the measurement server, and the timespan of the measurement. The former three entries tell the observatory how to interpret the data. The later three are used provide extra context to the analyzer modules, and to determine which uploads should be analyzed together.

The upload functionality is provided to the PATHspider user using a number of command line flags to set the location of the PTO, the PTO credentials, the measurement campaign, the measurement

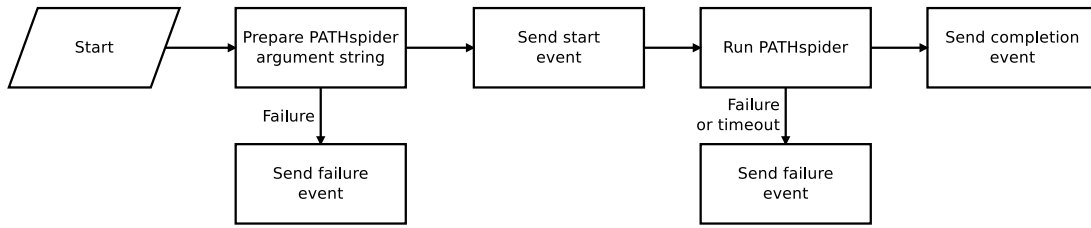


Figure 3.3: Flowchart of the PATHspider SaltStack execution module program flow.

location and the target filename. A full overview of the options is given in Table B.1. All options can also be specified in a JSON formatted configuration file. Settings passed via the command line will overwrite the values in the configuration file.

3.5 SaltStack Implementation

As SaltStack will be used to orchestrate the measurements, an interface between SaltStack and PATHspider must be designed. This interface is documented in Section 3.5.1. Next, Section 3.5.2 describes the orchestration logic.

3.5.1 SaltStack Module

One of the ways in which SaltStack supports remote code execution, is through the use of *execution modules*. These modules are Python scripts that are executed on the SaltStack minions, and have access to the local variable stores, the SaltStack event bus and all other SaltStack components.

To facilitate automated PATHspider measurements, a custom execution module was written. This module does three things: it provides a wrapper for the invocation of PATHspider, it preprocesses the PATHspider argument string, and it fires events informing the orchestrator about the state of the measurement. A simplified version of the control flow of the execution module is shown in Figure 3.3.

SaltStack events are identified by a *tag*, and contain a key-value store. Typically, this tag is a hierarchical structure containing the most important event information. The PATHspider execution module uses three types of tags. They are listed in Table B.3. Each of these tags is appended with the hostname of the measurement node generating the event. The content of each event also follows a predefined scheme. The fields in this scheme are listed in Table B.4, and Listing 3.2 shows an example event.

3.5.2 SaltStack State Logic

Now that SaltStack can interface PATHspider, the measurement state logic must be configured. This is done using the SaltStack *reactor* module. The reactor module can be used to configure the orchestrator to automatically execute commands when certain events are received. The orchestrator can then instruct the measurement servers to execute a new set of tasks. Figure 3.4 shows the orchestration steps necessary to set up a measurement node and run a measurement.

The first step of this process is to request the cloud service provider to create a new measurement server. Using SaltStack's *cloud* module, this step is combined with the installation of the SaltStack minion. Thus, with a single command, a new measurement server is created and connected

to the management infrastructure. Once the measurement server is ready, it will generate a `server started` event. The orchestrator will react to this event by sending the startup tasks list to the measurement server. The measurement server will use the `task` variable in its local variable store to determine which tasks from the task list should be executed. When using SpiderWeb, the `task` variable is automatically set to `"pathspider"`, causing the measurement server to fetch the PATHspider inputfile, and to install and execute PATHspider itself. SaltStack will use the custom PATHspider execution module (see Section 3.5.1) to call the PATHspider command, and just before the PATHspider command is invoked, this execution module will fire a `spider started` event. The orchestrator will respond to this event by instructing the measurement node to advertise the start of the measurement on Slack.

When the measurement is completed, and the results are uploaded to the PTO, the PATHspider execution module will fire a `spider completed` event. The orchestrator will react to this event by commanding the measurement server to send a Slack message. Once the Slack message is send, the measurement server will issue a deletion request. This is done in the form of an event. When the orchestrator receives this event, it will send a request to the cloud provider to destroy the measurement node.

A number of things should be noted about this procedure:

- Most work is done on the measurement servers. This is intentional to keep the load on the orchestrator low.
- Depending on the application needs, simple or complex tasks can be linked to events. In the current implementation, some tasks are so simple that they might as well be executed on the orchestrator (e.g. posting a message to Slack). However, to keep the system expandable, it was chosen to move all work to the measurement servers.
- Only the orchestrator interacts with the cloud providers. This allows for better management of the cloud service provider API-keys.

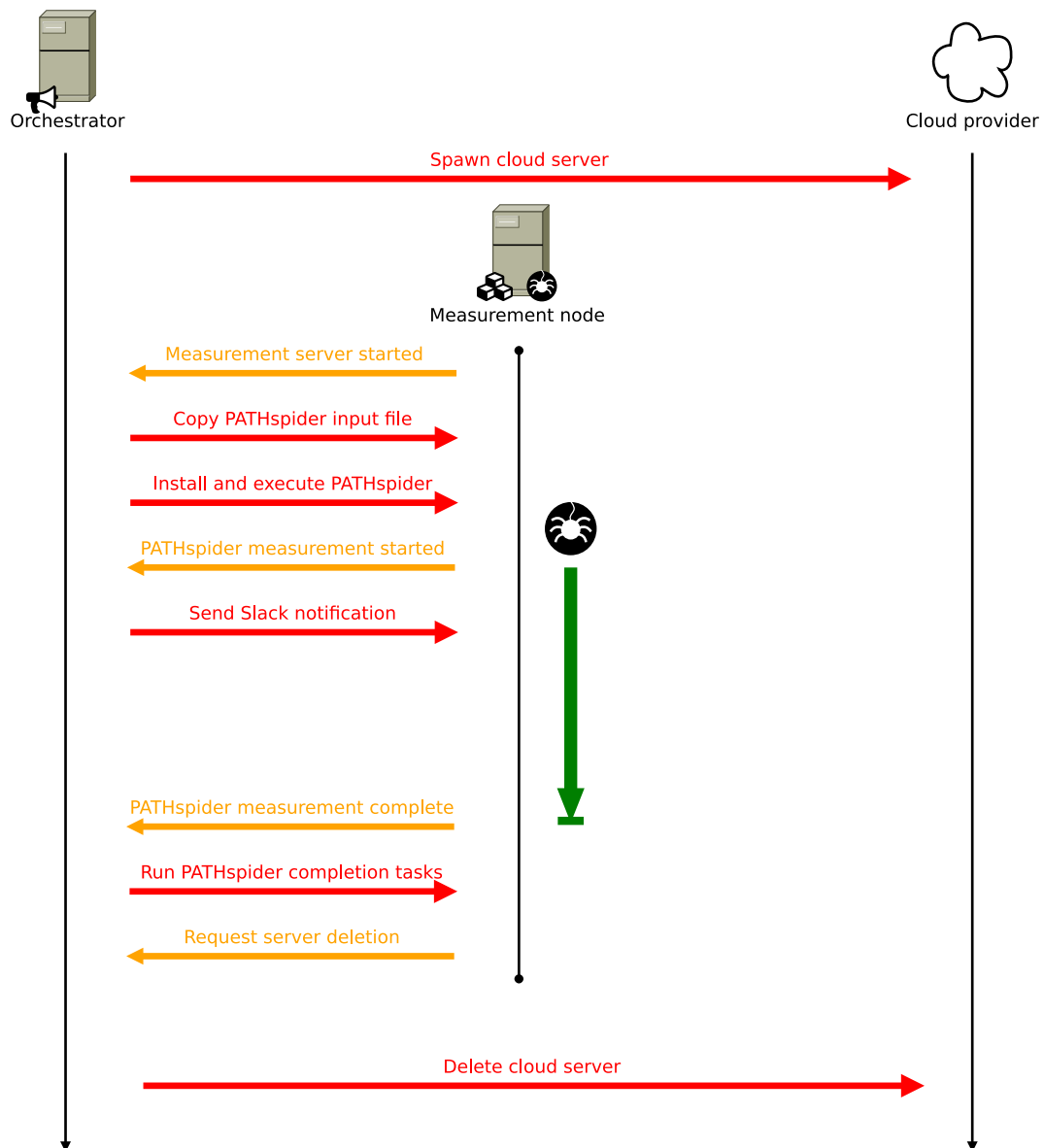


Figure 3.4: Timeline showing the orchestration steps involved in performing a PATHspider measurement using the SaltStack infrastructure.

Listing 3.1: An example SpiderWeb configuration file.

```
1 {
2   "campaign": "modern-times",
3   "pathspider_args": "-i eth0 -w50 --pto-campaign {campaign}
4     --pto-url 'https://observatory.com/hdfs' --pto-api-key 1337
5     --pto-filename {id} --pto-location {cloud-location} ecn",
6   "input_file": "/target_list.csv",
7   "minions":
8     {"do-nyc2-2048": 3,
9      "do-sfo1-2048": 3,
10     "do-ams3-2048": 3
11    },
12   "when_done": "destroy"
13 }
```

Listing 3.2: An example event generated by the PATHspider execution module. This specific event is generated if PATHspider exceeds its maximum execution time.

```
1 /mami/pathspider/spider/failed/BEsjFVRfHN-do-sgp1-2048-1
2 {
3   'finished': True,
4   'success': False,
5   'error': "Pathspider timeout",
6   'message': None
7 }
```

Chapter 4

First Application: ECN

Based on the generic PATHspider measurement framework presented in Chapter 3, a first application will be developed: the automated and continuous measurement of internet path transparency for ECN. The example application will perform measurements over the Alexa top million on a weekly basis.

4.1 Introduction and Background

4.1.1 Explicit Congestion Notification (ECN)

ECN is a protocol that allows routers to signal to hosts that network congestion is imminent, so that hosts can proactively lower their data transmission rate [11]. It is implemented as an extension to both *transmission control protocol* (TCP) and IP, and is a fully optional feature.

During the TCP handshake, the use of ECN is negotiated as follows: in the SYN segment, an ECN capable host sets both the *ECN-echo* (ECE) and *congestion window reduced* (CWR) flags of the TCP header. When the receiving host responds with a SYN-ACK segment with the ECE flag set, but the CWR flag not set, ECN is successfully negotiated.

During an ECN-enabled TCP session, all packets are marked with an *ECN capable transport* (ECT) codepoint in the IP header. When a router on the path of the packet expects congestion to occur, it may change the ECT codepoint to a *congestion encountered* (CE) codepoint. When a host receives a packet with the CE codepoint set on a socket, it will set the ECE flag in its next transmission over that socket. The ECE flag is repeated until a segment with the CWR flag is received.

Previous work has shown that some internet middleboxes mangle segments or packets with the ECN flags or codepoints set [12, 13]. As a result of this, two ECN capable hosts might be unable to establish a connection, or their connection might be otherwise disturbed. This shows the importance of ECN internet path dependency, and the relevance of charting it.

4.1.2 PATHspider ECN plugin

PATHspider ships with a plugin for measuring ECN path dependency. The measurement is performed as explained in Section 2.1. First, a baseline measurement is made by attempting to open a regular TCP connection with the target host. Next, a TCP connection attempt with ECN negotiation is made. Once both connection attempts are made, a local, first analysis of the results

is performed. More concretely, for each measurement target, it is tested if certain *conditions* are observed. If they are, they are appended to the target’s output record.

The conditions that are tested for are listed below. In the following, ECN is said to be enabled whenever its use was negotiated during the TCP handshake, regardless of whether or not that negotiation was successful. For implementation purposes, the conditions are formalized using a hierarchical naming scheme. These formal names can be found in Table B.5.

- The ECN *connectivity* condition provides information about the success of the PATHspider connection attempts. Four different types of connectivity are defined. Firstly, ECN connectivity is said to be *working* if it was possible to connect to a measurement target with and without ECN enabled. Secondly, the connectivity is marked as *broken* if the connection could only be made without ECN enabled. Thirdly, if the connection was only successful with ECN enabled, connectivity is said to be *transient*. Finally, if both connection attempts failed, the connectivity is marked as *offline*.
- The ECN *negotiation* condition provides information on whether or not ECN could be successfully negotiated.
- The ECN *ECT* condition provides information on whether or not an ECT codepoint was received.
- The ECN *CE* condition provides information on whether or not a CE codepoint was received.

4.2 Implementation

4.2.1 PATHspider Modifications

During testing it was found that PATHspider and its ECN plugin had multiple bugs and shortcomings that prevented the accurate measurement of path transparency.

These mostly include general logic and concurrency errors, that — although very time consuming to resolve — are of limited interest to this thesis. Therefore, this section will only describe two modifications made: the updating of PATHspider’s build-in *Domain Name System* (DNS) resolver, and the addition of *Hypertext Transfer Protocol* (HTTP) requests to a measurement cycle.

Updates to the DNS resolver

As public measurement target lists (e.g. the Alexa top million, see Section 2.4) are typically provided as a list of hostnames, but the PATHspider ECN module measures against IP addresses, it is necessary to resolve the target lists to IP addresses before testing is performed. This can be done using PATHspider’s bulk DNS resolver plugin.

Initially, this was done once, and the resulting list of IP addresses was used for all measurements. However, it was noticed that over the period of a month, a significant number of webhosts would change their IP address. This means that it is necessary to periodically reresolve the list of measurement targets.

Because PATHspider’s bulk DNS resolver would only return the first IP address from the DNS response, it was possible that the same hostname would resolve to a different IP address, even if its DNS records did not change. This would complicate IP address based comparisons of measurement targets with multiple A or AAAA records. The DNS resolver was therefore modified to return all A and AAAA records from the DNS response. Furthermore, the resulting IP addresses are now automatically deduplicated to prevent virtual hosting servers from being tested multiple times during the same measurement run.

Table 4.1: The results of the benchmark of sending the HTTP request in the `connect()` versus the `post_connect()` function. *connect() duration* indicates the length of a single call to the `connect()` function, *Connection duration* indicates the time elapsed between entering the first and leaving the second `connect()` call, *Measurement duration* indicates the time needed to measure a single target, and *HTTP success ratio* indicates the fraction of times the HTTP response was successfully received.

	Request in <code>connect()</code>		Request in <code>post_connect()</code>	
	mean	standard deviation	mean	standard deviation
<code>connect()</code> duration [s]	0.258	0.872	0.256	0.876
Connection duration [s]	6.61	4.63	7.08	4.76
Measurement duration [s]	10.7	5.17	11.3	5.29
HTTP success ratio	0.992	0.001	0.952	0.003

Performing HTTP requests

Originally, the ECN plugin would close each of its connections to a measurement target without transferring any data. Because TCP control segments do not usually have an ECT codepoint set, this meant that no packets with ECT set would be received. To solve this problem, the plugin was modified to perform a HTTP request before closing the connection.

When measuring multiple targets simultaneously, PATHspider will spawn multiple *worker* threads. One for each measurement target. Because some protocol configurations — like the use of ECN — are system wide, PATHspider includes a mechanism to synchronise the workers with the system configuration. Every time it changes the system configuration, PATHspider will call the `connect()` function of each worker. For an ECN measurement, this will happen twice per measurement cycle. To avoid measuring time dependent features, the two `connect()` calls should be made as close together as possible. This also means that the `connect()` functions should be as short as possible. When more time consuming actions have to be performed, this can be done in the `post_connect()` function, that is called after both connections are made.

Because HTTP requests are time consuming, they were initially performed in the `post_connect()` function. However, it is not uncommon for the `post_connect()` call to be made ten seconds or more after the first `connect()` call. This would cause some webservers to (silently) close the HTTP connection before the request was made. To solve this, the *transmission* of the HTTP request was moved to the `connect()` function, while the response is still fetched in the `post_connect()` function. To verify that this does not affect the timing of the `connect()` calls, a benchmarking test was performed. Eight different measurement servers were used, four making the HTTP request in `connect()`, and four making the request in `post_connect()`. A sample of 10.000 hosts from the Alexa top million was measured ten times from each server. The results in Table 4.1 show that there are no negative effects on timing. The small improvement in performance is attributed to the fact that less connection timeouts occur.

4.2.2 Analyzer Modules

When measurement results are uploaded to the PTO, they are first grouped together by measurement run (or more strictly spoken, by time). The PTO then applies a number of analysers to the results. In this work, only the ECN *connectivity* conditions were analysed. Studying the other features of ECN path transparency is left as future work.

Table 4.2: Absolute count of what ECN connectivity condition pairs were measured. The first four columns show how many PATHspider instances observed each condition. The last column shows for how many targets this set of conditions was seen. Combined data from multiple measurements over parts of the Alexa top million. Table truncated.

ECN connectivity condition				Host count
Works	Broken	Transient	Offline	
7	0	0	0	66274
0	0	0	7	2473
6	0	0	1	308
6	1	0	0	215
6	0	1	0	207
...

Single measurement analysis

In the first version of the ECN measurement setup, a single measurement was made from each location, and the analyser structure shown in Figure 4.1 was used. Analysis happens in two steps. Firstly, a *direct* analyser retrieves the uploaded measurement output files from the upload service, and copies all conditions generated by PATHspider to the observation database. Secondly, a *dependency* analyser queries the observation database looking for signs of *path* or *site* dependency. A host (identified by an IP address) is said to exhibit site dependency, if all PATHspider results report connectivity as broken. When there is both broken and working connectivity reported, the host is marked as path dependent.

A number of test measurement runs were performed on a 10,000 host sample from the Alexa top million. Unfortunately, upon inspection of the analysis results, it was determined that this measurement setup was too sensitive to transient effects. Looking at Table 4.2, this is hinted at by the fact that there are about as many hosts that have broken ECN connectivity once, as there are hosts that have transient ECN connectivity once. Thus, it is very likely that transient network effects are causing the host’s connectivity to be marked as broken.

Duplicated measurement analysis

In order to prevent transient network effects from influencing the measurement results, it was decided to duplicate the measurements. Instead of taking one measurement from every location, three measurements are now made close together in time. To process these measurements, the analyser structure has been updated as shown in Figure 4.2. Direct observations from the same measurement location are now merged together into *super* observations. When it is not clear what the condition of the super observation should be, it is set to *weird*. The detailed logic used to merge observations together can be found in Appendix B.2. Results of the condition pairs observed with this new measurement setup are shown in Table 4.3. It can be seen that there is a much smaller fraction of hosts that is marked as broken or transient.

4.2.3 SpiderWeb Configuration

Using SpiderWeb, a weekly measurement of the full Alexa top million was configured. These measurements take place from Frankfurt, London, New York City, San Francisco, Singapore and Toronto. The SpiderWeb configuration file can be found in Appendix B.3.

Table 4.3: Absolute count of what ECN connectivity super condition pairs were measured during a single run. The first four columns show how many PATHspider instances observed each condition. The last column shows for how many targets this set of conditions was seen. Measurement were taken over the full Alexa top million, from 7 locations. Table truncated.

	ECN connectivity super condition					Host count
	Works	Broken	Transient	Offline	Weird	
7	0	0	0	0	0	748586
0	0	0	0	7	0	13236
6	0	0	0	1	0	2246
0	7	0	0	0	0	902
5	0	0	0	2	0	805
6	0	0	0	0	1	534
4	0	0	0	3	0	473
3	0	0	0	4	0	365
2	0	0	0	5	0	257
0	6	0	0	0	1	164
6	1	0	0	0	0	147
1	0	0	0	6	0	138
6	0	0	1	0	0	70
...

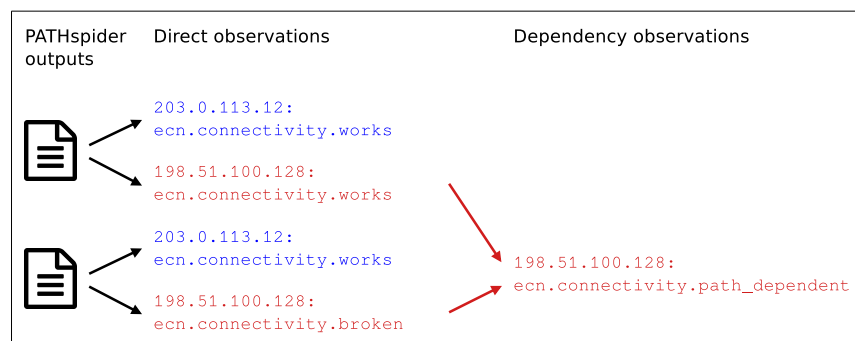


Figure 4.1: Schematic representations of the first version of the analyser logic. PATHspider outputs are converted in to *direct* observation, and from these path and site dependency is derived.

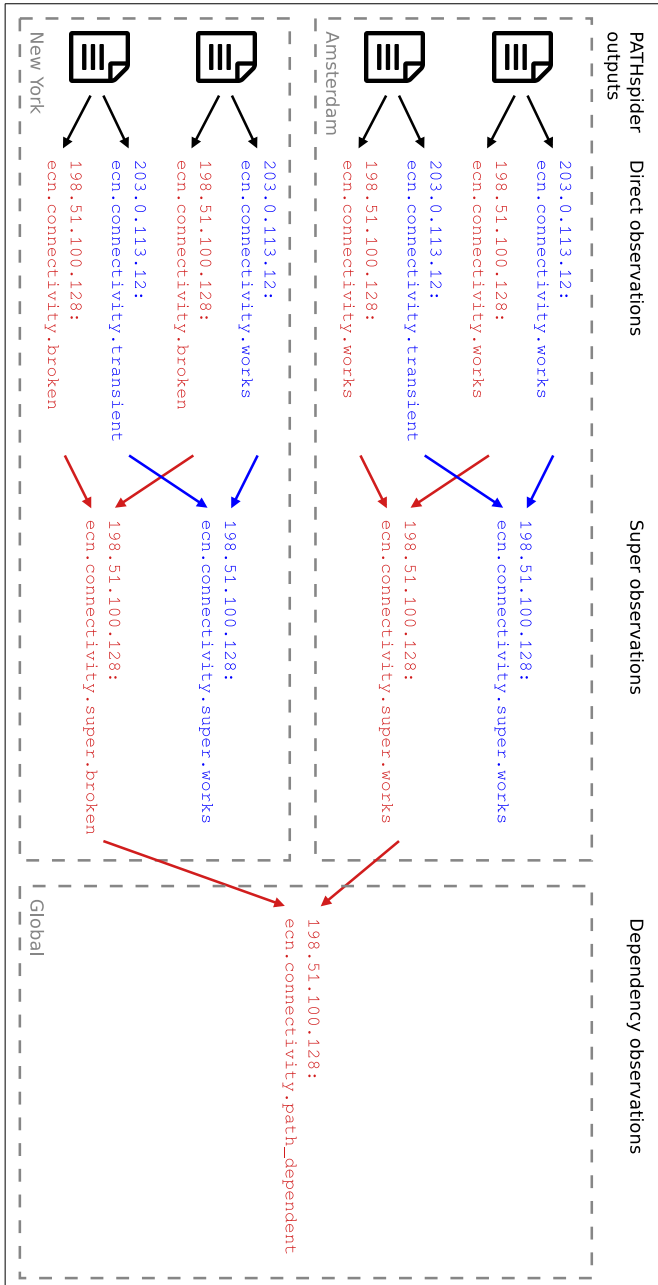


Figure 4.2: Schematic representations of the second version of the analyser logic. This design introduces the concept of *super* observations. PATHspider outputs are converted in to *direct* observation, these are then grouped by location to form *super* observations. Finally, the *super* observations are used to derive path and site dependency.

4.3 Results

Using the three layer analyser scheme from Figure 4.2, results for ECN path and site dependency are derived.

To get a better insight in to the measurement results, different grades of path dependencies are defined:

1. **weak** path dependency
A host is flagged as weakly path dependent, if their is at least one working, and at least one broken super condition for it.
2. **strict** path dependency
A host is flagged as strictly path dependent, if it is weakly path dependent, and there are no transient, offline or weird super conditions for it.
3. **strong** path dependency
A host is flagged as strongly path dependent, if it is strictly path dependent, and if it has at least two broken super conditions associated with it.

Similarly for site dependency, except that there is no notion of strict dependency here:

1. **weak** site dependency
A host is flagged as weakly site dependent, if there are broken, but no working super conditions associated with it.
2. **strong** site dependency
A host is flagged as strongly site dependent, if there are only broken super conditions for it.

4.3.1 State of ECN Path Transparency

At the time of this writing, three measurement runs have been performed and analysed. These measurements took place during week 49 through 51 of 2016. Each measurement was initiated in the beginning of the week, and lasted for about two days. The results are shown in Figures 4.3 and 4.4. Note that these graphs only show the hosts that were tested during all three measurements runs. Because the list of measurement targets gets re-resolved before every run (see Section 4.2.1), this is not the case for all measured hosts. Results in tabular form, as well as plots showing all hosts are included in Appendix C.

As can be seen in Figures 4.3 and 4.4, during each measurement run a similar amount of hosts was marked as path or site dependent. Interestingly enough, only a small number of hosts was always marked as path dependent. This indicates that ECN path dependency is a property that varies greatly over time. However, what exactly is causing this can not be concluded based on these results, and future work will be needed.

The results for ECN site dependency show greater persistency over time. This is to be expected as ECN site dependency is likely to be caused by a middlebox close to the target host, or by the host itself. Thus, one would only expect ECN site dependency to change when either the host or its network infrastructure is upgraded. A possible cause for the measured time variations is the presence of load balancers with heterogeneously configured handlers behind them.

Note again that this analysis is based on the ECN *connectivity* conditions. An analogue analysis can be done on the presence of ECT codepoints to gain a better understanding of how ECN connections are treated after they have been established. This analysis is left as future work.

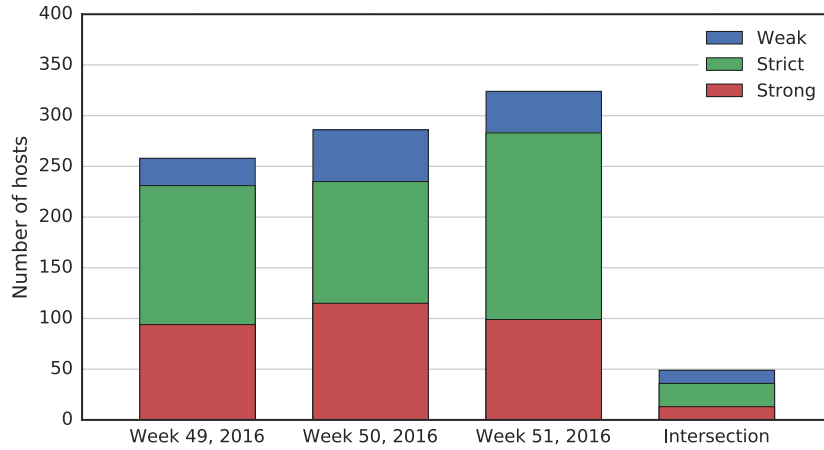


Figure 4.3: Graph of the number of hosts that exhibited weak, strict or strong path dependency over time. Only the hosts that were tested during all three measurement runs are plotted. The *Intersection* bar shows the hosts that were marked during every measurement. Note that the bar graphs are overlapping: all strongly path dependent hosts are also strictly path dependent, et cetera.

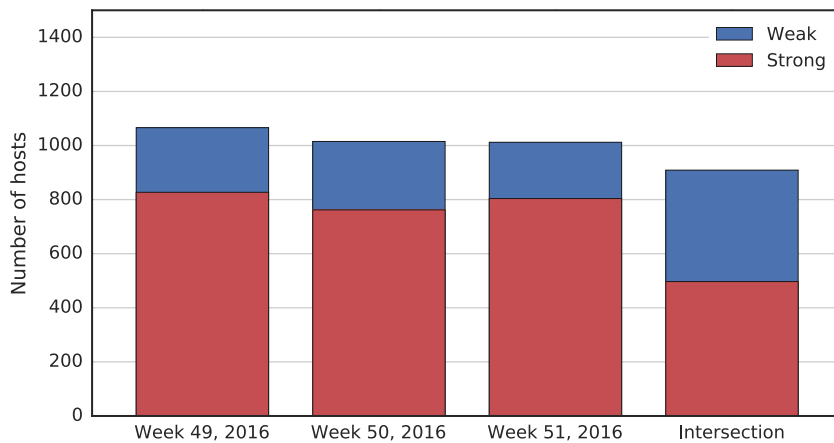


Figure 4.4: Graph of the number of hosts that exhibited weak or strong site dependency over time. Only the hosts that were tested during all three measurement runs are plotted. The *Intersection* bar shows the hosts that were marked during every measurement. Note that the bar graphs are overlapping: all strongly site dependent hosts are also weakly site dependent.

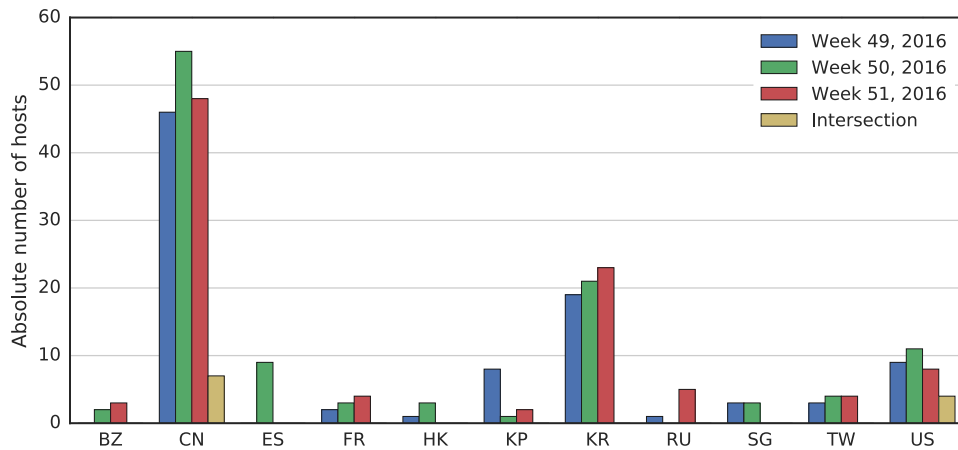
4.3.2 ECN and Internet Censorship

When the results for ECN path transparency are combined with the MaxMind *GeoLite2* geoIP database [14], the geographical distribution of ECN path transparency can be charted. This has been done, and the results are shown in Figures 4.5 and 4.6.

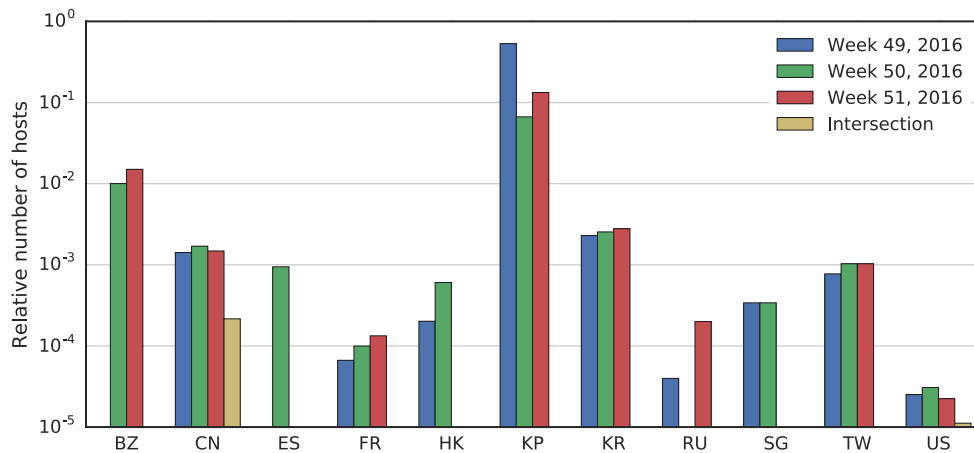
Figure 4.5(a) shows the distribution of strong ECN path dependency over the world in absolute numbers. It appears that China (CN), South Korea (KR) and the United States (US) exhibit the most path dependency. However, when the vertical axis is normalized to the number of entries every country has in the Alexa top million (see Figure 4.5(b)), it can be seen that that path dependency in the United States is over an order of magnitude lower than in the other two countries.

Four countries consistently show high levels of ECN path dependency. That is, one permille or more of interrogated hosts in those countries exhibit ECN path dependency. It is interesting to see that three of those countries (i.e. China (CN), North Korea (KP) and South Korea (KR)) are publicly know to apply heavy internet censorship [15, 16]. This shows that ECN path dependency is a strong indicator of internet freedom. It is most likely that the ECN path dependency in these countries is caused by unequally configured internet censorship servers, and that ECN functionality depends on which server performs the censorship. Why Taiwan (TW) shows a high level of ECN path dependency is unclear. One possible explanation is that because it is an island with limited global connections [17], ECN incompatible equipment at one international peering point can greatly influence the results.

Figure 4.6 shows a similar analysis for strong ECN site dependency. Although the link with internet censorship is not as clear here as it was before, China (CN), North Korea (KP) and South Korea (KR) all exhibit high levels of ECN site dependency. Furthermore, comparing Figure 4.5 with Figure 4.6 again shows that ECN site dependency is a much more time independent property than ECN path dependency.

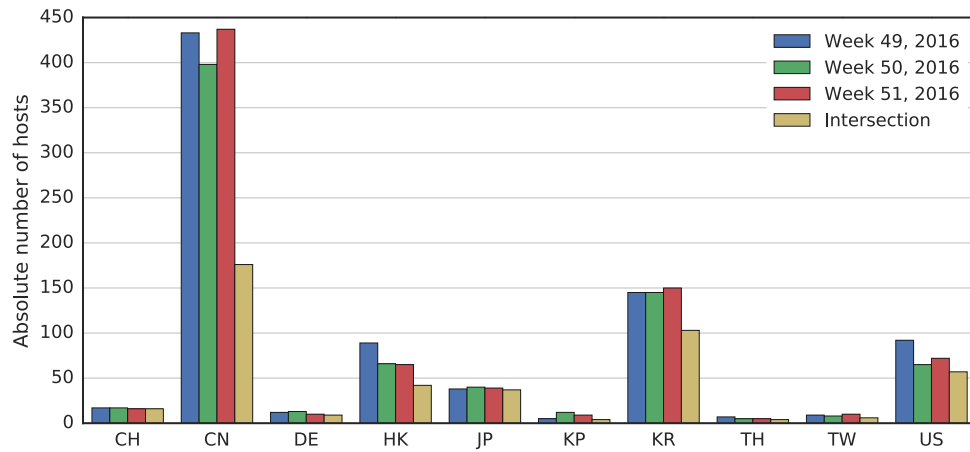


(a) Hosts in absolute numbers.

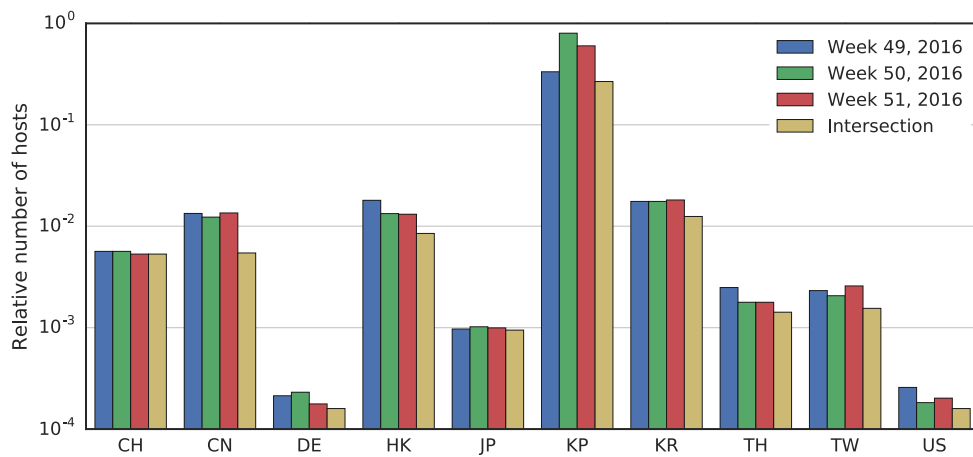


(b) Number of hosts normalized to the total amount of hosts from each country in the Alexa top million.

Figure 4.5: Graph of the number of hosts marked as strongly ECN path dependent by country. Only the top ten countries are shown. The *Intersection* bar shows the hosts that were marked during every measurement.



(a) Hosts in absolute numbers.



(b) Number of hosts normalized to the total amount of hosts from each country in the Alexa top million.

Figure 4.6: Graph of the number of hosts marked as strongly ECN site dependent by country. Only the top ten countries are shown. The *Intersection* bar shows the hosts that were marked during every measurement.

Chapter 5

Conclusion

The goal of this work was to design a system to support the continuous measurement of internet path transparency. For this system, a number of design criteria were specified: Firstly, making measurements should be possible from a large number of locations, and should be cost effective. Secondly, the system should be PATHspider compatible and should be flexible. Lastly, after the initial setup, both the measurements themselves, and the analysis of their results should be fully automated.

To meet all these requirements, a cloud based measurement framework was created. Using a cloud based system is inherently cost effective, and by providing compatibility with a broad range of cloud providers, a large number of measurement locations can be used. Furthermore, PATHspider compatibility is provided by only using Linux servers, and flexibility was considered during the entire design process. In order to allow for automated measurements, the framework is integrated with SaltStack, a software suite specifically designed to manage (ephemeral) cloud based infrastructure. Automated analysis of the measurement results is provided as a service by the Path Transparency Observatory.

After this framework was implemented, it was used for a first application: the measurement of internet path transparency for ECN. This application did not only showcase the effectiveness of the measurement framework, but also provided valuable insights in global ECN support. Two results are particularly interesting. Firstly, it was shown that ECN path dependency shows a high level of time dependence. Secondly, a strong link between nationwide internet censorship and ECN path dependency was discovered. The later clearly illustrates that a free and open internet is not only of social importance, but is also needed to ensure proper technical operation of the net.

Chapter 6

Future Work

This work paves the way for a number of future studies. Furthermore, the results of the ECN study open up a number of research questions. Below a list of suggested future work is provided.

- The most logical next research step is to use the framework designed during this thesis to set up and perform more internet path transparency measurement campaigns. For example, an analysis of when ECT tagged IP packages are received could be performed.
- To allow for more powerful measurement campaigns, a system for *reactive* measurement campaigns could be designed. These measurement campaigns would dynamically be changed depending on the observed measurement results. This would allow for a number of advanced measurement capabilities. For example, basic measurements could be used to trigger more advanced and intensive data collection.
- The ability to perform multiple measurements close together in time should be included in the PATHspider core. As PATHspider is also intended to be used as a stand alone tool, recognising transient effects should not be done on the PTO, but in the PATHspider core.
- As ECN path dependency is shown to be a highly time dependently property, more research is needed to determine why this is the case. This will provide better insight in what should be done to improve global ECN support, and can provide information useful for when designing and rolling out new protocols.
- As a part of the previous item, it should be investigated how filtering out fast changing transients influences the measurement results. This is needed to verify if the current use of *super* observations is justifiable.

Bibliography

- [1] F. Heart, A. McKenzie, J. McQuillan, and D. Walden, “ARPANET completion report,” DARPA, Tech. Rep., 1978.
- [2] International Telecommunication Union. (2016) ICT facts and figures 2016. [Online]. Available: <https://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx>
- [3] B. Carpenter and S. Brim, “Middleboxes: Taxonomy and Issues,” RFC 3234 (Informational), Internet Engineering Task Force, Feb. 2002. [Online]. Available: <http://www.ietf.org/rfc/rfc3234.txt>
- [4] 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*, ser. ANRW ’16. New York, NY, USA: ACM, 2016, pp. 62–64. [Online]. Available: <http://doi.acm.org/10.1145/2959424.2959441>
- [5] E. Gubser, “Building a path transparency observatory,” Master’s thesis, ETH Zurich, 2016.
- [6] SaltStack Inc. (2016) SaltStack automation for CloudOps, ITOps & DevOps at scale. [Online]. Available: <https://saltstack.com/>
- [7] Alexa Internet. (2017) Website traffic statistics. [Online]. Available: <http://www.alexa.com/siteinfo>
- [8] B. Trammel. (2016) MAMI Public Targets List. Measurement and Architecture for a Middleboxed Internet project. [Online]. Available: <https://github.com/mami-project/targets>
- [9] Slack Technologies. (2016) Slack messenger. [Online]. Available: <https://slack.com>
- [10] P. Vixie, *cron(8) Linux User’s Manual*, 4th ed., April 2010.
- [11] K. Ramakrishnan, S. Floyd, and D. Black, “The Addition of Explicit Congestion Notification (ECN) to IP,” RFC 3168 (Proposed Standard), Internet Engineering Task Force, Sep. 2001, updated by RFCs 4301, 6040. [Online]. Available: <http://www.ietf.org/rfc/rfc3168.txt>
- [12] S. Bauer, R. Beverly, and A. Berger, “Measuring the state of ecn readiness in servers, clients, and routers,” in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 171–180.
- [13] A. Medina, M. Allman, and S. Floyd, “Measuring interactions between transport protocols and middleboxes,” in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 336–341.
- [14] MaxMind, Inc. (2016) GeoLite2 Databases. [Online]. Available: <http://dev.maxmind.com/geoip/geoip2/geolite2/>
- [15] J. Pagliery, “A peek into north korea’s internet,” blog, CNN, 2014. [Online]. Available: <http://money.cnn.com/2014/12/22/technology/security/north-korean-internet/index.html>

- [16] OpenNet Initiative. (2017) Country profiles. [Online]. Available: <https://opennet.net/country-profiles>
- [17] TeleGeography, A Division of PriMetrica, Inc. (2017) Submarine cable map. [Online]. Available: <http://www.submarinecablemap.com/>

Appendix A

List of Acronyms

Acronyms

API	application programming interface
ARPANET	Advanced Research Projects Agency Network
CE	congestion encountered
CWR	congestion window reduced
DNS	Domain Name System
ECE	ECN-echo
ECN	Explicit Congestion Notification
ECT	ECN capable transport
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript object notation
MAMI	measurement and architecture for a middleboxed internet
PTO	Path Transparency Observatory
REST	representational state transfer. Alternative form: RESTful
TCP	transmission control protocol

Appendix B

Implementation Information

This appendix contains information that can help the reader get a better understanding of the implementation, but is not necessary to understand this thesis.

B.1 Tables

Table B.1: An overview of the PATHspider command line arguments related to the PTO.

Argument	Description
<code>--pto-url</code>	Sets the location of the PTO to use.
<code>--pro-api-key</code>	Sets the API-key to be used to authenticate to the PTO.
<code>--pto-campaign</code>	Sets the measurement campaign this upload belongs to.
<code>--pto-filename</code>	Sets the name this upload will be saved as.
<code>--pto-location</code>	Can be used to provide extra information about the physical location of the measurement.
<code>--pto-config</code>	Specifies configuration file to use. All of the above arguments can be defined in the configuration file. Command line arguments overwrite configuration file values.

Table B.2: An overview of the entries in a campaign configuration file.

Property	Description
<code>campaign</code>	The name of the measurement campaign.
<code>pathspider_args</code>	The argument string to pass to PATHspider.
<code>input_file</code>	The path to the file containing the target list.
<code>minions</code>	A JSON document specifying what cloud nodes should be instantiated.
<code>when_done</code>	What a measurement node should do after completion of the measurement. Current options are <code>destory</code> and <code>None</code> .

Table B.3: An overview of the event tags used by the PATHspider execution module.

Tag	Description
<code>mami/pathspider/spider/started/</code>	A PATHspider measurement has been started.
<code>mami/pathspider/spider/completed/</code>	A PATHspider measurement was successful.
<code>mami/pathspider/spider/failed/</code>	A PATHspider measurement was unsuccessful.

Table B.4: An overview of the fields used in events generated by the PATHspider execution module.

Field	Type	Description
<code>finished</code>	<code>boolean</code>	True if the event signifies that something has finished.
<code>success</code>	<code>boolean</code>	True if the event signifies that something was successful.
<code>error</code>	<code>string</code>	If an error occurred this field contains the name of the error.
<code>message</code>	<code>string</code>	An optional, freeform, message explaining the event.

Table B.5: Overview of the conditions used by the PATHspider ECN module, and all analyzers developed during this thesis.

Condition	Description
<code>ecn.connectivity.works</code>	A connection could be established with and without attempting ECN negotiation.
<code>ecn.connectivity.broken</code>	A connection could be established without ECN negotiation, but not with ECN negotiation.
<code>ecn.connectivity.transient</code>	A connection could be established with ECN negotiation, but not without ECN negotiation.
<code>ecn.connectivity.offline</code>	A connection could never be established.
<code>ecn.negotiated</code>	ECN could be negotiated.
<code>ecn.not_negotiated</code>	ECN could not be negotiated.
<code>ecn.ect_zero.seen</code>	The ECT(0) codepoint was received.
<code>ecn.ect_one.seen</code>	The ECT(1) codepoint was received.
<code>ecn.ce.seen</code>	The CE codepoint was received.
<code>ecn.connectivity.super.works</code>	Same as <code>ecn.connectivity.works</code> , but based on multiple observations
<code>ecn.connectivity.super.broken</code>	Same as <code>ecn.connectivity.broken</code> , but based on multiple observations
<code>ecn.connectivity.super.transient</code>	Same as <code>ecn.connectivity.transient</code> , but based on multiple observations
<code>ecn.connectivity.super.offline</code>	Same as <code>ecn.connectivity.offline</code> , but based on multiple observations
<code>ecn.connectivity.super.weird</code>	When multiple observations were combined in to this one, nothing could be derived about the hosts connectivity.
<code>ecn.path_dependent.weak</code>	See Section 4.3.
<code>ecn.path_dependent.strict</code>	See Section 4.3.
<code>ecn.path_dependent.strong</code>	See Section 4.3.
<code>ecn.site_dependent.weak</code>	See Section 4.3.
<code>ecn.site_dependent.strong</code>	See Section 4.3.

B.2 Super Observation Logic

The following code snippet is used to merge multiple conditions in to one *super condition*.

```

1  def calculate_super_condition(conditions):
2      """
3      Combines multiple conditions in to a super condition.
4      First looks if connections with or without ECN have ever been seen working.
5      Then uses this information to derive an observation about the host.
6      See source for exact logic.
7      :param list conditions: A list of conditions that should be merged.
8          Each element should be in the input conditions of this analyzer
9      :returns: the supercondition derived from the inputconditions.
10         Will always be in the output conditions of this analyzer
11     :rtype: list
12     """
13
14     ecn_seen_working = False
15     no_ecn_seen_working = False
16
17     ## FIRST, find out what we have seen working
18     for condition in conditions:
19         if condition == 'ecn.connectivity.works':
20             ecn_seen_working = True
21             no_ecn_seen_working = True
22
23         elif condition == 'ecn.connectivity.broken':
24             no_ecn_seen_working = True
25
26         elif condition == 'ecn.connectivity.transient':
27             ecn_seen_working = True
28
29         elif condition == 'ecn.connectivity.offline':
30             pass
31
32     ## SECOND, determine on the actual super condition.
33     # Everything is working, Yay!
34     if ecn_seen_working and no_ecn_seen_working:
35         super_condition = 'ecn.connectivity.super.works'
36
37     # Nothing is working, host must be offline!
38     if not ecn_seen_working and not no_ecn_seen_working:
39         super_condition = 'ecn.connectivity.super.offline'
40
41     # This hints at ECN broken. Let's verify that all observations agree:
42     elif not ecn_seen_working and no_ecn_seen_working:
43         if verify_all_elements_equal(conditions, 'ecn.connectivity.broken'):
44             super_condition = 'ecn.connectivity.super.broken'
45         else:
46             super_condition = 'ecn.connectivity.super.weird'
47
48     # This hints at ECN transient. Let's verify that all observations agree:
49     elif ecn_seen_working and not no_ecn_seen_working:
50         if verify_all_elements_equal(conditions, 'ecn.connectivity.transient'):
51             super_condition = 'ecn.connectivity.super.transient'
52         else:
53             super_condition = 'ecn.connectivity.super.weird'
54
55     return [super_condition]
56
57 def verify_all_elements_equal(array_to_check, element_value = None):
58     """
59     Verifies that all elements in an array are equal to a certain value.
60     If no value is passed, the first element of the array is used.
61     :param list array_to_check: the array to check the elements from
62     :param element_value: the value the elements should be equal to.
63         defaults to the first element of the array.
64     :returns: True if all elements are equal, False otherwise
65     :rtype: bool
66     """
67
68     # There is nothing in the array, that's not right
69     if len(array_to_check) == 0:
70         return False
71
72     # if no value is specified, just check that all values in the array are
73     # equal to the first one.
74     if element_value == None:
75         element_value = array_to_check[0]

```

```
76
77     for element in array_to_check:
78         if element != element_value:
79             return False
80
81     return True
```

B.3 SpiderWeb Configuration File

This is the configuration file used for the measurement campaign presented in Chapter 4.

```
1  {
2      "pathspider_args": "-i eth0 -w50 --pto-campaign {campaign}
          --pto-url 'https://observatory.mami-project.eu/hdfs'
          --pto-api-key <<redacted>> --pto-filename {id} --pto-location
          {cloud-location} ecn",
3      "input_file":
          "/var/dns-autoresolv/more-is-better/more-is-better-latest-scramble.csv",
4      "minions":
5          {"do-nyc2-2048": 3,
6           "do-sfo1-2048": 3,
7           "do-ams3-2048": 3,
8           "do-sgp1-2048": 3,
9           "do-lon1-2048": 3,
10          "do-fra1-2048": 3,
11          "do-tor1-2048": 3
12         },
13      "when_done": "destroy",
14      "campaign": "more-is-better"
15 }
```

Appendix C

Results of the ECN Study

This appendix adds to Section 4.3 by presenting the measurement data in tabular form, and by providing additional plots.

Table C.1: Overview of the number of hosts that were tagged as ECN path or site dependent per measurement run. The *intersection* row shows the number of hosts that were tagged with a certain condition over all three measurement runs. This table considers all measured hosts.

Measurement run	ECN path dependency			ECN site dependency	
	weak	strict	strong	weak	strong
Week 49, 2016	281	250	104	1155	897
Week 50, 2016	312	257	122	1096	818
Week 51, 2016	343	301	106	1088	866
Intersection	49	69	13	909	497

Table C.2: Overview of the number of hosts that were tagged as ECN path or site dependent per measurement run. The *intersection* row shows the number of hosts that were tagged with a certain condition over all three measurement runs. This table considers only hosts that were tested during all three measurement runs.

Measurement run	ECN path dependency			ECN site dependency	
	weak	strict	strong	weak	strong
Week 49, 2016	258	231	94	1066	827
Week 50, 2016	286	235	115	1015	762
Week 51, 2016	324	283	99	1012	804
Intersection	49	69	13	909	497

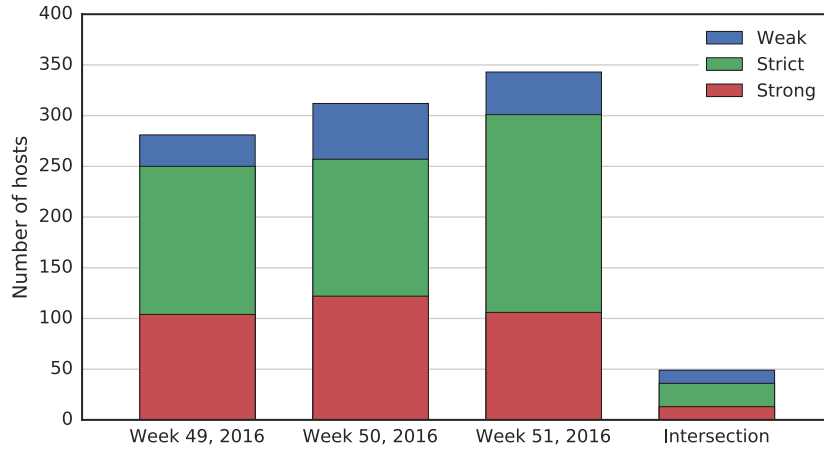


Figure C.1: Graph of the number of hosts that exhibited weak, strict or strong path dependency over time. All measured hosts are plotted. The *Intersection* bar shows the hosts that were marked during every measurement. Note that the bar graphs are overlapping: all strongly path dependent hosts are also strictly path dependent, et cetera.

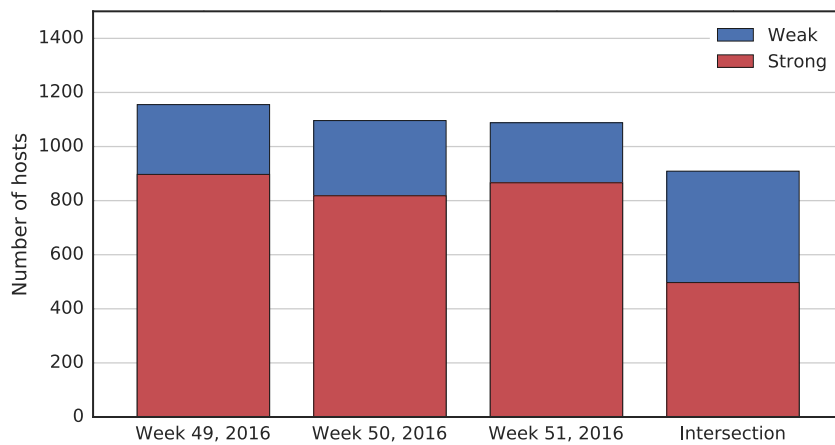


Figure C.2: Graph of the number of hosts that exhibited weak or strong site dependency over time. All measured hosts are plotted. The *Intersection* bar shows the hosts that were marked during every measurement. Note that the bar graphs are overlapping: all strongly site dependent hosts are also weakly site dependent.

Table C.3: Overview of the number of hosts that were tagged as strongly ECN path dependent, broken down per country. Numbers are given both in absolute form, and as a fraction of the total number of hosts that were measured in that country. The *intersection* column shows the number of hosts that were tagged with a certain condition over all three measurement runs. This table considers all measured hosts.

Country		Week 49, 2016		Week 50, 2016		Week 51, 2016		Intersection	
		Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
BR	Brazil	2	2.68e-04	0	0.00e+00	1	1.34e-04	0	0.00e+00
BZ	Belize	0	0.00e+00	2	1.01e-02	3	1.51e-02	0	0.00e+00
CA	Canada	0	0.00e+00	0	0.00e+00	1	8.35e-05	0	0.00e+00
CN	China	46	1.42e-03	55	1.70e-03	48	1.48e-03	7	2.16e-04
DE	Germany	1	1.77e-05	2	3.54e-05	1	1.77e-05	1	1.77e-05
EC	Ecuador	1	5.78e-03	1	5.78e-03	1	5.78e-03	1	5.78e-03
ES	Spain	0	0.00e+00	9	9.47e-04	0	0.00e+00	0	0.00e+00
FR	France	2	6.67e-05	3	1.00e-04	4	1.33e-04	0	0.00e+00
HK	Hong Kong	1	2.02e-04	3	6.05e-04	0	0.00e+00	0	0.00e+00
IT	Italy	1	1.10e-04	0	0.00e+00	0	0.00e+00	0	0.00e+00
KP	North Korea	8	5.33e-01	1	6.67e-02	2	1.33e-01	0	0.00e+00
KR	South Korea	19	2.30e-03	21	2.54e-03	23	2.78e-03	0	0.00e+00
MD	Moldova	1	6.45e-03	0	0.00e+00	0	0.00e+00	0	0.00e+00
MX	Mexico	1	7.64e-04	1	7.64e-04	0	0.00e+00	0	0.00e+00
MY	Malaysia	1	7.98e-04	0	0.00e+00	1	7.98e-04	0	0.00e+00
NL	Netherlands	0	0.00e+00	0	0.00e+00	1	4.20e-05	0	0.00e+00
PL	Poland	0	0.00e+00	0	0.00e+00	1	1.33e-04	0	0.00e+00
PS	Palestine	0	0.00e+00	1	3.57e-02	0	0.00e+00	0	0.00e+00
RU	Russia	1	3.99e-05	0	0.00e+00	5	2.00e-04	0	0.00e+00
SA	Saudi Arabia	1	2.37e-03	2	4.74e-03	1	2.37e-03	0	0.00e+00
SG	Singapore	3	3.41e-04	3	3.41e-04	0	0.00e+00	0	0.00e+00
TH	Thailand	1	3.55e-04	1	3.55e-04	0	0.00e+00	0	0.00e+00
TR	Turkey	2	2.76e-04	1	1.38e-04	1	1.38e-04	0	0.00e+00
TW	Taiwan	3	7.73e-04	4	1.03e-03	4	1.03e-03	0	0.00e+00
US	US	9	2.52e-05	11	3.08e-05	8	2.24e-05	4	1.12e-05
VN	Vietnam	0	0.00e+00	1	2.99e-04	0	0.00e+00	0	0.00e+00

Table C.4: Overview of the number of hosts that were tagged as strongly ECN site dependent, broken down per country. Numbers are given both in absolute form, and as a fraction of the total number of hosts that were measured in that country. The *intersection* column shows the number of hosts that were tagged with a certain condition over all three measurement runs. This table considers all measured hosts.

Country	Week 49, 2016		Week 50, 2016		Week 51, 2016		Intersection		
	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	
AR	Argentina	2	9.55e-04	2	9.55e-04	2	9.55e-04	2	9.55e-04
AT	Austria	1	5.36e-04	1	5.36e-04	1	5.36e-04	1	5.36e-04
AU	Australia	1	1.74e-04	1	1.74e-04	1	1.74e-04	1	1.74e-04
AZ	Azerbaijan	0	0.00e+00	1	4.55e-03	0	0.00e+00	0	0.00e+00
BA	Bosnia	3	3.70e-02	3	3.70e-02	3	3.70e-02	3	3.70e-02
BR	Brazil	0	0.00e+00	1	1.34e-04	1	1.34e-04	0	0.00e+00
CA	Canada	1	8.35e-05	1	8.35e-05	1	8.35e-05	1	8.35e-05
CH	Switzerland	17	5.64e-03	17	5.64e-03	16	5.31e-03	16	5.31e-03
CN	China	433	1.34e-02	398	1.23e-02	437	1.35e-02	176	5.43e-03
DE	Germany	12	2.12e-04	13	2.30e-04	10	1.77e-04	9	1.59e-04
DK	Denmark	1	4.95e-04	1	4.95e-04	1	4.95e-04	1	4.95e-04
DZ	Algeria	1	6.29e-03	1	6.29e-03	1	6.29e-03	1	6.29e-03
EE	Estonia	1	5.62e-04	1	5.62e-04	1	5.62e-04	1	5.62e-04
ES	Spain	3	3.16e-04	3	3.16e-04	3	3.16e-04	3	3.16e-04
FR	France	5	1.67e-04	6	2.00e-04	4	1.33e-04	4	1.33e-04
GB	UK	3	1.16e-04	1	3.85e-05	2	7.71e-05	1	3.85e-05
HK	Hong Kong	89	1.80e-02	66	1.33e-02	65	1.31e-02	42	8.48e-03
HU	Hungary	1	4.96e-04	1	4.96e-04	1	4.96e-04	1	4.96e-04
ID	Indonesia	1	3.72e-04	1	3.72e-04	1	3.72e-04	1	3.72e-04
IN	India	2	2.66e-04	2	2.66e-04	2	2.66e-04	2	2.66e-04
IR	Iran	2	4.46e-04	2	4.46e-04	3	6.69e-04	2	4.46e-04
JP	Japan	38	9.69e-04	40	1.02e-03	39	9.95e-04	37	9.44e-04
KP	North Korea	5	3.33e-01	12	8.00e-01	9	6.00e-01	4	2.67e-01
KR	South Korea	145	1.75e-02	145	1.75e-02	150	1.81e-02	103	1.25e-02
MX	Mexico	2	1.53e-03	2	1.53e-03	2	1.53e-03	2	1.53e-03
MY	Malaysia	2	1.60e-03	2	1.60e-03	1	7.98e-04	1	7.98e-04
PE	Peru	2	5.65e-03	2	5.65e-03	2	5.65e-03	2	5.65e-03
PH	Philippines	1	2.61e-03	0	0.00e+00	0	0.00e+00	0	0.00e+00
PT	Portugal	2	1.42e-03	2	1.42e-03	3	2.13e-03	2	1.42e-03
RU	Russia	4	1.60e-04	5	2.00e-04	5	2.00e-04	4	1.60e-04
SA	Saudi Arabia	1	2.37e-03	0	0.00e+00	1	2.37e-03	0	0.00e+00
SV	El Salvador	1	1.59e-02	1	1.59e-02	1	1.59e-02	1	1.59e-02
TH	Thailand	7	2.48e-03	5	1.77e-03	5	1.77e-03	4	1.42e-03
TR	Turkey	3	4.13e-04	3	4.13e-04	4	5.51e-04	3	4.13e-04
TW	Taiwan	9	2.32e-03	8	2.06e-03	10	2.58e-03	6	1.55e-03
UA	Ukraine	1	2.53e-04	1	2.53e-04	1	2.53e-04	1	2.53e-04
US	US	92	2.57e-04	65	1.82e-04	72	2.01e-04	57	1.59e-04
VN	Vietnam	3	8.98e-04	2	5.98e-04	4	1.20e-03	2	5.98e-04
WS	Samoa	0	0.00e+00	0	0.00e+00	1	5.00e-01	0	0.00e+00