Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik und Kommunikationsnetze

Tracing Internet Path Transparency

Master Thesis

Michael Walter
waltermi@ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

Supervisors:
Brian Trammell
Dr. Mirja Kühlewind
Prof. Dr. Laurent Vanbever

December 12, 2017

Abstract

Middleboxes like firewalls or NATs, get increasingly deployed in the Internet. Since the manipulation of Internet traffic by these middleboxes can lead to the dropping of new protocols or protocol extensions, tools are needed that can detect these protocol impairments. One tool that does exactly this is PATHspider [1]. Trough A/B testing PATHspider is able to detect connectivity issues with new protocols. The tool however is not able to exactly localize the source of a detected protocol impairment.
This thesis aims to solve this issue through integration of a tracerouting functionality into the PATHspider tool that can be activated if a protocol impairment has been detected. As a model for the integration, the traceroute extension tracebox [2] has been used. Middlebox interference is therein detected by sending TCP packets with different TTL values and comparing the content with the packet encapsulated in the returned ICMP message.
Since performing traceroutes leads to a huge amount of data, the graph database tool Neo4j is evaluated as way to visualize and further analyze the collected data. The IP/TCP protocol extension explicit congestion notification (ECN) is a good example to demonstrate protocol impairments. In a measurement campaign an update on the current status of ECN will be reported. Additionally, the usefulness of performing traceroutes with PATHspider is shown by demonstrating the propagation of the now widely supported ECN code points in the Internet. In contrast, the same will be done for the IP protocol extension differentiated services code point (DSCP) which is not (yet) widely deployed.

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, the deployment of middleboxes in the Internet has been increasing. Middleboxes are network devices that perform other functions than just forwarding traffic, this can be for security reasons, e.g. traffic control to firewalls or for performance reasons, e.g. traffic classification. These middleboxes are often not transparent and can have a negative impact on the evolution of IP/TCP when new protocols are not understood and thus not treated in the right way [3]. A good example for that is the IP/TCP protocol extension Explicit Congestion Notification (ECN). In the early deployment of ECN, middleboxes often cleared the IP bits or even dropped packets that indicated ECN-capability. Different studies showed a gradual improvement of the situation in the last years [4], [5], but as the measurements in this thesis will show, the problem still persists today.

To detect network impairments in current and future protocols there is a demand for network testing tools. One such tool is PATHspider, which through controlled experimental A/B testing fills a gap in the current Internet active measurement software environment [1]. PATHspider measures connectivity to a target point for both without (A measurement) and with (B measurement) the new protocol or the extension enabled and observes differences.
While performing measurements from different vantage points can help to distinguish between protocol impairments that occur close to the target or closer to the Internet core, an exact localization is not possible. The goal of this thesis is the implementation of a functionality that allows for a localization of the protocol impairment if one has been detected. For this, PATHspider will be extended by the possibility to make the Internet paths transparent with a traceroute functionality. In particular, the ability to compare the content of sent TCP packets with variable TTL values with the encapsulated packet in the returned ICMP message as demonstrated by G. Detal et al. [2] is integrated.

Visualizing and Analyzing the huge amount of data produced by performing traceroutes on parts of the Internet can be a challenge. This is why another task of this thesis is the evaluation of the graph database tool Neo4j as a help in displaying and analyzing the data obtained by the traceroute extension of PATHspider.

1.2 Contributions

The contributions of this semester thesis are as follows:

- Extending PATHspider with the functionality of performing traceroutes on previously measured targets.
- Extending PATHspider with an on line tracerouting functionality which allows for an immediate tracerouting after a protocol impairment has been detected during the measurement process.
- Visualizing the obtained traceroute data with Neo4j.
- Measuring the the current state of ECN deployment regarding connectivity and negotiation.
- Measuring and localizing ECN IP bit manipulation.
- Measuring and localizing differentiated services code point (DSCP) manipulation.

1.3 Overview

Chapter 2 gives some background information about the architecture and mechanics of PATHspider and the functionality of traceroute and its extension tracebox. Also the TCP protocols ECN and DSCP on which the measurements are performed are outlined. In Chapter 3 a detailed explanation of the concept and implementation of traceroute/tracebox into the existing PATHspider tool is provided. Informations about Neo4j and how the graph platform can be used to visualize traceroute data can be found in Chapter 4. Some general measurements of ECN and a demonstration of the abilities of the traceroute extension of PATHspider by detecting packet manipulation in the Internet are performed in Chapter 5. Finally in Chapter 6 some conclusions are drawn.

Chapter 2

Background

In this Chapter, the background of the PATHspider tool and the functionality of traceroute/tracebox will be provided. Furthermore, the two Internet protocol extensions Explicit Congestion Notification (ECN) and Differentiated Services Code Point (DSCP) will be explained, since these extensions are heavily involved in the measurements in Chapter 5.

2.1 PATHspider

PATHspider is a measurement tool for active testing of Internet Path transparency by performing A/B testing [1]. The tool is written in Python and a generalization of the ecnspider which has been used to probe for failures in the negotiation of Explicit Congestion Notification [5]. The extension with plug-ins makes it easy to do measurements of other protocols or protocol extensions. The tool, as well as the source code and the documentation are available at `https://pathspider.net`.

2.1.1 Architecture of PATHspider

PATHspider is structured in four main components or modules, the configurator, the workers, the observer and the merger as can be seen in Figure 2.1. The modules are implemented as threads and are launched with the start of PATH-spider. The jobs from the target queue, that contain at least the target address and port number, are evenly distributed among a selectable amount of workers. Every worker tries to establish two connections: First with the "A" configuration that serves as the base line measurement and second the "B" configuration. Measurements with these two configurations permit a distinction between hosts that do not respond and hosts that are failing because of the use of a particular transport protocol or extension.
Since some protocol extensions are in need of system-wide parameter change, a

configurator is needed. The configurator waits until all active workers complete the "A" measurement, then it changes the necessary parameters in the Linux kernel and the workers can perform the measurement with the "B" configuration. This procedure is repeated multiple times until no more jobs remain.
The observer module captures packets independently by using Python bindings for libtrace [6]. Each incoming packet is assigned to a flow depending on the source and destination ports and addresses observed in the packet header. Packets that are part of the same test for the same target will thus be recognized by belonging to the same flow.
The packet and its respective flow are passed to function chains that do analysis. There is a basic chain, that does count the number of packets and extract the source and destination ports and IPs. For specific measurements there exist other chains, that extract the required information for the respective measurement, like extracting the state of flags from an IP or TCP header.
If all packets belonging to a flow have been observed, the flow is completed and the corresponding flow information as well as the initial job record are passed to the merger. There, the information about the success of the connection as well as the information collected by the function chains are merged and given to the output.

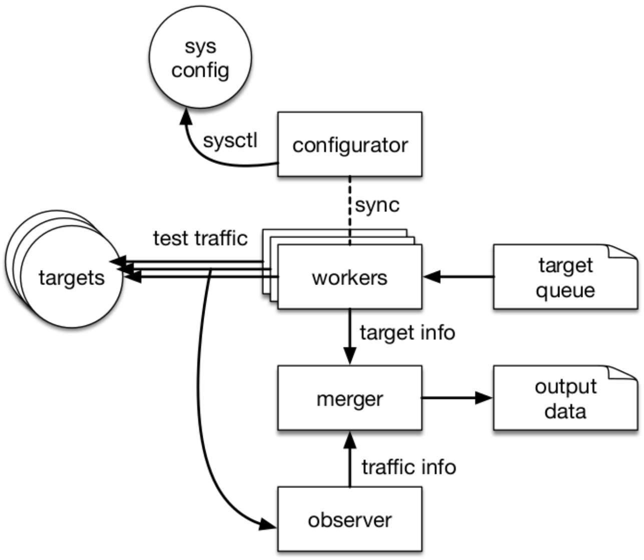

Figure 2.1: Architecture of PATHspider as depicted in [1]

The output is provided as a Newline delimited JavaScript Object Notation (NDJSON) file as can be seen in an example in Figure 2.2.
In the first line, the destination IP for which the test was performed can be seen. In the next lines some information about the job, that was already included

in the initial jobs list is displayed. Also included is the time frame in which packets of the respective job were observed. And most importantly, the plugin specific conditions that contain the information about the success of the connection and the state of observed flags important to the tested protocol are included.

```
{
"dip": "87.240.165.80",
"domain": "www.vk.com",
"rank": "16",
"dp": "80",
"time": {"from": "2017-09-06 16:14:10.491451",
         "to": "2017-09-06 16:14:15.655903"},
"conditions": ["ecn.connectivity.works",
          "ecn.negotiation.succeeded",
          "ecn.ipmark.ect0.seen",
          "ecn.ipmark.ect1.not_seen",
          "ecn.ipmark.ce.not_seen"]
}
```

Figure 2.2: Output of PATHspider

2.1.2 Extensibility

The modular architecture of PATHspider allows for an easy extension with plugins for the measurement of IP or transport protocol or any extension of these. PATHspider plugins are built by extending an abstract class that implements functions for different modules like the configurator, the workers or the merger. For the configurator, the two functions that prepare the configurator for each attempted connection mode have to be defined in the plugin. Optionally, pre-connection or post-connection routines can be implemented by the plugin. Those can perform additional actions with the opened connections without being synchronized by the configurator.
Plugins also implement the function chains which are used for the analysis of the packets collected by the observer. There exist chains for IP, TCP and UDP packets, that allow observation of different behavior. If for a specific measurement the available chains don't analyze the proper sections of the packet headers, additional chains can be designed and added.
The final function of the plugin is the merger function that merges the results and flows of the "A" and "B" measurements. This is done by assigning conditions to the initial jobs, depending on the success of the respective measurements.

2.2 Tracing Paths in the Internet

2.2.1 Traceroute

The goal of traceroute is to make the routing paths in the Internet visible it is thus an important tool in helping to make the Internet more transparent. This is why the ability to perform a traceroute is available on many modern operating systems.
For traceroute to work, it relies on a support protocol of the Internet layer, called the Internet Control Message Protocol (ICMP) [7]. The ICMP is used to send error messages and operational information, for example if a destination is unreachable or for router advertisement. One of the error messages is the "Time Exceeded" message, which informs that a packet is routed no further because the time-to-live (TTL) of the packet in the Internet is expired. The TTL in IPv4 or the hop limit in IPv6 is a 8 bit field allowing for a maximum hop number of 255. This number is set by the sender of a packet and is normally set to the maximum, to 128 or 64 hops. When the packet gets routed through the Internet the number gets reduced every time a host is passed. The TTL prevents a packet from circulating through the Internet indefinitely.
When a host receives a packet with a TTL of zero, it will send back an ICMP packet. The packet will contain the IP header and at least 8 bytes of the underlying transport protocol packet header of the initial sent packet.
There are different ways to implement traceroute, with ICMP echo packets, UDP packets or TCP packets. In this thesis traceroute with TCP packets is used. At first a TCP SYN packet with a TTL of 0 is sent to the destination for which the path should be traced. The first host the packet is routed through will respond with an ICMP message because the host doesn't route packets with TTL 0 any further. After the sender has received the ICMP packet, a new TCP SYN packet is sent out, now with a TTL of 1. The packet now passes the first host on the route, which decreases the TTL to zero and the second host will respond with an "Time Exceeded" ICMP message. The process is repeated with increasing TTL values until no more ICMP messages are returned or until the destination server responds with a TCP SYN/ACK message, which indicates that the packet has reached the destination.

Tracerouting is a good way to make routing in the Internet visible, but unfortunately it has its limitations. First, some host will just drop packets with a TTL of zero and not respond with an ICMP message. This will lead to "holes" in the path. It can be seen that a host that reduced the TTL is in place but no information about it is available.
Another issue making it harder to achieve path transparency is load balancing. Load balancers distribute network traffic on different hosts and this will lead to different routing paths to a destination. To reveal different paths, tracerouting

has to be repeated multiple times and there is never a guarantee, that all paths have been discovered.
Finally, the path a packet takes to a host doesn't necessarily have to be the same path the packet is routed back to the sender, anything happening on that path stays invisible to traceroute.

2.2.2 Tracebox

Tracebox takes the functionality of traceroute and extends it by detecting middlebox interferences in the Internet [2]. This is done similarly to traceroute by sending packets with increasing TTL values and waiting for ICMP messages. In difference to traceroute, tracebox allows for control of the probes sent and keeps track of those. The content of the sent back ICMP time-exceeded packets is compared to the stored original probes. This comparison allows for the detection and approximate localization of middleboxes by detecting changes in the header, other than the TTL.
Tracebox is able to detect changes in the IP header and in the first 8 bytes of the underlying transport layer header, because this is the minimal content of the ICMP packet according to RFC 792 [7]. The newer RFC 1812 [8] recommends including the whole IP packet into the ICMP reply, which is increasingly implemented into newly sold routers. This "full" ICMP allows for a better detection of changes in the packets, since the transport layer header can now be fully compared by tracebox.

2.3 Internet Protocol Extensions

2.3.1 Differentiated Service Code Point (DSCP)

Differentiated services is a networking protocol that provides mechanism for managing network traffic and quality of service. With differentiated services, network traffic can be classified into more and less important traffic. For example, critical network traffic like voice or streaming data may get precedence over non-critical web traffic.
DSCP occupies the 6 most significant bits of the 8-bit differentiated services field in the IPv4 and IPv6 header, as can be seen in Table 2.1b. Those 6-bits allow for a classification into 64 different traffic classes of which not all are officially assigned. DSCP was introduced by RFC 2474 [9] in 1998. Previous to that, the field was called the Type of Service field (2.1a) which consisted of a 3-bit Precedence field that had a similar function to todays DSCP by treating high priority packets with precedence. The last 5 bits went trough many definitions and were never in widespread use.

0	1	2	3	4	5	6	7
Precedence			Type of Service				

(a) Type of Service field

0	1	2	3	4	5	6	7
DSCP						ECN	

(b) Differentiated Services field

Table 2.1: Evolution from Type of Service to Diffferentiated Services

DSCP is not required to be used by network operators, this means the code points can be overwritten and deleted when entering subnetworks. This is done quite often as network operators have no reason to trust the code points set on packets entering their network. More common is the network internal usage of DSCP, packets sometimes are assigned code points when entering networks that have no meaning externally.
There are also DSCP values that have an official assigned classification, listed in Table 2.2. The most basic one is default forwarding which is basically just best effort forwarding. Code point 46 is reserved for Expedited Fowarding (EF), traffic marked by this has the characteristic of low delay and low loss, it is often given higher priority than all other traffic classes. Assured forwarding (AS) allows the operator to provide assurance of delivery as long as some subscribed rate of traffic is not exceeded. For assured forwarding 12 code points categorized in different classes with different drop possibilities are reserved. The class selector (CS) code points exist to maintain backward compatibility to the old precedence field. Each of the original 7 IP precedence classes can be mapped to a class selector, which only uses the first three bits.

DSCP Value	**Classification**
000 000	Default Forwarding (Best effort)
101 110	Expedited Forwarding

001 010
...
100 110
	Assured Forwarding (AF11)
	...
	Assured forwarding (AF43)

001 000
...
111 000
	Class Selector code point (CS1)
	...
	Class Selector code point (CS7)

Table 2.2: Differentiated Services Code Points

2.3.2 Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) is an optional IP/TCP protocol extension. It allows for notification of network congestion without dropping packets. Providing a successful negotiation of ECN between two endpoints, the receiving router of a packet may set a mark in the IP header to signal congestion on a echoed-back packet. After receiving the packet, the sender can reduce its transmission rate to avoid further congestion.

Since the release of RFC 3168 [10] the two least significant bits of the differentiated services field in the IPv4 and IPv6 header are used for ECN (see Table 2.1b). The four different code points used by ECN can be seen in Table 2.3. If a sender supports ECN, then packets can be marked by either ECT(0) or ECT(1) which are interchangeable. If congestion in the queue of the receiver is observed, the respective router can change the code point to CE.
In the transport layer in the TCP header ECN introduces two additional 1-bit flags: the ECE-Echo (ECE) and the Congestion Window Reduced (CWR) flags. If a congestion occurred, ECE is set on all packets back to the receiver until the sender acknowledges the ECE and sets CWR.
For ECN to be used, it has to be negotiated first. This is done during the TCP handshake: The connection initiator sets the ECE and CWR flags on the initial SYN, the receiver responds by setting the ECE on the SYN/ACK. When the ECE is received, the sender can set the ECT code point on all following packets. While ECN is implemented in most operating systems nowadays, most system won't negotiate ECN by default. ECN will be negotiated if requested by a remote node but negotiation in the opposite direction will have to be activated manually in the operating system.

Code Point	**Description**
00	Non ECN-Capable Transport, Non-ECT
10	ECN Capable Transport, ECT(0)
10	ECN Capable Transport, ECT(1)
11	Congestion Encountered, CE

Table 2.3: ECN code points in the IP header

Chapter 3

Traceroute Extension

Building a PATHspider extension for performing traceroutes was done honoring the modular structure of the PATHspider architecture (Figure 2.1) by adding new modules. One module is required to do the actual sending of the traceroute packets and one module is required to assemble the information collected by the observer and put it into the output. Those two modules are called the sender and the trace-merger. Additionally, a new traceroute function chain that identifies traceroute packets and analyses them has been developed.
The extension can be used outside the measurement work flow of PATHspider with a traceroute command, as well as on line during the measurement process. This chapter explains the components added to PATHspider and the different execution modes.

3.1 Traceroute Extension Components

3.1.1 Sender Module

The sender module is responsible for sending the traceroute TCP SYN packets. The IP addresses of the endpoints for which the traceroute should be performed are put into the input queue for the module. If the approximate number of hops to reach the destination is also available, it will be added to the queue too. An available approximate hop number can reduce the time for performing a traceroute substantially. If none is available, a default hop number of 50 is assumed, since measurements showed that in practice paths don't exceed this number of hops unless the packet is trapped in a loop.
The sending of the packets itself is done by the packet manipulation tool Scapy [11]. With Scapy packets can be crafted manually layer by layer. An example of the Scapy send function of a manually crafted packet can be seen below:

```
send(IP(ttl=255, dst = 127.0.0.1)/TCP(seq = 10000,
         sport = 2000, flags = 0x02))
```

In this example an IP/TCP packet is crafted and sent. All fields in the respective protocol headers have a default defined by Scapy but can be manually set. In the example above, the destination in the IP header of the packet was manually set to '127.0.0.1' with a TTL of 255. In the TCP header the packet a sequence number of 10000 and a source port of 2000 is set. Furthermore in "flags" the SYN flag on the packet was set, to initiate a TCP handshake.

It is possible to manually set most of the packet header values in the crafted packets to observe the evolution of those values as the packet traverses the path. But some values have to be fixed so that a traceroute can be performed correctly. One of course is the value of the TTL field, which will be increased by one for every packet sent out. Another one is the sequence number field of the TCP header. Since the TTL values are changed on the path, the number of the sent out packets has to be stored somewhere. And since the sequence field is very rarely disturbed on the way to the endpoint, this number is encoded there. This works like this: the first packet with TTL zero starts with a fixed predefined sequence number, if the next packet with a TTL of 1 is sent, the sequence number is also increased by one and so on. If an ICMP packet is captured, the initial TTL of a packet can then be established by looking at the sequence number of the encapsulated TCP packet.
Since there will often be multiple routing paths leading to an endpoint it often is necessary to repeat a traceroute multiple times to make the different paths visible. For this to work, the different traceroute attempts for the same destination must be assigned to different flows. Since the flow number of the flows, the packets belong to, is composed of the source and destination IP and the source and destination port seen in the packet headers, the source port of the sent out TCP packet is used to distinguish between different traceroute attempts. For every additional traceroute performed to the same destination, the TCP source port in the packet header will be increased by one, starting from a fixed initial port number.
An additional part of the TCP header that should not be disturbed if a response from the destination host is required are the TCP flags. The TCP handshake will not be initiated if they are not correctly set. If the interest lies solely in the path up to the last hop before the destination, the setting of the TCP flags doesn't matter.
Finally, if all packets required for performing the traceroutes have been sent out, the module can be shut down by putting a shutdown sentinel into its input queue.

3.1.2 Traceroute Chain

The traceroute chain is a newly written function chain that does analysis on the outgoing traceroute TCP packets and incoming traceroute ICMP packets captured by the observer. For the outgoing packets, the chain looks at the sequence number and stores it in the flow the packet was assigned to. This entry also contains the measured time the outgoing packet was seen by the observer and a byte array with the contents of the whole packet.
For incoming ICMP packets, the chain looks at the original TCP sequence number embedded in the ICMP message. The number of the hop on the path where the packet is received from is then calculated by subtracting the fixed initial sequence number from the sequence number observed in the packet. For this hop number, an entry in the flow is made and assigned to this will also be the IP of the host, the time the packet was observed, the length of the TCP header embedded in the ICMP and also a byte array containing the payload of the ICMP. This basic traceroute information is stored for every performed traceroute. For measurements of individual protocol or protocol extensions, there exist the possibility to perform additional analysis by a 'subchain' that records the status of flags or header fields important for the specific measurement. These recordings are added to the flow entry in the 'conditions' section of the respective hop number.
The function chain does not only analyze ICMP messages but also possible TCP responses from the traceroute target. Besides the IP, also the number of hops to reach the destination is recorded. Since the acknowledge number of a TCP SYN/ACK corresponds to the sequence number of the TCP SYN increased by one, the hop number on the path can be calculated by subtracting the initial fixed sequence number. The value of the acknowledge number is also used to guarantee that only the first TCP response will be analyzed. This is important because traceroute will send more than one packet that reaches the destination, especially when the initial hop number to reach the target is not known. Endpoints occasionally ignore packets or respond differently to subsequently sent TCP SYN packets. For the Analysis of the TCP response header there also exists the possibility to do a protocol specific analysis with the subchain and add those results to the respective flow entry.

3.1.3 Trace-merger Module

The trace-merger module is responsible for merging the information collected by the traceroute chain and stored in the flow queue. This is done by comparing the stored sequence numbers of the sent TCP packets with the stored hop numbers of the received ICMP packets. If a match is found, the timestamps in the entries are used to calculate the round-trip time between sending the TCP packet and receiving the respective ICMP packet. Additionally, a bytewise comparison of the stored byte arrays containing the initial TCP packet and the contents of the observed ICMP packet is done. The differences between this arrays will be recorded and allows for a detection of a manipulation of the packets headers. Finally, the trace-merger forwards the calculated information and the information collected by the traceroute function chain to the output.

3.2 Output

The output of the traceroute extension is provided in a NDJSON formatted file, similar to the output of the basic PATHspider. An example can be seen in Figure 3.1.

```
{
"sip": "188.166.56.229",
"dip": "172.217.18.99",
"1": {…},
"3": {…},
"4": {…},
"5": {"from": "84.116.134.142",
      "size": 120,
      "rtt": 31.672,
      "data": "['8: 4', '10: 4']"}
      "conditions": ["ECE.set", "CWR.set",
                     "ecn_no_ect", "DSCP: 0"]}
"6": {…},
"Destination": {"from": "172.217.18.99",
                "hops": 7,
                "conditions": ["ECE.set", "CWR.notset",
                               "ecn_no_ect", "SYN.set",
                               "ACK.set" , "DSCP: 0"]}
}
```

Figure 3.1: Output of the traceroute extension of PATHspider

Besides the source and destination IP, there exists an entry for every observed host that responded with an ICMP message, identifiable with its respective hop number. In this example the hop number "2" doesn't exist because frequently, no ICMP messages are received from certain hosts. For every host on the path to the destination, the IP, the size of the returned packet in bytes and the calculated round-trip-time in milliseconds is given, as can be seen in the example of hop "5".

The entry "data" stands for the result of the bytewise comparison of the original TCP packet and the contents of the returned ICMP packet. The first number of the "data" list entry stands for the byte in which the difference was detected and the second number for a decimal representation of the bitwise difference in the respective byte. The entry in the example in hop "5" indicates, that there were differences detected in the TTL field and the first byte of the checksum field. This difference of course occurred, because the initial TTL was 5 and the TTL in the returned packet was reduced to 0 while being routed through the Internet.

The "conditions" entry shows protocol specific information on a higher semantic level. The entries in that list are assembled by the subchain in the traceroute extension. In the example, the status of flags important to ECN and the DSCP value observed in the returned ICMP packet are included.

As a last entry in the output there is a line called "Destination". Whenever a TCP response from a the destination IP is received, this line is added to the output. It contains a calculated number of hops, based on the TTL value of the received TCP packet. Also included are the same conditions like in the intermediate host responses plus the status of the SYN and ACK flags to reveal if a correct response to the TCP SYN was provided.

3.3 Traceroute command

There are two ways to perform a traceroute in PATHspider, the first one is with the traceroute command. With this command, the sender, the trace-merger and the observer modules are started in separate threads. The sender can take a single IP as an input or a NDJSON file that contains "dip" keys with the respective IPs, like it is the case with the output file of the basic PATHspider. Traceroute performs a lot faster if there is also a "hops" key provided for every target, which includes the number of hops to reach the target. If no such value is provided, traceroute will assume a hop number of 50. The observer, the trace chain and the trace-merger will perform their assigned function until the sender is done with sending the packets. If this occurs, the modules are shutdown and the results written to the output.

The traceroute command with all possible options can be seen here:

```
pspdr traceroute --flows f --cond c --ip p
                 --input i --output o
```

The number of traceroutes per target that should be performed can be chosen with the optional argument `--flows`, the default value is 3. If a value to the argument `--cond` is given, entries in the input file are searched for this value in the "conditions" section. The traceroute for the target will then only be performed if the provided value from the traceroute command matches the value in the input file. If no condition argument is provided, traceroute will be performed for all targets in the input file. With the `--ip` argument, a single IP can be provided for tracerouting instead of an input file. In the last two arguments an input file and an output file can be defined. If no output file is chosen, the output will be directly displayed in the console.

3.4 PATHSpider measurement work flow

The second method of performing a traceroute with PATHspider is directly during a PATHspider measurement. For the PATHspider "measure" command, an optional argument `--trace` was added, which triggers a traceroute for the tested targets.
For this to work, some adjustments to the original PATHspider work flow had to be made. A simplification of the PATHspider measurement work flow can be seen in Figure 3.2. The modules and queues depicted in black are the most important ones from the original PATHspider, the red ones were added as part of

the traceroute extension and have been described at the beginning of this chapter.

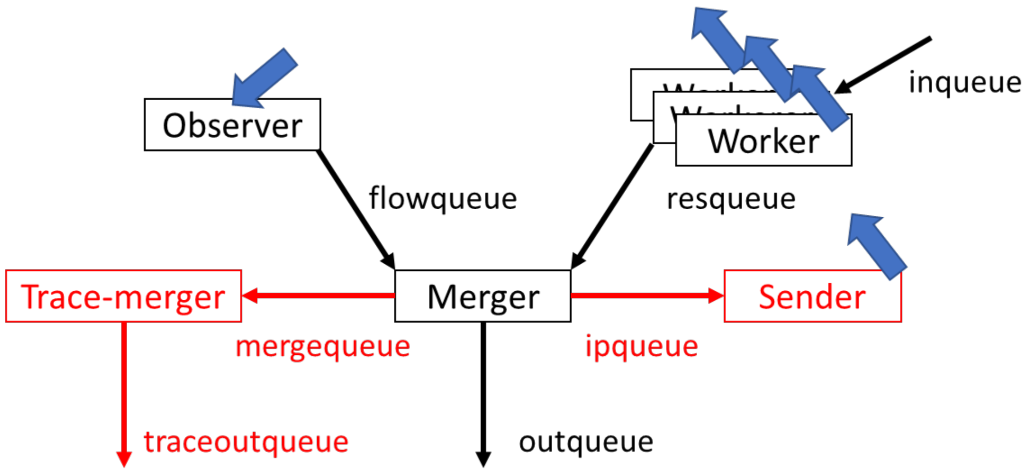

Figure 3.2: Extended PATHspider measurement work flow

When a flow from the *flowqueue* and a result from the *resultqueue* have been successfully merged, conditions concerning the success of the connection are assigned. As soon as the merging of a job is complete, a traceroute for this target can be performed if a certain condition is observed. The conditions, for which a traceroute should be conducted have to be defined in the measurement plugin in the `traceroute_conditions` list. The merger checks if merged jobs fulfill any of these conditions and if they do, the IP of the job and the number of hops to reach the destination are added to the *ipqueue* which serves as the input for the sender module.

If a flow contains ICMP packets of the type "TTL exceeded", it is marked by the basic function chain. The merger checks for marked flows and if one is detected it is forwarded to the *mergequeue* from where on the trace-merger will handle the flow.

In the basic PATHspider, as soon as the workers are finished, the observer is shutdown and any unobserved flows, that could not be merged are pushed out before the merger is shutdown. With the extension, when the workers are shutdown an intermediate sentinel will be sent to the merger. This causes the merger to push out the unobserved flows and shutdown all functions except the one checking for and forwarding the flows containing ICMP packets. If the conditions require it, a traceroute can now be conducted for the unobserved flows, before the sender and the observer are finally shutdown.

Chapter 4

Graphical Visualization with Neo4j

The amount of data obtained by the PATHspider traceroute extension can be huge if performed for thousands of hosts. Graph databases can be used to visualize and analyze vast amounts of data. This is why the database tool Neo4j was evaluated for the use with traceroute data. The community edition of the database is freely available at `https://neo4j.com`

4.1 Neo4j

Neo4j is a graph database management system. It is specifically optimized to store and traverse graphs of connected data. The data is thus stored in nodes and the nodes are connected by edges, called relationships. Neo4j introduces its own, SQL-inspired query language called Cypher to match patterns of nodes and relationships in the graph. Import of data into the database is done with a Comma-separated values (CSV) file and Neo4j also supports easy integration of external graph visualization tools. Visualization is also possible directly in the default Neo4j server, which is a local server that can be accessed via the browser and has been used for this work. To work with the data, there are officially supported drivers for almost every popular programming language available. The work in the scope of this thesis has been done in Python.
To get an idea of the structure on how data is stored, an example is displayed in Figure 4.1. Every node in Neo4j has a label which indicates the type of the node. A node type could for example be a person or a country like depicted. Additionally, every node has a number of properties, all the nodes with the same label have the same properties but with individual values. The relationships that connect the nodes are also classified by labels which should be appropriate to the types of nodes they connect. An example for a label for a connection between a person labeled node and a country labeled node could be "LIVES_IN".

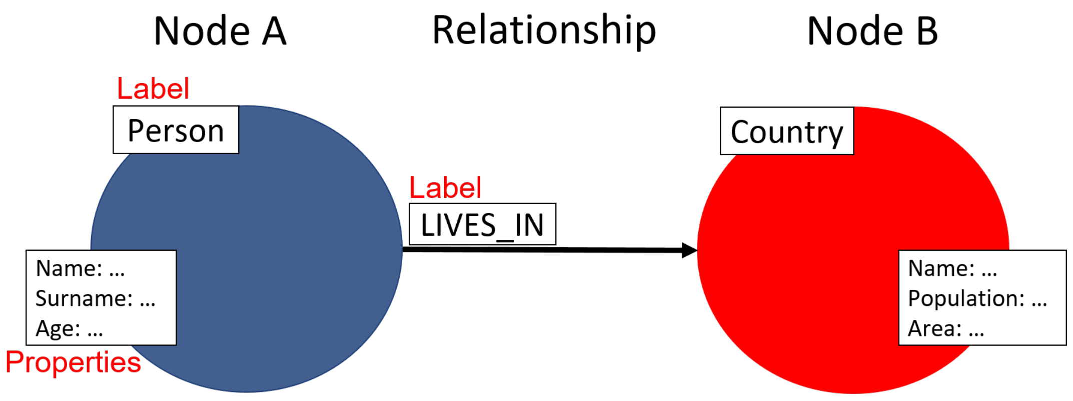

Figure 4.1: General node and edge setup in Neo4j

The Cypher language is built to be easily human readable and focuses on *what* to retrieve and not *how* to retrieve it. An example of a simple query concerning the person-country example can be seen here:

```
MATCH (:person {name:'Michael'})-[:LIVES_IN]->(c:country)
RETURN c.name
```

This query returns the names of all countries where a person named Michael lives in. In Cypher, nodes are represented by parentheses, properties by curly brackets and relationships with square brackets. The connection between the nodes are represented by dashes and arrows. This query matches all nodes labeled person that have the name "Michael" and are connected through a relationship of type "LIVES_IN" to a country node. Returned is then the name of the country.

4.2 Displaying traceroute data with Neo4j

A basic network of traceroute data consists of 3 types of nodes: the test servers , the intermediate hosts on the path and the endpoints, the destination servers. In Neo4j this leads to three labels assigned to the three types of nodes: start, host, and end. The connections between all those nodes are represented by relationships labeled "CONNECTS_TO".
The traceroute data is loaded into Neo4j with two CSV files, one containing the connections and an optional one containing additional information concerning the nodes. In the first file, all the connections in the network are represented by a tuple of connected IPs on a line. When loading the file, first a node is generated for every IP there doesn't already exist one. As a property to the node the

respective IP is assigned and a relationship between the two nodes in the CSV line is created. After the database is filled with the data, the starting points and the endpoints of the graph are identified and their labels are switched from "host" to "start" and "end" respectively. The network is now complete. With an additional CSV file more properties can be added. The CSV file just has to contain the IP address of the node as a first value, followed by an arbitrary number of properties.
A small example on how the visualization of traceroute data looks like can be seen in Figure 4.2. The raw data for this Visualization is a traceroute performed for the following 3 websites depicted as the yellow "end" nodes on the left side: www.ethz.ch (top), www.google.com (middle), www.pathspider.net (bottom). The graph shows all paths leading from the red start node on the right to the targets. In this example the traceroute has been conducted three times for every web server, such that the different paths created by loadbalancing can be made visible.

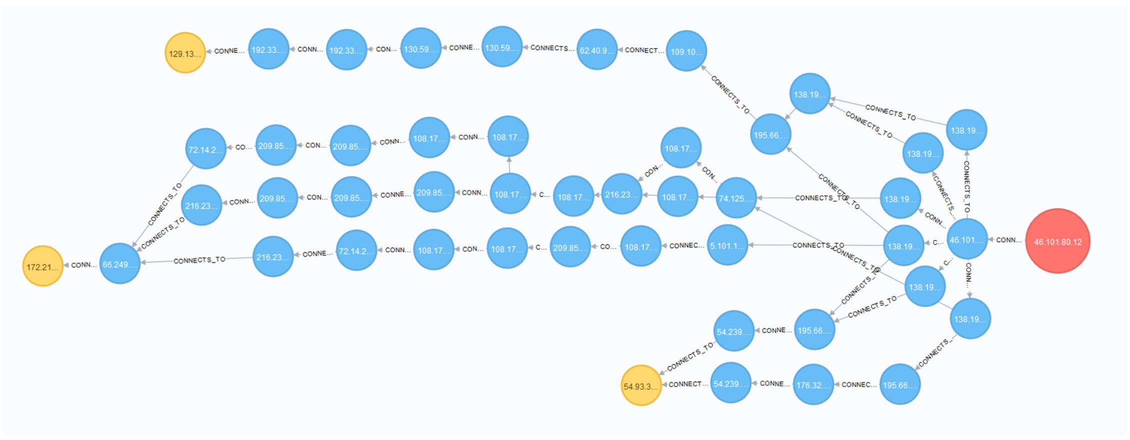

Figure 4.2: Neo4j traceroute visualization

4.2.1 Autonomous System (AS)

Having a large network with nodes can be confusing and it can be hard to read anything meaningful out of it. A way to reduce the complexity of the graph is not to focus on every individual node but only on the autonomous system (AS) the nodes belong to. Because IP addresses that belong to an autonomous system are under the control of a single entity and share a common routing policy, the network analysis on an AS-level can already give insight into some of the problems occurring in the Internet.
The goal was the integration of the AS architecture into the existing traceroute graph database, not only as a simplification of the network but also as an ad-

ditional parameter for analysis. To achieve this, a request for the autonomous system for all IP addresses in the network to RIPE NCC, the Regional Internet Registry for Europe is made. The respective Autonomous System Number (ASN) is added as a new property to the existing nodes. For every additional autonomous system returned by the request, a new node with the label "as" is made with the ASN as a property. The AS node is connected to all host nodes that contain the respective ASN as a property by a relationship labeled "AS_CONNECTS_TO". With this addition of more nodes and properties the networks grows more complex but by selecting only parts of the network for visualization the complexity can actually be reduced.
An example on how the network of the previous section seen in Figure 4.2 does look on the AS level, can be seen in Figure 4.3. The AS nodes are displayed as pink nodes with their respective AS numbers. The AS request is not successful for all IP addresses as can be seen with the blue host nodes displayed on the right hand side of the picture. The IP addresses belonging to the routers that are used for loadbalancing inside the network of the data center from where the measurements were performed from are not publicly announced and can thus not be assigned to an autonomous system. But nevertheless, a comparison between the graph network generated in this example and the one in the previous section shows a more compact visualization of the traceroute data on the AS level.

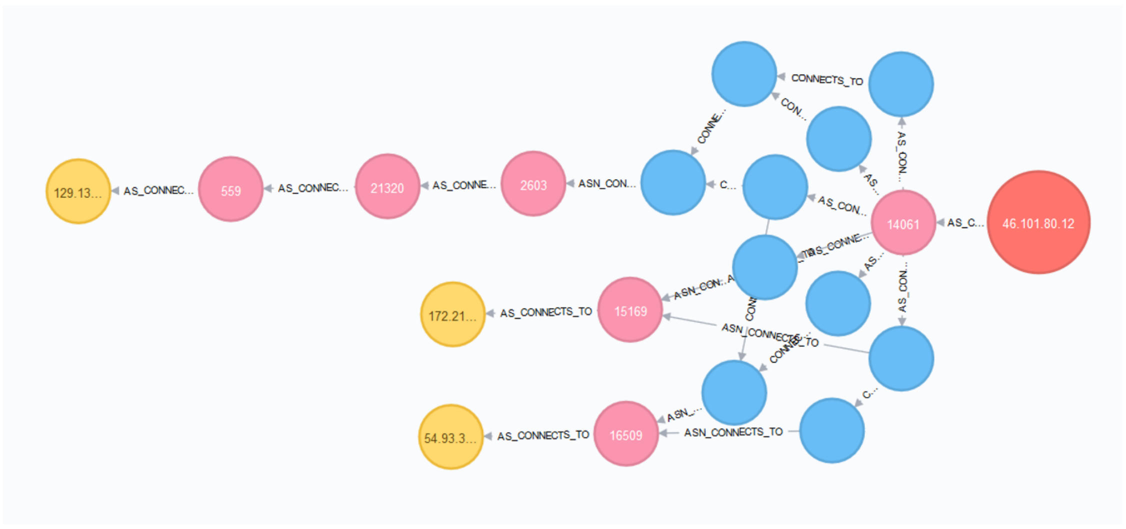

Figure 4.3: Neo4j traceroute visualization on the AS level

4.3 Conclusion on Neo4j

As demonstrated in the previous two sections, visualization of the data collected with the PATHspider traceroute extension works well with small datasets. Some problems occurred when the attempt was made to display the traced paths of hundreds of targets inside the browser. Another problem with the visualization inside the browser is created by the fact that nodes can only be displayed in different colors depending on their label. This is not very practical if the goal is to display the influence of a certain property on the network. For both problems the use of an external visualization tool supported by Neo4j could help.
The easy use of the Cypher language made up for those minor issues in visualization. The example in section 4.1 showed well, the easy and intuitive use of the query language for data analysis. What the language is not made for is manipulation of the data that exceeds some simple cases. For this it is necessary to use one of the drivers provided by Neo4j which allows for combining the query language with another supported programming language.
Neo4j is a good tool to use in combination with the traceroute data. While the visualization can give an overview of the collected data and help to understand it, the analysis of the data can be speed up extensively with the use of Cypher queries.

Chapter 5

Measurements and Results

5.1 General Experimental Setup

All the measurement targets for which measurements have been conducted in this chapter, are from the Alexa 1 million most visited websites list, obtained on the 30.10.2017. For all websites a DNS resolution was made and if the resolver provided more than one IP for a website then all were taken into the list independent of being IPv4 or IPv6. After filtering out the duplicates, a target list of 751064 IP addresses resulted. All measurements were conducted in November 2017, using the extended PATHspider tool, as described in Chapter 3 and they were made from servers of the cloud infrastructure provider Digital Ocean located in Amsterdam and London.

5.2 State of ECN in the Internet

The goal of the first experiment was to get a general overview on the state of ECN in the Internet. The measurement was done with the ECN plugin provided together with the PATHspider tool. The 'A' test attempts a basic connection to a web server with the ECN option in the Linux Kernel set to default. This means enable ECN when requested by incoming connections but to not request ECN on outgoing connections. In the 'B' test, the ECN option is changed so that ECN is also requested on outgoing connections. This causes the CWR and ECE flags of the TCP header on the SYN of outgoing TCP packets to be set. The different conditions the plugin assigns to jobs dependent on the connections success can be seen in Table 5.1. For the cases where the ECN connectivity works, the plugin additionally determines if the ECN negotiation attempt was answered correctly with the ECE flag set on the SYN/ACK and marks it with ecn.negotiation.successful. If the flags are reflected, meaning the host answers with the CWR and ECE flag set, the flows are marked with ecn.negotiation.reflected. If no TCP ECN flags were set at all on the SYN/ACK, the condition ecn.negotiation.failed was assigned.

Condition	**Basic (A test)**	**ECN-enabled (B test)**
ecn.connectivity.works	works	works
ecn.connectivity.broken	works	failed
ecn.connectivity.transient	failed	works
ecn.connectivity.offline	failed	failed

Table 5.1: ECN connectivity conditions assigned by the ECN plugin

The results concerning the connectivity tested by 3 runs of the Alexa list on different weeks in November 2017 can be seen in Table 5.2. Because the Alexa list was generated on October 30, 2017 a decline in the successful connections from 95.39% in the first measurement to 93.82% in the last measurement can be seen, as some portion of the websites did go offline during that time, especially some from the rear part of the list. This example shows well that the measurements are most reliable when done shortly after acquiring the Alexa target list.

	1. run	**2. run**	**3. run**
Works	716419 (95.39%)	708196 (94.23%)	704636 (93.82%)
..of which IPv6	92008 (12.25%)	91577 (12.93%)	89966 (12.77%)
Broken	1608 (0.21%)	1757 (0.23%)	3331 (0.44%)
Transient	2592 (0.35%)	2665 (0.35%)	2829 (0.38%)
Failed	30445 (4.06%)	38441 (5.12%)	40267 (5.36%)

Table 5.2: ECN Connectivity

It is worth to have a closer look at the connections that break if ECN negotiation is enabled. This means that packets are discarded if the TCP ECE and CWR flags are set and this was a big problem in the early days of ECN. To establish an exact number of times this occurs, only the IP addresses that broke in all 3 runs were further analyzed. This process filters out the unstable connections where it is hard to tell if ECN is the problem or the connectivity per se. Of the initial 1600 to 3300 broken hosts only 66 IP addresses remained after this filtering. For those 66 targets traceroutes with and without ECN enabled were performed as a confirmation. Of the 66 only 8 were confirmed to always drop packets with the ECN TCP flags set. Additionally, one endpoint always answered with a RST to a received packet with ECN flags. Furthermore, the path lengths acquired in the traceroute were compared between the two settings and no differences could be found. This means that the packets didn't get dropped on the path to the destination but always after the last recorded hop. It can be concluded that ECN packet dropping is no longer a big issue, since a lot of network operators seem to have reconfigured their equipment. There will probably be more than those 9 servers that have problems with ECN but for most of them the connection is very unstable and this makes it hard to establish the cause of the packet dropping.

Besides the connectivity of ECN enabled packets, the rate of successful ECN negotiations is also interesting to observe, as it is an indicator on the state of the deployment of ECN in the Internet. The success rates of the negotiations of the three PATHspider runs can be seen in Table 5.3. Over all three measurements, there is a successful negotiation rate of at least 82.2%. Interesting is the drop of negotiation from the first run to the third, it seems that more ECN capable web servers did go offline than web servers that didn't support ECN. Striking is the very high ECN support amongst hosts running IPv6 with a rate of nearly 97%. There is also a very stable amount of hosts that do not engage in ECN and reflect the ECN flags, but the amount of those hosts is small with only 0.26% of the cases.

	1. run	**2. run**	**3. run**
Working	716419	708196	704636
..of which IPv6	92008 (12.25%)	91577 (12.93%)	89966 (12.77%)
Negotiated	589821 (82.33%)	582595 (82.26%)	579200 (82.20%)
Negotiated (IPv4)	500585 (80.17%)	493863 (80.10%)	492045 (80.05%)
Negotiated (IPv6)	89236 (96.99%)	88732 (96.90%)	87155 (96.88%)
Reflected	1858 (0.26%)	1817 (0.25%)	1819 (0.26%)

Table 5.3: ECN code point in the IP header

The negotiation data obtained in this measurement confirms the growing appreciation of ECN in the last years. In Figure 5.1 the fraction of top 1Million Alexa websites supporting ECN over the years is shown, with data from [13],[14],[15],[5],[4],[16],[17]. The scatter plot shows an almost linear increase in the support of ECN since 2008 with a growth of more than 5% per year. The increase of the servers supporting ECN is expected to slow down in the future as ECN nears full deployment in the Internet.

5.3 Setting ECN code points on the TCP SYN

The goal of this measurement was to demonstrate the capability of hosts to negotiate ECN when an ECN IP code point is set. Currently it is not allowed to set IP ECN code points on TCP control packets like the SYN packet. So it is interesting to observe if there would be a difference in the negotiation of ECN. For this measurement a new plugin for PATHspider was written. The plugin is

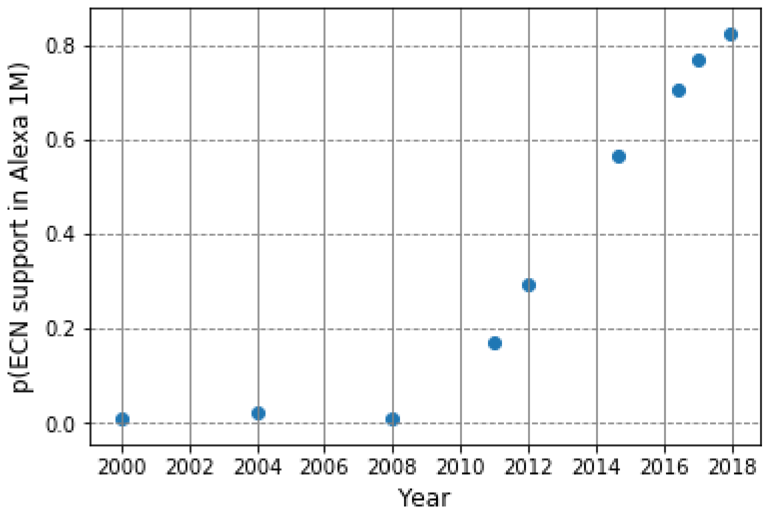

Figure 5.1: Trend of ECN capable hosts

called "ecnflags" and sets up 4 connection configurations: One without ECN, one with ECN but without any flags set and then 2 configurations with the code points ECT(0) and CE set respectively. The flow combinator compares the connection results and assigns the conditions according to Table 5.4. The Table displays all conditions assigned dependent on the negotiation success according to the possible code points set. If any other combination was observed, the condition "ecn.negotiation.transient" was assigned.

Condition	**non-ECN**	**ECT(0)**	**CE**
ecn.negotiation.failed	fails	fails	fails
ecn.negotiation.successful	works	fails	fails
ecn.negotiation.ect0.successful	works	works	fails
ecn.negotiation.ce.successful	works	works	works
ecn.negotiation.transient	maybe	maybe	maybe

Table 5.4: ECN negotiation conditions

Theoretically, PATHspider should be able to handle more than two configurations but this experiment showed that there are some issues where PATHspider crashes when more than one worker is used. This is why the measurement could only be conducted with one worker and because this leads to a greater time consumption, the Alexa top 1 million list was reduced to 50'000 entries for this measurement. In the initial measurement the code point ECT(1) was also observed, but since there seemed to be no real difference to the behavior when code point ECT(0) is set, the examination of this code point was dropped to enhance

performance.
Another issue was that connections that didn't respond weren't handled in the right way. This is why the only thing that is clear about the ECN connectivity with ECN code points is that 95,59% of all connections were successful independent of the code point set. This corresponds well with the number of working connections seen in the first measurement. This doesn't guarantee that no packets are being dropped because of certain code points being set but at least it is safe to tell that it is not a widespread problem.

The findings concerning the success of the negotiation dependent on the code points can be seen in Table 5.5. The number of hosts that don't negotiate ECN also fits very well to the numbers seen in the first experiment. The insight we gain with this experiment is that 69,35% of all hosts only negotiate ECN when no code point is set. 12.79% of hosts negotiate ECN no matter what code point is set on the TCP SYN. There are very little endpoints that only negotiate ECN when ECT(0) is set but not when CE is set (0.05%) or on any other combination of code points (0,19%). 141 hosts didn't get a condition assigned for the negotiation, this is caused by an overload of PATHspider despite of only one worker being used.

Condition	# Hosts	Percentage
ecn.negotiation.successful	33145	69,35%
ecn.negotiation.failed	8280	17,32%
ecn.negotiation.ce.successful	6112	12,79%
ecn.negotiation.ect0.successful	26	0,05%
ecn.negotiation.transient	89	0,19%
No condition assigned	141	0,30%

Table 5.5: Negotiation success with different ECN code points

The amount of "ecn.negotiation.successful" conditions assigned will be slightly higher in this experiment than in reality. The reason for this is that some code points will be erased on the path and will thus not be seen by the target. The extent of this problem will be covered in the next section.
These measurements show that the rate of negotiation drops dramatically when code points are set. Since there are attempts made in the Internet community to allow for setting code points during the TCP handshake in the future [12], these findings should be taken into consideration.

5.4 ECN and DSCP Manipulation on the Path

The goal of this measurement is to demonstrate the ability of the PATHspider traceroute extension to detect packet manipulation on the path to a host. For this the contents of the IP differentiated services field were controlled and observed.

5.4.1 Measurement Setup

For this measurement the Alexa top 1 million websites list has been used again, but this time it was reduced to IP addresses that have a unique prefix of length 24 for IPv4 or length 64 for IPv6. This decision was made because conducting traceroutes takes a long time and IP addresses with the same prefix of length 24 lie in the same collision domain and will thus probably share the same path.
For these remaining 223367 targets a traceroute with the newly implemented traceroute extension of PATHspider has been conducted. To observe manipulation of the differentiated services field, DSCP has been set to 46 (Expedited Forwarding) and for ECN, ECT(0) (ECN capable) was set on the traceroute TCP SYN packet. The ECE and CWR flags in the TCP header for ECN negotiation were not set. As an extension to the traceroute function chain described in subsection 3.1.2, a subchain called "trace_ecn" was developed. This subchain looks at the ECN code points and DSCP in the IP header, as well as the TCP ECN flags and stores them in a conditions list.

5.4.2 General Results

Of all the traceroutes performed, 90.37% or 201854 hosts gave back a TCP response to the TCP SYN request, most of them a correct SYN/ACK response. 263 hosts responded with a RST flag to the SYN. The tracerouting was repeated for these hosts with an empty differentiated services field. The measurements yielded the same result, meaning the contents of the differentiated services field has no influence on the correct response to the TCP SYN.
For further analysis, only the traceroutes with a TCP response were taken, since for the others there is no guarantee that the whole path has been traced.

In Subsection 2.2.2 it was mentioned that older routers often only include the first 8 bytes of the TCP packet in the ICMP message. The amount of data collected with these traceroutes gives a good opportunity to look at the number of routers that include the full TCP packet, compliant with RFC 1812. Of all performed traceroutes there were 172510 unique hosts observed that responded with an ICMP message. Of those hosts, 46,41% or 80062 included the full TCP packet in their response. An examination of the deployment of

RFC1812-compliant routers was made by Detal et al. [2] in 2013. In that paper, the proportion of RFC1812-compliant routers on the paths to the top 5000 Alexa web sites from 72 vantage points was measured.
As a comparison, Figure 5.2 shows the proportion of RFC1812-compliant routers in the paths of the current measurement as a CDF. The value 0 on the horizontal axis stands for paths that contain no compliant routers, the value 1 stands for paths made solely of RFC1812-compliant routers. The measurements revealed, that 12,3% of all paths contain no compliant routers, compared to 20% of paths in the study from 2013. The difference is even bigger if we look at paths where less than 40% of the routers are RFC1812-compliant: In 2013 more than 80% of the measured paths consisted of less than 40% compliant routers. Now in 2017 this amount has roughly halved, as less than 45% of all paths have a proportion of less than 40% compliant routers. With the deployment of new routers, this amount is likely to rise further in the next years.

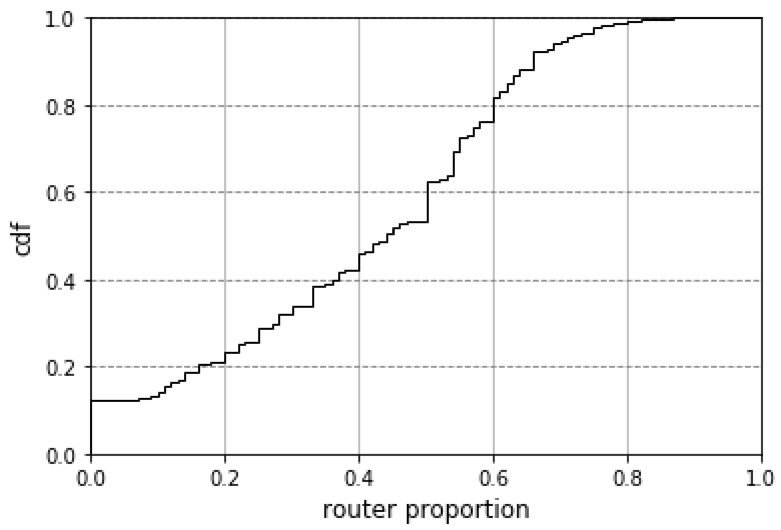

Figure 5.2: Proportion of RFC1812-compliant routers on a path

Back to the question if the contents of the differentiated services field are manipulated on the path. To answer this question, the state of the field in the last recorded hop was considered. It is important to keep in mind, that the information that reaches the endpoint hasn't necessarily to be the same as in the last recorded hop. It may be that there are hosts and middleboxes in between that further manipulate the packets. The problem with loadbalancing in the Internet is, that there is a possibility that header information is different, depending on the path a packet was routed trough. To reduce the impact of last hops that are shared by multiple endpoints on the result, last hops that are shared are only considered one time. This resulted in a total number of 80995 unique last hops.

TCP response	201854	100,00%
Nonce (00)	198562	98,37%
ect0 (01)	3252	1,61%
ect1 (10)	22	0,01%
ce (11)	18	0,01%

Table 5.6: Observed ECN code points in the TCP response

5.4.3 ECN

The results of the traceroute concerning the ECN part of the differentiated services is examined in this Subsection. Not only will the state of the code point on the last hop be analyzed but also the code point observed in the TCP response from the destination host.

TCP Response

Table 5.6 shows the code points observed in the TCP response of the web servers. The vast majority of servers (98.37%) responded with an empty ECN field, this is the expected response, since ECN was never negotiated. Not a negligible amount of servers (1.61%) reflected the ECT(0) and some servers even responded with an ECT(1) or CE. Since setting the ECT(0) code point on the TCP SYN is not allowed currently, the response to the ECN code point has no real consequences but is interesting nonetheless.
Since no ECN negotiation attempt was made, one would assume that all servers respond with the TCP ECN flags unset. Inspection of the TCP ECE and CWR flags revealed that a tiny amount of misconfigured servers didn't respond as planned. Four servers responded with an ECE flag set, one with CWR set and 2 endpoints responded with both ECE and CWR set.

Last Hop

Even more interesting is the analysis of the ECN code points on the last hop. If no packet manipulation on the path would exist, ECT(0) would be observed on all lasts hops. But as can be seen in Table 5.7 which shows all code points observed on the last hops, this is the case only for 97.37% of the last hops. In 2.63% of the cases the ECN code point has been erased.

Unique last hops	80995	100,00%
Non-ecn (00)	2131	2,63%
ect0 (01)	78864	97,37%

Table 5.7: Observed ECN code points on the last hop

An inspection of the position of the code point removal with Neo4j revealed that the code points get erased mostly in the last couple of hops but there was at least a case were the code point was erased at the border to an AS. This has the consequence that the code point was removed for all web servers hosted within this AS. An exact analysis on where the code points get removed on the path shows Figure 5.3. For all paths where the ECT(0) code point was erased on the last hop, the hop on which the erasure occurred was divided through the number of hops on the path and the relative position of the host in the path cumulated. From the resulting graph can be learned that 50% of the code point removal happens in the last hop and in more than 90 % of the cases in the last 40% of the path. Figure 5.3 shows that the position of the code point manipulation happens close to the host and not close to the core of the Internet.

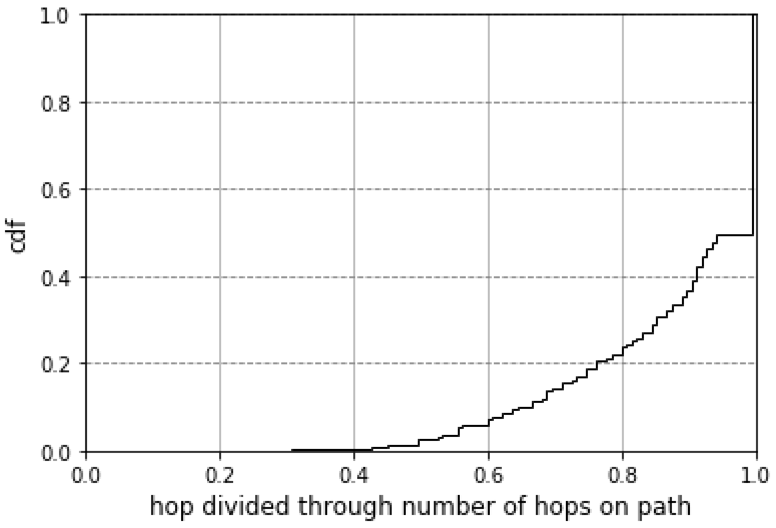

Figure 5.3: Position of ect code point removal on path

Like mentioned before, since setting ECT(0) on the SYN packet is not allowed, the removal of these code points does not have any consequences on the ECN functionality. But currently there are attempts made to advance ECN in a way that code points on TCP control packets, like the SYN will be allowed[12]. If the decision is made to officially allow it, ECN could break for all these destinations where the ECN code point gets erased on the path. But that's not all, out of curiosity, an additional traceroute was conducted on the destinations that contained a last hop where the code point was erased. This time TCP packets without a SYN and with the same ECN code point set were sent. The results show that approximately 2/3 of the last hops still had the ECT(0) erased. This means that the code point is erased on these paths independent of the TCP packet being a control packet or not. This basically breaks the ECN functionality for all endpoints that lie behind the device that manipulated the code point, even if ECN has been successfully negotiated between the sender and the receiver. This is a very good example of a problem that can't be detected by the basic PATHspider tool but only by conducting a traceroute.

5.4.4 DSCP

After analyzing the results of the ECN code point, the question arises if there can be seen a similar behavior on the last hop when inspecting DSCP.

Last Hop

When analyzing the DSCP on the last hop of the path, a big difference to the ECN code point can be seen: the DSCP on the last hop is mostly not the initial one anymore. Table 5.8 shows a detailed listing of the code points observed on the last hop if the code point has at least 0,02% share. The initial DSCP of 46 propagated through the path to the last hop in only 16,83% of the cases. Most observed were code point 6 and code point 0 in roughly one third of the cases each. It is no surprise that precisely those two code points are observed the most. If a packet enters a new autonomous system on the path, the DSCP is often erased because network operators do not trust DSCP coming into their network. This explains why code point 0 is often seen. But also when code point 6 is observed, an attempt to clear the DSCP field was made. The reason the code point was not set to 0 lies in the previous definition of the field. In Table 2.1a the predecessor to DSCP, the precedence field can be seen. This precedence field consisted of the first 3 bits of the present DSCP field. Some devices are not configured for the new field yet, which leads to a removal of the first 3 bits only of the DSCP field. In the case of code point 46 this clearing results in the code point 6.
In the third column of Table 5.8, the official assigned category explained in Subsection 2.3.1 is displayed. It can be seen that a lot of code points observed are assigned to one such function. If the code points are used by the network operators because of their official meaning or for some other reason is hard to tell. What stands out is that all nine asserted forwardings and all 7 class selectors were observed. In total 51 different code points of the possible 64 were observed.

Code Point	# Observed	Percentage	Meaning
6	29820	36,82%	
0	28271	34,90%	Best Effort
46	13633	16,83%	EF
18	2767	3,42%	AF21
8	1410	1,74%	CS1
14	1020	1,26%	AF13
10	656	0,81%	AF11
5	575	0,71%	
24	500	0,62%	CS3
32	366	0,45%	CS4
1	266	0,33%	
22	260	0,32%	AF23
34	228	0,28%	AF41
40	202	0,25%	CS5
2	147	0,18%	
26	145	0,18%	AF31
16	118	0,15%	CS2
59	44	0,05%	
48	44	0,05%	CS6
12	43	0,05%	AF12
63	40	0,05%	
42	38	0,05%	
7	38	0,05%	
30	38	0,05%	AF33
27	38	0,05%	
36	38	0,05%	AF42
17	33	0,04%	
56	32	0,04%	CS7
38	27	0,03%	AF43
9	25	0,03%	
52	19	0,02%	
20	18	0,02%	AF22
28	18	0,02%	AF32
41	15	0,02%	
Various	63	0,08%	

Table 5.8: DSCP values observed at the last hop

Similar to ECN the obtained data was also analyzed concerning the position of the DSCP manipulation. For all performed traceroutes where the code point 46 could not be observed on the last hop, the position in the path where first another code point was recorded. The position was then normalized by division trough the amount of hops in a path. The result of this analysis can be seen in Figure 5.4. The difference to ECN is obvious. While for ECN almost all manipulations happen in the second half of the path, for DSCP more than 70% of all manipulations happen in the first half of the path. It can be concluded that DSCP code point manipulation happens in most cases in the core of the Internet and not near the endpoints.

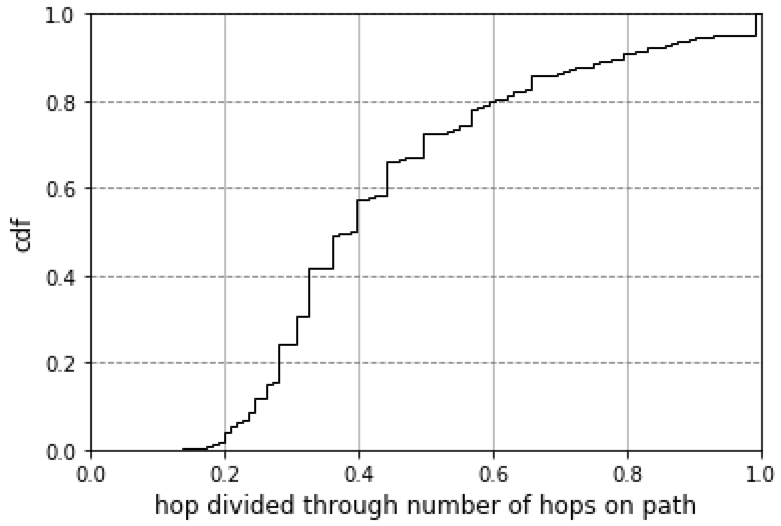

Figure 5.4: Position of DSCP 46 removal on path

TCP response

Now that the behavior of DSCP on the path to the last hop is known, a short analysis of DSCP in the TCP response will be made. In the TCP response of the 201854 hosts, 37 of the 64 different code points were observed. The variability of the observed code points is thus smaller than the ones observed in the last hop. A list of the most common code points observed can be seen in Table 5.9. Not surprisingly, most hosts responded with the default value DSCP 0. While the 1,94% of hosts that responded with code point 6 can be explained with the reflection of the code points observed on the last hop, there is no explanation for the 7.57% of hosts responding with DSCP 2. In general, these values are probably not very accurate since the code points set by the servers were likely manipulated on the path to the test server in a similar way the traceroute packets were manipulated on the path to the target servers.

Code Point	# Observed	Percentage	Meaning
0	179650	89,00%	Best Effort
2	15287	7,57%	
6	3926	1,94%	
8	693	0,34%	CS1
5	398	0,20%	
4	379	0,19%	
1	355	0,18%	
20	244	0,12%	AF22
46	219	0,11%	EF
12	156	0,08%	AF12
32	136	0,07%	CS4
10	93	0,05%	AF11
9	72	0,04%	
18	47	0,02%	AF21
Various	199	0,10%	

Table 5.9: DSCP values observed in the TCP response

Chapter 6

Conclusion

This thesis has shown the importance of performing traceroutes in Internet measurements. The basic PATHspider tool is useful for giving an impression on the behavior of newly deployed protocols or protocol extensions. Once problems are detected, the traceroute extension of PATHspider can be used to inspect the source of the problem. Are packets being dropped on the path when certain flags are set? Are IP fields manipulated on the path and does this happen near the sender or near the destination? These are question that can be answered by the traceroute extension and some of these questions concerning ECN and DSCP have been answered with the measurements conducted in the last chapter. With the traceroute extension, PATHspider has the possibility to do follow-up measurements on detected impairments. The fact that all this can be done in one tool can strengthen the status of PATHspider in the Internet measurement software ecosystem.

Working with the graph database Neo4j on traceroute data works very well. The visualization can be used to gain a quick overview of the collected data and with the use of the query language, in-depth analysis on the data can be conducted. Working with Neo4j has the potential to make analysis of the collected data more efficient because once the data is imported a lot of information can be acquired with simple queries when previously large scripts had to be written to analyze the data.

Further work

Traceroute can be especially useful to locate path dependent problems that can only be seen when doing measurements from different vantage points or when repeating traceroute multiple times. Time did not permit for measurement of ECN or especially DSCP from different vantage points. Another question that arises is how well can traceroute detect and localize problems with other protocol extension like TCP Fast Open (TFO) or Multipath TCP (MTCP).

During the work with PATHspider some limitations on the tool were detected when doing large measurements with multiple workers. There are currently plans to rewrite the tool in the programming language GO which should make the processes faster and more reliable. Concerning the speed of traceroute, the current bottleneck lies in Scapy. While Scapy is easy to use and offers a lot of comfortable functions, it is also very slow. For conducting the traceroutes of the 200000 targets in the last experiment almost a week was needed. To do traceroutes of thousands of targets more efficiently, further work is required that aims in replacing Scapy with a faster tool.

As for Neo4j, the database tool will have to confirm its usefulness in future measurement campaigns. The measurements should be conducted from different vantage points and for different protocol extensions, to test in which situations it can be helpful to use Neo4j. Also, since traceroute provides a huge amount of data, visualization of the data in browser was pushed to its limit. Future projects using Neo4j should consider using external visualization tools.

Bibliography

[1] I. Learmonth, B. Trammell, M. Kühlewind and G. Fairhurst, PATHspider: A tool for active measurement of path transparency, In *Proceedings of the 2016 Applied Networking Research Workshop*, pages 62-64, July 2016.

[2] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanabel and B. Donnet, Revealing middlebox interference with tracebox, In *Proceedings of the 2013 Internet Measurement Conference*, pages 1-8, October 2013.

[3] M. Honda, Y Nishida, C. Raiciu, A. Greenhalgh, M. Handley, H. Tokuda, Is It Still Possible to Extend TCP?, In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurment Conference (IMC)* 2010.

[4] M. Kühlewind, S. Neuner, B. Trammell, On the State of ECN and TCP Options in the Internet, In *Proceedings of the Passive and Active Measurement 2013*, 2013.

[5] B. Trammell, M. Kühlewind, D. Boppart, I. Learmonth, G. Fairhust, and R. Scheffenegger, Enabling internet-wide deployment of explicit congestion notification, In *Passive and Active Measurement Conference*, pages 193-205, New York, USA, 2015.

[6] S. Alcock, P. Lorier and R. Nelson, Libtrace: A packet capture and analysis library, In *SIGCOMM Computer Communication Review*, pages 43-48, April 2012.

[7] J. Postel, Internet control message protocol, Internet Engineering Task Force, RFC 792, September 1982.

[8] F. Baker, Requirements for IP version 4 routers, Internet Engineering Task Force, RFC 1812, June 1995.

[9] K.Nichols, S. Blake, F. Baker, D. Black, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, Internet Engineering Task Force, RFC 2474, December 1998.

[10] K. Ramakrishnan, S. Floyd, D. Black, The Additions of Explicit Congestion Notification (ECN) to IP, Internet Engineering Task Force, RFC 3169 September 2001.

[11] P. Biondi, "Scapy", see http://www.secdev.org/projects/scapy/.

[12] M.Bagnulo, B.Briscoe, ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets, Internet-Draft, May, 2017.

[13] A. Medina, M. Allman, S. Floyd, Measuring the evolution of transport protocols in the Internet, In *SIGCOMM Comput. Commun. Rev. 35*, pages 37-52, 2005.

[14] A.Langley , Probing the viability of TCP extensions, http://www.imperialviolet.org/binary/ecntest.pdf, 2008.

[15] S. Bauer, R. Beverly, A. Berger, Measuring the state of ECN readiness in servers, clients and routers, In *Proc. of Internet Measurement Conference*, 2011.

[16] B. Trammell, 70% of popular Web sites support ECN, https://mami-project.eu/index.php/2016/06/13/70-of-popular-web-sites-support-ecn/, 2016.

[17] P. De Vaere, Continuous Measurement of Internet Path Transparency, https://pub.tik.ee.ethz.ch/students/2016-HS/SA-2016-68.pdf, 2016.